



ALTAIR

Altair PBS Professional 2024.1

Administrator's Guide

You are reading the Altair PBS Professional 2024.1

Administrator's Guide (AG)

Updated 2/27/24

Copyright © 2003-2024 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "Altair® Access™", "Altair® Control™", "Altair® PBS Professional®", "PBS Pro™", "PBS™", "Altair® Grid Engine®", "Altair Breeze™", "Altair Mistral™", "Altair® Software Asset Optimization™", "Altair® SAO™", "Altair® SAO Predict™", "Altair® Accelerator™", "Altair® Accelerator™ Plus", "Altair® Allocator™", "Altair® Monitor™", "Altair® Hero™", and "Altair® FlowTracer™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to <https://secure.altair.com/UserArea/agreement.html> or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes. This document is proprietary information of Altair Engineering, Inc.

Contact Us

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.altair.com

Sales

hpcsales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Language	Telephone	e-mail
Australia		61 37 068 9972	anz-pbssupport@altair.com
Brazil	Portuguese	55 113 884 0414	br_support@altair.com
Canada	English	1 416 447 6463	
China		86 216 146 9080	pbs@altair.com.cn
France		33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany		49 7031 309 9519	pbssupport@europe.altair.com
India		91 80 66 29 4500	pbs-support@india.altair.com
Japan		81 34 571 1454	pbs@altairjp.co.jp
Malaysia		60 39 212 1216	pbs-support@india.altair.com
Mexico	English	52 557 005 7890	mx-support@altair.com
Russia ###		+49 7031 6208 22	pbssupport@europe.altair.com
Singapore ###		+91 80 66 29 4500	pbs-support@india.altair.com
South Africa		27 21 140 4668	pbssupport@europe.altair.com
South America		+55 11 3884 0414	mx_support@altair.com
South Korea		82 26 105 2473	support@altair.co.kr
Sweden	English	4 646 590 2142	
United Kingdom	English	44 204 519 7852 44 204 519 4195	pbssupport@europe.altair.com
United States	English	1 248 614 2425 1 978 275 8350	
Americas	English		pbssupport@altair.com
Asia-Pacific			
Europe, Middle East, & Africa	English		pbssupport@europe.altair.com

Contents

About PBS Documentation	xi
1 New Features	1
1.1 New Features in This Release	1
1.2 Changes in Previous Releases	2
1.3 Commercial-only Features	17
1.4 Backward Compatibility	17
2 Configuring the Server and Queues	19
2.1 The Server	19
2.2 How PBS Uses Mail	21
2.3 Queues	23
3 Configuring MoMs and Vnodes	37
3.1 About MoMs	37
3.2 About Vnodes: Virtual Nodes	41
3.3 Creating Vnodes	42
3.4 Configuring Vnodes	45
3.5 Deleting Vnodes	53
4 Scheduling	57
4.1 Chapter Contents	57
4.2 Scheduling Each Partition Separately	59
4.3 Scheduling Policy Basics	66
4.4 Choosing a Policy	83
4.5 About Schedulers	93
4.6 Using Queues in Scheduling	103
4.7 Scheduling Restrictions and Caveats	103
4.8 Errors and Logging	104
4.9 Scheduling Tools	104

Contents

5	Using PBS Resources	227
5.1	Chapter Contents	227
5.2	Introduction to PBS Resources	228
5.3	Glossary	228
5.4	Categories of Resources	230
5.5	Resource Types	234
5.6	Resource Formats	234
5.7	Setting Values for Resources	236
5.8	Overview of Ways Resources Are Used	239
5.9	Resources Allocated to Jobs and Reservations	240
5.10	Using Resources to Track and Control Allocation	249
5.11	Using Resources for Topology and Job Placement	250
5.12	Using Resources to Prioritize Jobs	251
5.13	Using Resources to Restrict Server or Queue Access	251
5.14	Custom Resources	252
5.15	Managing Resource Usage	283
5.16	Where Resource Information Is Kept	305
5.17	Viewing Resource Information	307
5.18	Resource Recommendations and Caveats	309
6	Configuring and Using PBS with Cgroups	311
6.1	Chapter Contents	311
6.2	Introduction to Cgroups	311
6.3	Why Use Cgroups?	312
6.4	How PBS Uses Cgroups	313
6.5	Configuring Cgroups	316
6.6	Configuring MPI for Cgroups	350
6.7	Managing Jobs with Cgroups	352
6.8	Managing Oom Killer for Cgroups	353
6.9	Caveats and Errors	353
7	Configuring PBS for Containers	355
7.1	Introduction	355
7.2	The PBS Container Hook	360
7.3	Prerequisites	360
7.4	Configuring PBS for Containers	361
7.5	Caveats and Restrictions	366
7.6	Errors and Logging	366
8	Making Your Site More Robust	367
8.1	Robustness	367
8.2	Failover	367
8.3	Checkpoint and Restart	387
8.4	Reservation Fault Tolerance	401
8.5	Vnode Fault Tolerance for Job Start and Run	403
8.6	Preventing Communication and Timing Problems	410
8.7	Preventing File System Problems	418
8.8	OOM Killer Protection	418

Contents

9	Administration	419
9.1	Specifying Scheduler Username	420
9.2	The PBS Configuration File	421
9.3	Environment Variables	427
9.4	Event Logging	429
9.5	Managing Machines	435
9.6	Managing the Data Service	439
9.7	Setting File Transfer Mechanism	441
9.8	Some Performance Tips	449
9.9	Temporary File Location for PBS Components	450
9.10	Administration Caveats	452
9.11	Support for Globus	452
9.12	Support for Hyperthreading	452
9.13	How To...	453
9.14	Avoiding Memory Hogging by Python Scripts	453
10	Managing Jobs	455
10.1	Routing Jobs	455
10.2	Limiting Number of Jobs Considered in Scheduling Cycle	455
10.3	Allocating Resources to Jobs	455
10.4	Grouping Jobs By Project	457
10.5	Job Prologue and Epilogue	458
10.6	Linux Shell Invocation	463
10.7	When Job Attributes are Set	464
10.8	Job Termination	466
10.9	Job Exit Status Codes	470
10.10	Rerunning or Requeueing a Job	472
10.11	Job IDs	473
10.12	Where to Find Job Information	473
10.13	Job Directories	474
10.14	The Job Lifecycle	478
10.15	Managing Job History	480
10.16	Environment Variables	483
10.17	Adjusting Job Running Time	483
10.18	Managing Number of Run Attempts	484
10.19	Managing Amount of Memory for Job Scripts	484
10.20	Allowing Interactive Jobs on Windows	484
10.21	Releasing Unneeded Vnodes from Jobs	487
10.22	Tolerating Vnode Faults	487
10.23	Managing Job Array Behavior	488
10.24	Recommendations	488

11 Security	489
11.1 Configurable Features	489
11.2 User Roles and Required Privilege	489
11.3 Using Access Control Lists	492
11.4 Authentication for Daemons & Users	508
11.5 Encrypting PBS Communication	517
11.6 Restricting Execution Host Access	521
11.7 Changing the PBS Service Account Password	522
11.8 Paths and Environment Variables	523
11.9 File and Directory Permissions	523
11.10 Root-owned Jobs	524
11.11 Passwords	524
11.12 Windows Firewall	525
11.13 Logging Security Events	525
11.14 Securing Containers	527
12 Accounting	529
12.1 The Accounting Log File	529
12.2 Viewing Accounting Information	530
12.3 Format of Accounting Log Messages	530
12.4 Types of Accounting Log Records	532
12.5 Timeline for Accounting Messages	545
12.6 Resource Accounting	551
12.7 Options, Attributes, and Parameters Affecting Accounting	555
12.8 Accounting Caveats and Advice	557
13 Using MPI with PBS	559
13.1 Integration with MPI	559
13.2 Prerequisites	559
13.3 Types of Integration	559
13.4 Transparency to the User	561
13.5 Integrating Intel MPI 4.0.3 On Linux Using Environment Variables	561
13.6 Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script	562
13.7 Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script	562
13.8 Integration Using the TM Interface	562
13.9 Integration on the Fly using the <code>pbs_tmsh</code> Command	562
13.10 Integration by Wrapping	563
13.11 Wrapping an MPI Using the <code>pbsrun_wrap</code> Script	565
13.12 Unwrapping MPIs Using the <code>pbsrun_unwrap</code> Script	568
13.13 Integration By Hand	568
13.14 How Processes are Started Using MPI and PBS	573
13.15 Limit Enforcement with MPI	575
13.16 Restrictions and Caveats for MPI Integration	576

14	Configuring PBS for SELinux	577
14.1	Overview of PBS Support for MLS-compliant SELinux	577
14.2	Terminology	577
14.3	How Support for SELinux Works	577
14.4	Enforcement of Permissions	578
14.5	Special Attributes and Directories	578
14.6	Prerequisites	579
14.7	Caveats and Restrictions	579
14.8	Installing PBS For Use With SELinux	579
14.9	Configuring PBS for SELinux	581
14.10	Managing an SELinux System	582
15	Managing Power Usage	583
15.1	Monitoring and Controlling Job Power Usage	583
15.2	Power Management Attributes, Resources, Etc.	587
15.3	Caveats and Restrictions for Power Management	589
16	Provisioning	591
16.1	Introduction	591
16.2	Definitions	591
16.3	How Provisioning Can Be Used	591
16.4	How Provisioning Works	592
16.5	Configuring Provisioning	599
16.6	Viewing Provisioning Information	604
16.7	Requirements and Restrictions	607
16.8	Defaults and Backward Compatibility	609
16.9	Example Scripts	609
16.10	Advice and Caveats	617
16.11	Errors and Logging	619
17	Support for HPE	623
17.1	Support for HPE with Cpusets	623
17.2	Support for HPE Cray Shasta	624
18	Support for NEC SX-Aurora TSUBASA	627
18.1	Vnodes for NEC SX-Aurora TSUBASA	627
18.2	Terminology	627
18.3	Resources for SX-Aurora TSUBASA	628
18.4	Configuring PBS for NEC SX-Aurora TSUBASA	629
18.5	Debugging on NEC SX-Aurora TSUBASA	630
18.6	Suspending and Resuming Jobs	630
18.7	Job Accounting on NEC SX-Aurora TSUBASA	630
19	Mixed Linux-Windows Operation	631
19.1	Introduction to Mixed Linux-Windows Operation	631
19.2	Configuration	631
19.3	Troubleshooting Mixed Linux-Windows Complex	633

Contents

20 Problem Solving	635
20.1 Debugging Tools	635
20.2 Security and Permissions Problems	636
20.3 Troubleshooting Jobs	636
20.4 Troubleshooting Daemons	640
20.5 Troubleshooting Vnodes	642
20.6 Troubleshooting Client Commands	643
20.7 Troubleshooting PBS Licenses	645
20.8 Crash Recovery	645
20.9 Other Troubleshooting	646
20.10 Getting Help	647
Index	649

About PBS Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

Document Conventions

Abbreviation

The shortest acceptable abbreviation of a command or subcommand is underlined

Attribute

Attributes, parameters, objects, variable names, resources, types

Command

Commands such as `qmgr` and `scp`

Definition

Terms being defined

File name

File and path names

Input

Command-line instructions

Method

Method or member of a class

Output

Output, example code, or file contents

Syntax

Syntax, template, synopsis

Utility

Name of utility, such as a program

Value

Keywords, instances, states, values, labels

Notation

Optional Arguments

Optional arguments are enclosed in square brackets. For example, in the `qstat` man page, the `-E` option is shown this way:

`qstat [-E]`

About PBS Documentation

To use this option, you would type:

```
qstat -E
```

Variable Arguments

Variable arguments (where you fill in the variable with the actual value) such as a job ID or vnode name are enclosed in angle brackets. Here's an example from the `pbsnodes` man page:

```
pbsnodes -v <vnode>
```

To use this command on a vnode named "my_vnode", you'd type:

```
pbsnodes -v my_vnode
```

Optional Variables

Optional variables are enclosed in angle brackets inside square brackets. In this example from the `qstat` man page, the job ID is optional:

```
qstat [<job ID>]
```

To query the job named "1234@my_server", you would type this:

```
qstat 1234@my_server
```

Literal Terms

Literal terms appear exactly as they should be used. For example, to get the version for a command, you type the command, then "--version". Here's the syntax:

```
qstat --version
```

And here's how you would use it:

```
qstat --version
```

Multiple Alternative Choices

When there are multiple options and you should choose one, the options are enclosed in curly braces. For example, if you can use either "-n" or "--name":

```
{-n | --name}
```

List of PBS Professional Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

PBS Professional Release Notes

Supported platforms, what's new and/or unexpected in this release, deprecations and interface changes, open and closed bugs, late-breaking information. For administrators and job submitters.

PBS Professional Big Book

All your favorite PBS guides in one place: *Installation & Upgrade*, *Administrator's*, *Hooks*, *Reference*, *User's*, *Programmer's*, *Cloud*, *Budget*, and *Simulate* guides in a single book.

PBS Professional Installation & Upgrade Guide

How to install and upgrade PBS Professional. For the administrator.

PBS Professional Administrator's Guide

How to configure and manage PBS Professional. For the PBS administrator.

PBS Professional Hooks Guide

About PBS Documentation

How to write and use hooks for PBS Professional. For the PBS administrator.

PBS Professional Reference Guide

Covers PBS reference material: the PBS commands, resource, attributes, configuration files, etc.

PBS Professional User's Guide

How to submit, monitor, track, delete, and manipulate jobs. For the job submitter.

PBS Professional Programmer's Guide

Discusses the PBS application programming interface (API). For integrators.

PBS Professional Manual Pages

PBS commands, resources, attributes, APIs.

PBS Professional Licensing Guide

How to configure licensing for PBS Professional. For the PBS administrator.

PBS Professional Cloud Guide

How to configure and use the PBS Professional Cloud feature in order to burst jobs to the cloud.

PBS Professional Budgets Guide

How to configure Budgets and use it to track and manage resource usage by PBS jobs.

PBS Professional Simulate Guide

How to configure and use the PBS Professional Simulate feature.

Where to Keep the Documentation

If you're not using the *Big Book*, make cross-references work by putting all of the PBS guides in the same directory.

Ordering Software and Licenses

To purchase software packages or additional software licenses, contact your Altair sales representative at pbssales@altair.com.

New Features

This chapter briefly lists new features by release, with the most recent listed first.

For deprecations, please see the *Release Notes*.

The *Release Notes* included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the *Release Notes*, as they may contain information that was not available when this book was written.

1.1 New Features in This Release

GraphQL Web API

Application developers can use a GraphQL web API with PBS Professional. See the *PBS Professional GraphQL Web API Guide*.

Budgets Updates Formula at PBS

Budgets takes care of contacting PBS and updating the formula. See the *Budgets Guide*.

Configurable Timeout for qsub

You can specify the amount of time qsub waits before giving up on connecting to the server, and the interval at which it attempts to connect to the server. This is useful for failover setups. See [section 8.6.4, “Configuring Timeout for qsub Connection”, on page 414](#).

Specifying Whether Queued Job Limit Applies to Array Jobs or Individual Subjobs

You can specify whether the server max_queued limit attribute counts an array job as a single job, or whether each sub-job is counted for the limit, using the elim_on_subjobs server attribute. See [section 10.23, “Managing Job Array Behavior”, on page 488](#).

Configurable Job Dispatch Rate

You can configure how long the scheduler waits after dispatching a job before it dispatches the next job; see [section 4.3.11, “Configurable Job Dispatch Rate”, on page 82](#).

Support for IPv6

The core PBS Professional supports IPv6. See the Special Notes section in the *PBS Professional Release Notes* for restrictions and supported configurations.

Budgets Provides Burn Rate Reports

Budgets provides burn rate reports. See the *Budgets Guide*.

Budgets Provides Default Configuration for Quick Start

Budgets provides an optional default configuration and tools for easy setup. See the *Budgets Guide*.

1.2 Changes in Previous Releases

Optional Python Garbage Collection (2022.1.3)

You can choose to run Python garbage collection in order to manage memory usage. See [section 9.14, “Avoiding Memory Hogging by Python Scripts”, on page 453](#).

Cgroups Management for OOM Killer

You can manage the OOM killer in cgroups. See [section 6.8, “Managing Oom Killer for Cgroups”, on page 353](#).

Cloud Costs Integrated with Budgets (2022.1)

You can get quotes for and manage cloud job costs. See the *PBS Professional Budgets Guide*.

New Hooks (2022.1)

PBS has the new hook events `postqueuejob`, `management`, `modifyvnode`, `jobobit`, `resv_begin`, `resv_confirm`, and `modifyresv`. See the *PBS Professional Hooks (Plugins) Guide*.

New Option to `pbs_ralter` to Specify Allowed Idle Time (2022.1)

PBS lets you alter a reservation to specify its allowed idle time. See [“`pbs\_ralter`” on page 85 of the PBS Professional Reference Guide](#).

Choice of Mailer is Configurable (2022.1)

You can choose the mailer PBS uses via the new `mailer` server attribute. See [section 2.2, “How PBS Uses Mail”, on page 21](#).

Multi-host Jobs Can Resume After MoM Restart (2022.1)

Multi-vnode jobs can survive a MoM restart when using `pbs_mom -p`. See [“`pbs\_mom`” on page 71 of the PBS Professional Reference Guide](#).

MoM Can Use Custom Command for Local Copy (2022.1)

You can configure MoM to use a custom command for local copy, by specifying the command in the `PBS_CP` parameter in `pbs.conf`. See [section 9.7, “Setting File Transfer Mechanism”, on page 441](#).

Scheduler Can Run as Non-root User (2022.1)

The scheduler(s) can run as a user other than root. See [section 9.1, “Specifying Scheduler Username”, on page 420](#).

Scheduler Makes Persistent Connection to Server (2022.1)

Each scheduler makes a persistent connection to the server. See [“Schedulers” on page 1 in the PBS Professional Installation & Upgrade Guide](#).

Ability to Obfuscate Existing Snapshots (2022.1)

You can obfuscate existing snapshots. See [“`pbs\_snapshot`” on page 111 of the PBS Professional Reference Guide](#).

Adding Resources to Running Reservations (2022.1)

You can add resources to a running reservation. See [section 4.9.37.4, “Modifying Reservations”, on page 201](#).

PBS Cloud Uses Simulate for Bursting to Cloud (2021.1.3)

PBS Cloud uses Simulate to figure out which nodes to burst and which jobs to run. See the *PBS Professional Cloud Guide*.

Budgets Provides Cost Estimate (2021.1.3)

Budgets can provide the job submitter with an estimate of the cost for running a job. See the *PBS Professional Budgets Guide*.

Managing GPUs Outside of Cgroups (2021.1.3)

You can use other methods besides the cgroups hook to manage GPUs on specific nodes, while still running the cgroups hook. See [section 6.5.5.6, “Not Using Cgroups to Manage GPUs”, on page 349](#) and [section 5.14.7, “Using GPUs”, on page 279](#).

New Option to Specify that User’s Home Directory is Shared (2021.1.1)

The administrator can specify that the user's home directory is shared in order to prevent sister MoMs from prematurely removing job files. See [Chapter 10, “Staging and Execution Directories for Job”, on page 474/](#)

New Postpaid Mode for Budgets (2021.1.1)

You can use Budgets in postpaid or prepaid mode. See [“Postpaid or Prepaid Mode” on page 1 of the PBS Professional Budgets Guide](#).

Support for NVIDIA MIG (2021.1.1)

PBS supports NVIDIA MIGs. See [Chapter 6, “Configuring and Using PBS with Cgroups”, on page 311](#).

Improvements to Cgroups Hook (2021.1.1)

The cgroups hook has improvements to help manage swap and device discovery, and better default values for configuration file parameters `mem_fences`, `reserve_amount` in memory subsystem, `vnode_hidden_mb`. See [Chapter 6, “Configuring and Using PBS with Cgroups”, on page 311](#).

Better License Management for Cloud (2021.1.1)

PBS Cloud checks for application license availability; see the *PBS Cloud Guide*.

Offline Install Procedure for Cloud (2021.1.1)

You can install PBS Cloud on an offline host. See the *PBS Cloud Guide*.

New Options for Altering Reservations (2021.1)

You can change a reservation's select specification, and the administrator can override the scheduler to change the start time, end time, and duration of a reservation. See [“pbs_ralter” on page 85 of the PBS Professional Reference Guide](#).

Integration with NEC SX-Aurora TSUBASA (2021.1)

PBS provides topologically aware job resource requests and scheduling. For configuration information, see [Chapter 18, “Support for NEC SX-Aurora TSUBASA”, on page 627](#). For job submission instructions, see [“Submitting Jobs to NEC SX-Aurora TSUBASA”, on page 205 of the PBS Professional User’s Guide](#).

Integration with Container Access Control (2021.1)

PBS allows you to whitelist container registries, and supports logging into private container registries. You can also manage mount paths for greater security. See [Chapter 7, “Configuring PBS for Containers”, on page 355](#).

Managing Shared Job Directory Behavior (2021.1)

You can prevent sister MoMs from prematurely removing shared job directories and files during node release. See [section 10.13.1, “Staging and Execution Directories for Job”, on page 474](#).

Limiting Number of Subjobs Running at One Time (2021.1)

Job submitters can limit the number of simultaneously running subjobs for an array job; see [“Limiting Number of Simultaneously Running Subjobs”, on page 156 of the PBS Professional User’s Guide](#).

New Cgroups Hook (2020.1)

PBS has an expanded cgroups hook with many new capabilities. This hook replaces the cpuset MoM. See [Chapter 6, “Configuring and Using PBS with Cgroups”, on page 311](#).

Cloud Bursting Feature (2020.1)

PBS now has its own cloud bursting feature. See the PBS [Cloud Guide](#).

Budget Allocation Feature (2020.1)

PBS now has its own budget allocation feature. See the PBS [Budgets Guide](#).

Workload Simulation Feature (2020.1)

PBS now has its own workload simulation feature. See the PBS [Simulate Guide](#).

Timeout for Dynamic Server Resource Scripts (2020.1)

By default, PBS allows a dynamic server resource script 30 seconds to run. You can configure the timeout; see [section 5.14.3.1, “Creating Server Dynamic Resource Scripts”, on page 263](#).

Specifying Hosts or Vnodes to Keep when Releasing Unneeded Vnodes (2020.1)

You can specify how many hosts or which vnodes to keep when releasing unneeded vnodes. See [“pbs_release_nodes” on page 92 of the PBS Professional Reference Guide](#).

Using Undo Live Recorder to Capture Daemon Execution Recordings (2020.1) (Removed 2021.1.2)

You can use Undo Live Recorder to capture execution history for analysis by Altair support. See [section 20.1.4, “Finding PBS Version Information”, on page 635](#).

PBS Reconfirms Degraded Reservations (2020.1)

If reservation vnodes become unavailable, PBS looks for replacements. See [section 8.4.2, “Finding Replacement Vnodes for Degraded and In-conflict Reservations”, on page 402](#).

New Default for TPP Message Processing (2020.1)

The default for the number of TPP messages the server can process per thread iteration is now 64. See [“rpp_max_pkt_check” on page 295 of the PBS Professional Reference Guide](#).

Automatic Deletion of Idle Reservations (2020.1)

PBS can automatically delete idle reservations. See [“Introduction to Creating and Using Advance and Standing Reservations”, on page 138 of the PBS Professional User’s Guide](#).

Flexible Job-specific Reservations (2020.1)

You can create flexible job-specific reservations for queued or running jobs. See [“Job-specific Reservations”, on page 142 of the PBS Professional User’s Guide](#).

Altering Reservation Duration, Authorized Groups, Authorized Users (2020.1)

You can alter the duration of a reservation; see [“pbs_ralter” on page 85 of the PBS Professional Reference Guide](#).

Accounting Record for Job Suspend and Resume (2020.1)

PBS records job suspension and resumption in the accounting log. See [Chapter 12, “Accounting”, on page 529](#).

Managing Number of Scheduler Threads (2020.1)

You can set the maximum number of threads used by each scheduler. See [section 4.5.8.3, “Setting Number of Scheduler Threads”, on page 102](#).

Configurable Authentication Methods (2020.1)

You can use various authentication methods with PBS; see [section 11.4, “Authentication for Daemons & Users”, on page 508](#).

Using TLS for Encryption (2020.1)

You can use TLS encryption with PBS. See [section 11.5, “Encrypting PBS Communication”, on page 517](#).

Mixed Operation on Linux and Windows (2020.1)

You can use both Linux and Windows execution and client hosts in the same PBS complex. See [Chapter 19, “Mixed Linux-Windows Operation”, on page 631](#).

Run Jobs on First Available Resources (Beta 2020.1; no longer beta 2022.1)

You can submit a set of jobs that would all accomplish the same thing, but that specify different resources. PBS runs only the first that can run. See ["Running Your Job on First Available Resources", on page 110 of the PBS Professional User's Guide](#).

New pbs_login Command (2020.1)

PBS includes a new command for user authentication called `pbs_login`. See ["pbs_login" on page 69 in the PBS Professional Installation & Upgrade Guide](#).

One way to sort jobs for preemption (2020.1)

Jobs are chosen for preemption only by which have been running the shortest time. See [section 4.9.33, "Using Preemption", on page 180](#).

New Threading Option for Schedulers (2020.1)

You can specify the number of threads each scheduler runs. See ["pbs_sched" on page 105 of the PBS Professional Reference Guide](#).

License Server for Node and Socket Licenses (2020.1)

PBS uses a license server to license hosts in the complex. See the *PBS Works Licensing Guide*.

Specifying Additional Arguments for Container Engines (2020.1)

Job submitters can specify additional container engine arguments such as secondary groups and shared memory; see ["Specifying Additional Arguments to Container Engine", on page 134 of the PBS Professional User's Guide](#).

Update to SELinux Support

Support for SELinux is updated. See [Chapter 14, "Configuring PBS for SELinux", on page 577](#).

Preemption via Deletion (19.4)

You can use deletion to preempt jobs. See [section 4.9.33, "Using Preemption", on page 180](#).

New Scheduler Attributes for Preemption (19.4)

The `preempt_order`, `preempt_prio`, `preempt_queue_prio`, and `preempt_sort` preemption settings are now scheduler attributes with the same names and formats. See ["Scheduler Attributes" on page 299 of the PBS Professional Reference Guide](#).

All Groups Included in Group ACLs (19.4)

All of a user's groups are included in the list of groups in group ACLs. See [section 11.3.4.5, "Contents of Group ACLs", on page 494](#).

Changes to qstat Job Output (19.4)

Wide output lines can be displayed for any default or alternate `qstat` job output formats, and when output size is too large for a field, the last character is replaced with an asterisk. See ["qstat" on page 200 of the PBS Professional Reference Guide](#).

Subjob Run Count Tracking (19.4)

PBS tracks the `run_count` attribute for subjobs, and holds job arrays whose subjobs hit the run count limit. See [section 10.18, "Managing Number of Run Attempts", on page 484](#).

Faster Read of Custom Job Resources by Execution Hooks (19.4)

You can specify which custom resources are cached at MoMs so that execution hooks can read them faster. See [section 5.14.2.5, "Specifying Whether Resource is Cached at MoM", on page 259](#).

Applications Running in Containers Can Use Ports (19.4)

PBS can provide ports for applications running in containers. See ["Configuring PBS for Containers"](#).

Support for Singularity Containers (19.4)

You can run PBS jobs in Singularity containers. See ["Configuring PBS for Containers"](#).

New Post-suspend and Pre-resume Hooks (19.4)

PBS has two new hook events for just after suspending a job and just before resuming it. See ["Event Types" on page 87 in the PBS Professional Hooks Guide](#).

Scheduler Logging Consistent with Other Daemons (19.4)

Schedulers use the same logging scheme as other daemons. See ["Event Logging" on page 429 of the PBS Professional Reference Guide](#).

Option to Capture Only PBS Configuration Information with `pbs_snapshot` (19.4)

You can use the new `pbs_snapshot --basic` option to capture just PBS configuration information. See ["pbs_snapshot" on page 111 of the PBS Professional Reference Guide](#).

Support for Cray Shasta Systems (19.4)

PBS is supported on Cray's Shasta systems.

Expanded and New Accounting Records (19.4)

PBS writes a new "a" accounting record when a job is altered, and the "Q" record is expanded to include more information. See [section 12.4, "Types of Accounting Log Records", on page 532](#).

IP Address Can Be Used for Vnode Name (19.4)

You can use the IP address as the vnode name. See ["Vnode Name" on page 358 of the PBS Professional Reference Guide](#).

Developer Libraries and Headers in Developer Package (19.4)

The libraries and headers needed for development but not for running PBS have been moved to a developer package. See ["Developer Headers and Libraries" on page 19 in the PBS Professional Programmer's Guide](#).

PBS Uses Python 3 (19.4)

As of 19.4.1, PBS uses Python 3.

New Basic Option to `pbs_snapshot` (19.4)

The `pbs_snapshot` command has a new `--basic` option. See ["pbs_snapshot" on page 111 of the PBS Professional Reference Guide](#).

New Maintenance Reservation (19.4)

PBS provides a new type of reservations for performing maintenance. See [section 4.9.37, "Reservations", on page 196](#).

Windows MoMs and Clients Run with Linux Server, Schedulers, Comms (19.4)

As of 19.4.1, PBS complexes that run Windows MoMs and Clients run with Linux server, schedulers, and comms.

Undo Live Recorder Debugger (19.4) (Removed 2021.1.2)

Undo's Live Recorder integration enhances our ability to pinpoint root causes of problematic behavior. (This capability is used under the direction of Altair support staff to speed troubleshooting.)

PBS Defaults to 24/7 Primetime (19.2)

You can use PBS without configuring primetime and/or holidays. See [section 4.9.34, "Using Primetime and Holidays", on page 190](#)

Microsecond Logging (19.2)

You can choose to have daemons log with microsecond resolution. See [section 9.4.4.1, "Event Logfile Format", on page 432](#).

Limiting ncpus Count to Cores (19.2)

You can opt not to include hyperthreads when calculating the value for ncpus that MoM reports to the server. See [Chapter 6, "Configuring and Using PBS with Cgroups", on page 311](#).

Change in Enabling Power Provisioning (19.2)

You enable power provisioning by enabling the PBS_power hook. See [Chapter 15, "Managing Power Usage", on page 583](#).

Settable Maximum Job ID (19.2)

You can set the maximum value for job IDs, job array IDs, and reservation IDs, using the [max_job_sequence_id](#) server attribute.

Job Vnode Fault Tolerance (19.2)

You can allocate extra vnodes to jobs to allow jobs to successfully start and run despite vnode failures. See [section 8.5, "Vnode Fault Tolerance for Job Start and Run", on page 403](#).

Hooks Support Reliable Job Startup and Run (19.2)

Hooks have been enhanced to allow you to provide jobs with extra vnodes in case of vnode failure. See [section 8.5, "Vnode Fault Tolerance for Job Start and Run", on page 403](#).

New Reservation End Hook (19.2)

You can create hooks for the end of a reservation. See ["resv_end: Event when Reservation Ends" on page 100 in the PBS Professional Hooks Guide](#).

Enhancements to pbs_snapshot (19.2)

You can run pbs_snapshot without root privilege, and the command captures JSON output. See ["pbs_snapshot" on page 111 of the PBS Professional Reference Guide](#).

Tunable Job Release Wait Time for Cray (19.2)

You can set the amount of time that PBS waits between sending release requests to ALPS.

Managing Power Usage on Cray (18.2)

You can power nodes up and down, limit ramp rate, and use power profiles for jobs. See [Chapter 15, "Managing Power Usage", on page 583](#).

On Cray, PBS Creates One Vnode per Compute Node (18.2)

Default behavior on the Cray has changed to create one vnode per compute node.

Suspend and Resume on Cray (18.2)

You can use suspend and resume on Cray.

Installing PBS on Cray CLE 5.2 via RPM (18.2)

PBS is installed on Cray CLE 5.2 via RPM. (No longer available)

Performance Enhancement for PBS on Cray via Improved MoM Reporting (18.2)

You can improve the performance of PBS on Cray by using the vnode_pool vnode attribute. This allows only one MoM to report inventory, and reduces communication traffic.

Periodic Synchronization of Inventory on Cray (18.2)

PBS periodically makes sure that its inventory matches what ALPS reports.

On Cray, Automatic Creation of One Vnode Per Compute Node (18.2)

PBS automatically creates one vnode for each compute node.

Installing PBS on CLE 6 via IMPS (18.2)

(No longer available)

Support for Xeon Phi (18.2)

PBS supports Xeon Phi.

1.2.1 New Scheduling Features

Restricting Placement Set Creation to Resources with Values that Have Been Set (18.2)

See [section 4.9.32, “Placement Sets”, on page 168](#).

Soft Walltimes for Jobs (18.2)

You can set a soft walltime for jobs, and PBS can estimate a job's soft walltime. See [section 4.9.44, “Using Soft Walltime”, on page 218](#)

Formula Uses Fairshare (18.2)

You can use fairshare in the job sorting formula. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

Manage Partitions with Multischeds (18.2)

You can schedule each partition separately. See [section 4.2, “Scheduling Each Partition Separately”, on page 59](#).

Run Jobs in a Cloud (18.2)

PBS can burst jobs to a cloud. See the *PBS Professional Cloud Guide*.

1.2.2 New Hooks Features

The `execjob_prologue` Hook Runs on All Sister MoMs (18.2)

The `execjob_prologue` hook runs on all sister MoMs. See ["execjob_prologue: Event Just Before Execution of Top-level Job Process" on page 104 in the PBS Professional Hooks Guide](#).

Python Version Changed to 2.7.1 (18.2)

PBS 18.2.1 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Periodic Server Hook (18.2)

PBS has a periodic hook that runs at the server. See ["periodic: Periodic Event at Server Host" on page 101 in the PBS Professional Hooks Guide](#).

Hook to Run Job Start Time Estimator (18.2)

PBS has a built-in hook named `PBS_est` that can run the job start time estimator. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).

Configurable Python Interpreter Restarts (18.2)

You can configure how often you want the Python interpreter to restart. See ["Restarting the Python Interpreter" on page 24 in the PBS Professional Hooks Guide](#).

PBS Can Report Custom Resources Set in Hooks (18.2)

MoM can accumulate and report custom resources that are set in a hook. See ["Setting Job Resources in Hooks" on page 50 in the PBS Professional Hooks Guide](#)

1.2.3 Other New Features

Managing Job Resource Use with Cgroups (18.2)

You can use cgroups to manage the resources used by jobs. See [Chapter 6, "Configuring and Using PBS with Cgroups", on page 311](#).

Running Jobs in Containers (18.2)

Job submitters can run each job in its own container. See [Chapter 7, "Configuring PBS for Containers", on page 355](#) and ["Running Your Job in a Container", on page 132 of the PBS Professional User's Guide](#).

Power Provisioning (18.2)

PBS can monitor and control job power usage. See [Chapter 15, "Managing Power Usage", on page 583](#).

Collecting Diagnostic Information with `pbs_snapshot` Command (18.2)

See ["pbs_snapshot" on page 111 of the PBS Professional Reference Guide](#).

New `pbs_ralter` Command (18.2)

You can change reservations using the `pbs_ralter` command. See ["Modifying Reservations", on page 144 of the PBS Professional User's Guide](#).

Privileged Access to Server for MoMs (18.2)

You can give all MoMs privileged access to the server without having to explicitly add their hosts to the `acl_hosts` server attribute. See [section 11.3.7.3, "Access to Server for MoMs", on page 500](#).

Releasing Unneeded Vnodes from Jobs (18.2)

You can release vnodes that were allocated to jobs when those vnodes are no longer needed. See ["Releasing Unneeded Vnodes from Your Job", on page 129 of the PBS Professional User's Guide](#).

Running Subjobs Survive Server Restart (18.2)

Subjobs of an array job will continue to run during a restart of the server.

Writing Output and Error Files Directly to Final Destination (18.2)

You can have PBS write your standard output and error files directly to their final destination. See ["Writing Files Directly to Final Destination", on page 47 of the PBS Professional User's Guide](#).

Deleting Output and Error Files (18.2)

You can have PBS delete your standard output and error files. See ["Avoiding Creation of stdout and/or stderr", on page 45 of the PBS Professional User's Guide](#).

Output for `qstat` in JSON and DSV Formats; `qstat` Attribute Output on Single Line (18.2)

You can get output from `qstat` in JSON or DSV formats. You can also print out attribute information in one unbroken line. See ["qstat" on page 200 in the PBS Professional Installation & Upgrade Guide](#).

Specifying Resources to Release on Suspension (18.2)

You can specify which resources you want released when jobs are suspended. See [section 5.9.6.2, "Job Suspension and Resource Usage", on page 247](#).

Maintenance State for Powered-up Vnodes (18.2)

You can suspend a job and put all the vnodes belonging to a job into the *maintenance* state. See [section 9.5.2, "Performing Maintenance on Powered-up Vnodes", on page 436](#).

Debuginfo RPM Package (18.2)

PBS is packaged with a debuginfo RPM package. See [section 20.1.3, "Using the debuginfo RPM Package", on page 635](#).

Logging Hostname and Interfaces (18.2)

Each time a log file is opened, PBS logs the hostname and interface information. See [section 9.4, "Event Logging", on page 429](#).

Subjobs Survive Server Restarts (18.2)

Subjobs keep running after you stop the server. See [“Impact of Stop-Restart on Running Linux Jobs” on page 154 in the PBS Professional Installation & Upgrade Guide](#) and [“Impact of Stop-Restart on Running Windows Jobs” on page 160 in the PBS Professional Installation & Upgrade Guide](#).

You can see all attributes for subjobs; see [“Viewing Status of a Job Array”, on page 161 of the PBS Professional User’s Guide](#).

Jobs Can Use Provisioning for Some Chunks (18.2)

Jobs can request an AOE for some chunks as long as all chunks use the same AOE. See [Chapter 15, “Using Provisioning”, on page 219](#).

Node Licenses (18.2)

You can license your hosts using node licenses. See the *PBS Works Licensing Guide*.

PBS Can Send Mail for Subjobs (18.2)

PBS can send mail for subjobs. See [“Specifying Email Notification”, on page 25 of the PBS Professional User’s Guide](#).

Server Periodic Hook (14.2)

You can run a hook periodically at the server. See [“periodic: Periodic Event at Server Host” on page 101 in the PBS Professional Hooks Guide](#).

Hook to Run Job Start Time Estimator (14.2)

PBS has a built-in hook named PBS_est that can run the job start time estimator. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).

PBS Can Report Custom Resources Set in Hooks (14.2)

MoM can accumulate and report custom resources that are set in a hook. See [section 5.2.4.12, “Setting Job Resources in Hooks”, on page 50](#).

Configurable Python Interpreter Restarts (14.2)

You can configure how often you want the Python interpreter to restart. See [“Restarting the Python Interpreter” on page 24 in the PBS Professional Hooks Guide](#).

Python Version Changed to 2.7.1 (14.2)

PBS 14.2.1 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Name for MoM to Use for Parent Vnode (14.2)

You can specify the name that MoM should use for her parent vnode and child vnodes. See [section 3.3.2, “How to Choose Vnode Names”, on page 42](#).

Grouping Jobs and Sorting by ID (14.2)

When getting job status, you can group jobs and sort them by ID. See [“Grouping Jobs and Sorting by ID”, on page 181 of the PBS Professional User’s Guide](#).

Support for systemd (14.2)

PBS supports using systemctl commands to start, stop, restart, and status PBS. See [“Methods for Starting, Stopping, or Restarting PBS” on page 144 in the PBS Professional Installation & Upgrade Guide](#).

Support for Native Package Managers on Linux (14.2)

PBS supports use of RPM for installation and upgrading. See [“Installation” on page 19 in the PBS Professional Installation & Upgrade Guide](#) and [“Upgrading” on page 67 in the PBS Professional Installation & Upgrade Guide](#).

Server Sets Job Comment on Run or Reject (14.2)

The server sets the job comment when the job is run or rejected. See [section 10.7.3.1, “Comment Set When Running Job”, on page 465](#).

Update to Accounting R Record (14.2)

PBS writes the R accounting record when MoM is restarted with -p or -r. See [section , “R”, on page 539](#).

Interactive GUI Jobs on Windows (13.1)

Users can run interactive GUI jobs on Windows. See ["Submitting Interactive GUI Jobs on Windows", on page 127 of the PBS Professional User's Guide](#).

Administrators can choose a remote viewer for interactive GUI jobs. See [section 10.20.1, “Configuring PBS for Remote Viewer on Windows”, on page 484](#).

MUNGE Integration (13.1)

PBS can use MUNGE to create and validate credentials. See [section 11.4.4, “Authentication via MUNGE”, on page 509](#).

Controlling Backfill Depth at the Queue (13.1)

Administrators can choose the backfilling depth independently at each queue. See [section 4.9.3, “Using Backfilling”, on page 109](#).

Optional Scheduler Cycle Speedup (13.1)

You can optionally speed up the scheduling cycle. See [section 4.9.40, “Scheduler Cycle Speedup”, on page 209](#).

Preventing Some Jobs from Being Top Jobs (13.1)

You can prevent a job from being a top job by setting its topjob_ineligible attribute to True. See [section 4.9.17.1, “Making Jobs Ineligible to be Top Jobs”, on page 139](#).

Improved Mail on Windows (13.1)

Under Windows, you can specify an SMTP server. (As of 19.4.1, PBS does not use an SMTP server.)

New Hook Events (13.0)

PBS provides three new hook events:

- An `execjob_launch` hook runs just before MoM runs the user's program
- An `execjob_attach` hook runs when `pbs_attach` is called
- An `exechost_startup` hook runs when MoM starts up

See ["When and Where Hooks Run" on page 15 in the PBS Professional Hooks Guide](#), ["`execjob\_launch`: Event when Execution Host Receives Job" on page 106 in the PBS Professional Hooks Guide](#), ["`execjob\_attach`: Event when `pbs\_attach\(\)` runs" on page 107 in the PBS Professional Hooks Guide](#), and ["`exechost\_startup`: Event When Execution Host Starts Up" on page 114 in the PBS Professional Hooks Guide](#).

Configuration Files for Hooks (13.0)

You can use configuration files with hooks. See ["Using Hook Configuration Files" on page 33 in the PBS Professional Hooks Guide](#).

Configuring Vnodes in Hooks (13.0)

You can use hooks to configure vnode attributes and resources. See ["Setting and Unsetting Vnode Resources and Attributes" on page 49 in the PBS Professional Hooks Guide](#).

Adding Custom Resources in Hooks (13.0)

You can use hooks to add custom non-consumable host-level resources. See ["Adding Custom Host-level Resources" on page 69 in the PBS Professional Hooks Guide](#).

Node Health Hook Features (13.0)

PBS has node health checking features for hooks. You can offline and clear vnodes, and restart the scheduling cycle. See ["Offlining and Clearing Vnodes Using the `fail` action Hook Attribute" on page 72 in the PBS Professional Hooks Guide](#) and ["Restarting Scheduler Cycle After Hook Failure" on page 69 in the PBS Professional Hooks Guide](#).

Hook Debugging Enhancements (13.0)

You can get hooks to produce debugging information, and then read that information in while debugging hooks. See ["Debugging Hooks" on page 185 in the PBS Professional Hooks Guide](#).

Managing Built-in Hooks (13.0)

You can enable and disable built-in hooks. See ["Managing Built-in Hooks" on page 181 in the PBS Professional Hooks Guide](#).

Scheduler Does not Trigger modifyjob Hooks (13.0)

The scheduler does not trigger modifyjob hooks. See the PBS Professional Hooks Guide.

Faster, Asynchronous Communication Between Daemons (13.0)

PBS has a communication daemon that provides faster, asynchronous communication between the server, scheduler, and MoM daemons. See ["Communication" on page 47 in the PBS Professional Installation & Upgrade Guide](#).

Enhanced Throughput of Jobs (13.0)

By default, the scheduler runs asynchronously to speed up job start, and jobs that have been altered via `qalter`, `server_dyn_res`, or peering can run in the same scheduler cycle in which they were altered.

Creating Custom Resources via qmgr (13.0)

You can create any custom resources using nothing but the `qmgr` command. See [section 5.14.2.6, "Defining Custom Resources via qmgr", on page 259](#).

Job Sorting Formula: Python Math Functions and Threshold (13.0)

You can use standard Python math functions in the job sorting formula. You can also set a threshold for job priority, below which jobs cannot run. See [section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 151](#).

Fairshare: Formula and Decay Factor (13.0)

You can use a mathematical formula for fairshare, and you can set a custom decay factor. See [section 4.9.19, "Using Fairshare", on page 139](#).

Preempted Jobs can be Top Jobs (13.0)

You can specify that preempted jobs should be classified as top jobs. See [section 4.9.16, "Calculating Job Execution Priority", on page 136](#). You can use a new scheduler attribute called `sched_preempt_enforce_resumption` for this; see [section 4.9.3, "Using Backfilling", on page 109](#).

Limiting Preemption Targets (13.0)

You can specify which jobs can be preempted by a given job. See [section 4.9.33.4.i, "Setting Job Preemption Targets", on page 182](#).

Limiting Number of Jobs in Execution Queues (13.0)

You can speed up the scheduling cycle by limiting the number of jobs in execution queues. See [section 4.5.8.2, "Limiting Number of Jobs Queued in Execution Queues", on page 102](#).

Improved Round-robin Behavior (13.0)

The `round_robin` scheduler parameter produces improved behavior. See [section 4.9.38, "Round Robin Queue Selection", on page 204](#).

Limiting Resources Allocated to Queued Jobs (13.0)

You can set limits on the amounts of resources allocated to queued jobs specifically. See [section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 283](#).

Running qsub in the Foreground (13.0)

By default, the `qsub` command runs in the background. You can run it in the foreground using the `-f` option. See ["qsub" on page 216 of the PBS Professional Reference Guide](#).

Windows Users can Use UNC Paths (13.0)

Windows users can use UNC paths for job submission and file staging. See ["Set up Paths", on page 8 of the PBS Professional User's Guide](#) and ["Using UNC Paths", on page 37 of the PBS Professional User's Guide](#).

Automatic Installation and Upgrade of Database (13.0)

PBS automatically installs or upgrades its database. See ["Automatic Upgrade of Database \(13.0\)" on page 68 in the PBS Professional Installation & Upgrade Guide](#).

Longer Job and Reservation Names (13.0)

You can use job and reservation names up to 236 characters in length. See ["Formats" on page 353 of the PBS Professional Reference Guide](#).

Address Disambiguation for Multihomed Systems (13.0)

You can disambiguate addresses for contacting the server, sending mail, sending outgoing traffic, and delivering output and error files. See ["PBS with Multihomed Systems" on page 60 in the PBS Professional Installation & Upgrade Guide](#).

Support for Hydra Process Manager in Intel MPI (13.0)

Intel MPI is integrated with PBS. See ["Integrating Intel MPI 4.0.3 On Linux Using Environment Variables" on page 561](#).

Enhancements to pbsnodes Command (13.0)

You can now use the `pbsnodes` command to edit the comment attribute of a host, to write out host information, and to operate on specific vnodes. See ["pbsnodes" on page 36](#).

Primary Group of Job Owner or Reservation Creator Automatically Added to Job group_list (13.0)

The job submitter's and reservation creator's primary group is automatically added to the job or reservation `group_list` attribute. See ["qsub" on page 216](#) and ["pbs rsub" on page 96](#).

Intel MPI Integrated under Windows (13.0)

MPI is integrated with PBS under Windows (as well as Linux). See ["Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script" on page 562](#).

MPICH2 Integrated under Windows (13.0)

MPICH2 is integrated with PBS under Windows (as well as Linux). See ["Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script" on page 562](#).

PBS pbsdsh Command Available under Windows (13.0)

The `pbsdsh` command is available under Windows. See ["pbsdsh" on page 30](#).

PBS TM APIs Available under Windows (13.0)

The PBS TM APIs are available under Windows. See ["TM Library" on page 95](#) of the *PBS Professional Programmer's Guide*.

PBS pbs_attach Command Available under Windows (13.0)

The `pbs_attach` command is available under Windows. See ["pbs_attach" on page 56](#).

Xeon Phi Reported on Cray (13.0)

PBS automatically detects and reports a Xeon Phi in the ALPS inventory.

Command Line Editing in qmgr (12.2)

The `qmgr` command provides a history and allows you to edit command lines. See ["Reusing and Editing the qmgr Command Line" on page 153 of the PBS Professional Reference Guide](#).

Interactive Jobs Available under Windows (12.2)

Job submitters can run interactive jobs under Windows. See ["Running Your Job Interactively", on page 123 of the PBS Professional User's Guide](#).

Job Run Count is Writable (12.2)

Job submitters and administrators can set the value of a job's run count. See [section 10.18, “Managing Number of Run Attempts”, on page 484](#) and [“Controlling Number of Times Job is Re-run”, on page 121 of the PBS Professional User’s Guide](#).

runjob Hook can Modify Job Attributes (12.2)

The runjob hook can modify a job's attributes and resources. See [“Using Attributes and Resources in Hooks” on page 45 in the PBS Professional Hooks Guide](#).

Jobs can be Suspended under Windows (12.2)

You can suspend and resume a job under Windows.

Configuration of Directory for PBS Component Temporary Files (12.2)

You can configure the root directory where you want PBS components to put their temporary files. See [section 9.9, “Temporary File Location for PBS Components”, on page 450](#).

Execution Event and Periodic Hooks (12.0)

You can write hooks that run at the execution host when the job reaches the execution host, when the job starts, ends, is killed, and is cleaned up. You can also write hooks that run periodically on all execution hosts. See the PBS Professional Hooks Guide.

Shrink-to-fit Jobs (12.0)

PBS allows users to specify a variable running time for jobs. Job submitters can specify a *walltime* range for jobs where attempting to run the job in a tight time slot can be useful. Administrators can convert non-shrink-to-fit jobs into shrink-to-fit jobs in order to maximize machine use. See [“Adjusting Job Running Time”, on page 112 of the PBS Professional User’s Guide](#) and [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

PBS Supports Socket Licensing (11.3)

PBS lets you use socket licenses to license hosts. See the *PBS Works Licensing Guide*.

Deleting Job History (11.3)

You can delete job histories. See [section 10.15.9, “Deleting Moved Jobs and Job Histories”, on page 483](#).

Managing Resource Usage by Project (11.2)

You can set resource usage limits for projects, at the server and queue. You can set limits for the amount of each resource being used, or for the number of jobs. Jobs have a new attribute called *project*. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

PBS Daemons Protected from OOM Killer (11.2)

PBS daemons are protected from being terminated by an OOM killer. See [section 8.8, “OOM Killer Protection”, on page 418](#).

PBS Supports X Forwarding for Interactive Jobs (11.2)

PBS allows users to receive X output from interactive jobs. See [“Receiving X Output from Interactive Linux Jobs”, on page 126 of the PBS Professional User’s Guide](#), and [section 9.3.1.1, “Contents of Environment File”, on page 428](#).

Support for Accelerators on Cray (11.2)

PBS provides tight integration for accelerators on Cray.

Support for Interlagos on Cray (11.1)

No longer supported.

Improved Cray Integration (11.0)

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs.

Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See the PBS Professional Hooks Guide.

Enhanced Job Placement (11.0)

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job's placement request. See ["Specifying Job Placement", on page 66 of the PBS Professional User's Guide](#).

Choice in PBS service account Name (11.0)

Under Windows, the PBS service account used to run PBS daemons can have any name. See ["Creating PBS Service Account in Domained Environment" on page 41 in the PBS Professional Installation & Upgrade Guide](#).

Change of Licensing Method (11.0)

As of 11.0, PBS is licensed using a new Altair license server. See the *PBS Works Licensing Guide*.

Change in Data Management (11.0)

PBS uses a new data service. See [section 9.6, "Managing the Data Service", on page 439](#).

Choice in Job Requeue Timeout (11.0)

You can choose how long the job requeue process should be allowed to run. See [section 8.6.3, "Setting Job Requeue Timeout", on page 414](#).

Backfilling Around Top N Jobs (10.4)

PBS can backfill around the most deserving jobs. You can configure the number of jobs PBS backfills around. See [section 4.9.3, "Using Backfilling", on page 109](#).

Estimating Job Start Times (10.4)

PBS can estimate when jobs will run, and which vnodes each job will use. See [section 4.9.15, "Estimating Job Start Time", on page 133](#).

Unified Job Submission (10.4)

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or Linux system. See ["Python Job Scripts", on page 14 of the PBS Professional User's Guide](#).

Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See [Chapter 16, "Provisioning", on page 591](#).

New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See the PBS Professional Hooks Guide.

New Scheduler Attribute (10.2)

PBS allows the administrator to set the scheduler's cycle time using the new `sched_cycle_length` scheduler attribute. See the `pbs_sched_attributes(7B)` manual page.

Walltime as Checkpoint Interval Measure (10.2)

PBS allows a job to be checkpointed according to its walltime usage. See the `pbs_job_attributes(7B)` manual page.

Managing Resource Usage (10.1)

You can set separate limits for resource usage by individual users, individual groups, generic users, generic groups, and the total used. You can limit the amount of resources used, and the number of queued and running jobs. These limits can be defined separately for each queue and for the server. See [section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 283](#). These new limits are incompatible with the limit attributes existing before Version 10.1.

Managing Job History (10.1)

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See [section 10.15, “Managing Job History”, on page 480](#).

Reservation Fault Tolerance (10.1)

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See [section 8.4, “Reservation Fault Tolerance”, on page 401](#).

Checkpoint Support via Epilogue (10.1)

Checkpointed jobs can be requested if the epilogue exits with a special value. See [section 8.3.7.3, “Requeueing via Epilogue”, on page 398](#).

Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See the PBS Professional Hooks Guide.

Versioned Installation (10.0)

PBS is now automatically installed in versioned directories. For most platforms, different versions of PBS can coexist, and upgrading is simplified. See [Chapter 3, “Installation”, on page 19](#) and [Chapter 6, “Upgrading”, on page 67](#) in the PBS Professional Installation and Upgrade Guide.

Resource Permissions for Custom Resources (9.2)

You can set permissions on custom resources so that they are either invisible to users or cannot be requested by users. This also means that users cannot modify a resource request for those resources via `qalter`. See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).

Extension to Job Sorting Formula (9.2)

The job sorting formula has been extended to include parentheses, exponentiation, division, and unary plus and minus. See [section 4.9.3, “Using Backfilling”, on page 109](#).

Eligible Wait Time for Jobs (9.2)

A job that is waiting to run can be accruing “eligible time”. Jobs can accrue eligible time when they are blocked due to a lack of resources. This eligible time can be used in the job sorting formula. Jobs have two new attributes, `eligible_time` and `accrue_type`, which indicates what kind of wait time the job is accruing. See [section 4.9.13, “Eligible Wait Time for Jobs”, on page 129](#).

Job Staging and Execution Directories (9.2)

PBS now provides per-job staging and execution directories. Jobs have new attributes `sandbox` and `jobdir`, the MoM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job's `sandbox` attribute is unset or is set to `HOME`, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. If MoM's `$jobdir_root` is set to a specific directory, that is where PBS will create job-specific staging and execution directories. If MoM's `$jobdir_root` is unset or set to `PBS_USER_HOME`, PBS will create the job-specific staging and execution directory under the user's home directory. See [section 10.13.1, “Staging and Execution Directories for Job”, on page 474](#).

Standing Reservations (9.2)

PBS now provides both advance and standing reservation of resources. A standing reservation is a reservation of resources for specific recurring periods of time. See [section 4.9.37, “Reservations”, on page 196](#).

New Server Attribute for Job Sorting Formula (9.1)

The new server attribute "job_sort_formula" is used for sorting jobs according to a site-defined formula. See [section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 151](#).

Change to sched_config (9.1)

The default for job_sort_key of "cput" is commented out in the default sched_config file. It is left in as a usage example.

Change to Licensing (9.0)

PBS now depends on an Altair license server that will hand out licenses to be assigned to PBS jobs. See the *PBS Works Licensing Guide*. PBS Professional versions 8.0 and below will continue to be licensed using the proprietary licensing scheme.

Installing With Altair Licensing (9.0)

If you will use floating licenses, we recommend that you install and configure the Altair license server before installing and configuring PBS. PBS starts up faster. See ["Overview of Installation" on page 19 in the PBS Professional Installation & Upgrade Guide](#).

Unset Host-level Resources Have Zero Value (9.0)

An unset numerical resource at the host level behaves as if its value is zero, but at the server or queue level it behaves as if it were infinite. An unset string or string array resource cannot be matched by a job's resource request. An unset boolean resource behaves as if it is set to "False". See [section 4.9.28.7, "Matching Unset Resources", on page 160](#).

Better Management of Resources Allocated to Jobs (9.0)

The resources allocated to a job from vnodes will not be released until certain allocated resources have been freed by all MoMs running the job. The end of job accounting record will not be written until all of the resources have been freed. The "end" entry in the job end ('E') record will include the time to stage out files, delete files, and free the resources. This will not change the recorded "walltime" for the job.

1.3 Commercial-only Features

PBS is dual-licensed. Altair releases a commercial version and an open-source version. The core of the product is the same, but the commercial version contains additional features available only in the commercial (licensed) version of PBS. We list some examples here, but there are more:

- Estimated start times for non-top jobs via pbs_est
- Container integration
- PBS licensing

1.4 Backward Compatibility

1.4.1 New and Old Resource Usage Limits Incompatible

The new resource usage limits are incompatible with the old resource usage limits. See [section 5.15.1.15, "Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1", on page 298](#), [section 5.15.1.13.v, "Do Not Mix Old And New Limits", on page 297](#), and [section 5.15.1.14.i, "Error When Setting Limit Attributes", on page 297](#).

1.4.2 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history, PBS releases j1's dependency, and takes appropriate action. If job j1 depends on a finished job j3 that has been purged from job history, j1 is rejected just as in previous versions of PBS where the job was no longer in the system.

1.4.3 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.

1.4.4 OS-level Checkpointing Not Supported

PBS does not directly support OS-level checkpointing. PBS supports checkpointing using site-supplied methods. See [section 8.3, "Checkpoint and Restart", on page 387](#).

1.4.5 Scheduler Parameters Changed to Scheduler Attributes (19.4.1)

The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in \$PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. You cannot use the old parameters. Make sure that you use qmgr to set the attributes as desired. See ["Scheduler Attributes" on page 299 of the PBS Professional Reference Guide](#).

1.4.6 Old -l nodes Syntax Incompatible with Cgroups

The cgroups hook does not transform old "-lnodes" syntax into the new select and place directives. If you need to support the old syntax on hosts managed by the cgroups hook, you can write a queuejob hook to do that for you, or you can have job submitters explicitly specify mem, vmem, and cgswap for jobs.

Configuring the Server and Queues

This chapter describes how to configure the server and any queues.

2.1 The Server

2.1.1 Configuring the Server

You configure the server by setting server attributes via the `qmgr` command:

```
Qmgr: set server <attribute> = <value>
```

For a description of the server attributes, see [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

For a description of the `qmgr` command, see [“qmgr” on page 152 of the PBS Professional Reference Guide](#).

2.1.2 Default Server Configuration

The default configuration from the binary installation sets the default server settings. An example server configuration is shown below:

```
qmgr
Qmgr: print server
#
# Create queues and set their attributes.
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server query_other_jobs = True
set server resources_default.ncpus = 1
set server resv_enable = True
set server node_fail_requeue = 310
set server max_array_size = 10000
set server default_chunk.ncpus=1
```

2.1.3 The PBS Node File

The server creates a file of the nodes managed by PBS. This node file is written only by the server. On startup each MoM sends a time-stamped list of her known vnodes to the server. The server updates its information based on that message. If the time stamp on the vnode list is newer than what the server recorded before in the node file, the server will create any vnodes which were not already defined. If the time stamp in the MoM's message is not newer, then the server will not create any missing vnodes and will log an error for any vnodes reported by MoM but not already known.

Whenever new vnodes are created, the server sends a message to each MoM with the list of MoMs and each vnode managed by the MoMs. The server will only delete vnodes when they are explicitly deleted via `qmgr`.

This is different from the node file created for each job. See ["The Job Node File", on page 79 of the PBS Professional User's Guide](#).

2.1.4 Server Configuration Attributes

See ["Server Attributes" on page 281 of the PBS Professional Reference Guide](#) for a table of server attributes.

2.1.5 Recording Server Configuration

If you wish to record the configuration of a PBS server for re-use later, you may use the `print` subcommand of `qmgr` (8B). For example,

```
qmgr -c "print server" > /tmp/server.out
qmgr -c "print node @default" > /tmp/nodes.out
```

will record in the file `/tmp/server.out` the `qmgr` subcommands required to recreate the current configuration including the queues. The second file generated above will contain the vnodes and all the vnode properties. The commands could be read back into `qmgr` via standard input:

```
qmgr < /tmp/server.out
qmgr < /tmp/nodes.out
```

2.1.6 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. The Globus MoM is no longer available.

2.1.7 Configuring the Server for Licensing

The PBS server must be configured for licensing. You must set the location where PBS will look for the license server(s), by setting the server attribute `pbs_license_info`, then force the server to re-query for licenses by setting the server's `scheduling` attribute to `True`. The other server licensing attributes have defaults, but you may wish to set them as well. See the *PBS Works Licensing Guide*.

You may also wish to have redundant license servers. See the *Altair License Management System Installation and Operations Guide*, available at www.pbsworks.com.

2.2 How PBS Uses Mail

PBS sends mail to the administrator for administration-related issues, and to job submitters for job-related issues. See ["Specifying Email Notification", on page 25 of the PBS Professional User's Guide](#) for information about mail PBS sends to job submitters.

PBS sends mail to the administrator under the following circumstances:

- When failover occurs, PBS sends an email is sent to and from the account defined in the server's `mail_from` attribute.
- When the database is stopped unexpectedly. For example:
"Panic shutdown of Server on database error. Please check PBS_HOME file system for no space condition."
- When your license is expiring, PBS sends mail once a day.

2.2.1 Configuring Choice of Mailer

You may want to wrap `sendmail`, or preprocess emails before sending them to users, for example deleting certain mail or aggregating it.

You can specify the mailer PBS uses by setting the server's `mailer` attribute to the path to the mailer. For example:

```
qmgr -c 'set server mailer = '/usr/bin/pbs_mail_preprocessing'
```

This attribute defaults to `SENDMAIL_CMD`. The default mail server PBS uses on Linux is `/usr/lib/sendmail`.

2.2.1.1 Requirements for Mailer

The mailer you choose needs to work similarly to `sendmail`, as follows:

- Read the body from `stdin`
- Accept "`-f $from`" as a parameter
- Accept an email address as the last parameter

2.2.2 Configuring Server Mail Address

You can configure the account that is used as the address to both send and receive administrative mail. These are the same account. For example, when failover occurs, an email is sent to and from the account defined in the server's `mail_from` attribute, saying that failover has occurred.

Use the `qmgr` command to set the `mail_from` server attribute to an address that is monitored regularly:

```
Qmgr: s server mail_from=<address>
```

2.2.3 Specifying Mail Delivery Domain

You can use the `PBS_MAIL_HOST_NAME` parameter in `pbs.conf` on the server host to direct mail to a domain in which the user can receive it. For example, if a job is submitted from a cluster node, it may not be possible for mail to be delivered there, especially if the job runs on a different cluster.

You can specify the destination domain for email that is sent by the server to the administrator or to job submitters or reservation creators by setting the `PBS_MAIL_HOST_NAME` parameter in `pbs.conf`.

2.2.3.1 Delivering Mail to Administrator

The default username for administrative mail is "adm". The following table shows where PBS sends administrator mail:

Table 2-1: How PBS Sets Administrator Mail

Value of <code>mail_from</code>	Destination
<code><username>@<hostname></code>	<code><username>@<hostname></code>
<code>user</code>	<code>user</code> (Destination depends on mail server configuration)
<code>unset</code>	<code>adm</code> (Destination depends on mail server configuration)

2.2.3.2 Delivering Mail to Job Submitter or Reservation Creator

The `Mail_Users` attribute is a list of one or more usernames. For each entry in the list, PBS handles the entry according to the rules in the following table showing where PBS sends job or reservation mail:

Table 2-2: How PBS Sets Job or Reservation Mail

Value of <code>Mail_Users</code>	Value of <code>PBS_MAIL_HOST_NAME</code>	
	Set	Unset
<code><user-name>@<hostname></code>	Linux: <code><username>@<hostname></code>	<code><username>@<hostname></code>
<code>user</code>	<code>user@PBS_MAIL_HOST_NAME</code>	<code>user@<server FQDN from Job_Owner attribute of job></code>
<code>unset</code>	<code><job owner>@ PBS_MAIL_HOST_NAME</code>	Linux: <code><job owner>@ <server FQDN from Job_Owner attribute of job></code>

2.2.4 Attributes, Parameters Etc. Affecting Mail

mailer

Server attribute specifying mailer that PBS should use. Default value: `SENDMAIL_CMD`.

mail_from

Server attribute. Mail is sent from and to this account when failover occurs.

Mail_Points

Job and reservation attribute. List of events where PBS sends mail to users who are the job or reservation owner, or are listed in the `Mail_Users` job or reservation attribute.

Mail_Users

Job and reservation attribute. List of users to whom mail about job or reservation is sent.

PBS_MAIL_HOST_NAME

Parameter in `pbs.conf`. Optional. Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's `Mail_Users` attribute. See [section 2.2.3, "Specifying Mail Delivery Domain", on page 22](#).

Should be a fully qualified domain name. Cannot contain a colon (":").

PBS_O_MAIL

Value of `MAIL` environment variable, taken from job submitter's environment.

2.3 Queues

When a job is submitted to PBS and accepted, it is placed in a queue. Despite the fact that the name implies first-in, first-out ordering of jobs, this is not the case. Job submission order does not determine job execution order. See [Chapter 4, "Scheduling", on page 57](#).

You can create different queues for different purposes: queues for certain kinds of jobs, queues for specific groups, queues for specific vnodes, etc. You can tell PBS how to automatically route jobs into each queue. PBS has a default execution queue named *workq*, where jobs are placed when no queue is requested; you can specify a different default queue. See [section 2.3.14, “Specifying Default Queue”, on page 35](#).

2.3.1 Kinds of Queues

2.3.1.1 Execution and Routing Queues

There are two main types of PBS queues: *routing* and *execution*.

- A routing queue is used only to move jobs to other queues (destination queues). These destination queues can be routing or execution queues, and can be located at different PBS servers. For more information on creating and using routing queues, see [section 2.3.6, “Routing Queues”, on page 27](#).
- An execution queue is used as the home for a waiting or running job. A job must reside in an execution queue to be eligible to run. The job remains in the execution queue during the time it is running. See [section 2.3.5, “Execution Queues”, on page 25](#).

2.3.1.2 Available Kinds of Queues

PBS supplies the following kinds of execution and routing queues:

Table 2-3: Kinds of Queues

Kind of Queue		Description	Link
Routing queues		Used for moving jobs to another queue	See section 2.3.6, “Routing Queues”, on page 27
Execution queues	Reservation queues	Created for reservation. Do not operate on these directly; instead, operate on the reservation.	See section 2.3.5.2.iv, “Reservation Queues”, on page 27
	Dedicated time queues	Holds jobs that run only during dedicated time.	See section 2.3.5.2.i, “Dedicated Time Queues”, on page 26
	Primetime queues	Holds jobs that run only during primetime.	See section 2.3.5.2.ii, “Primetime and Non-Primetime Queues”, on page 26
	Non-primetime queues	Holds jobs that run only during non-primetime.	See section 2.3.5.2.ii, “Primetime and Non-Primetime Queues”, on page 26
	Anytime queues	Queue with no dedicated time or primetime restrictions	See section 2.3.5.2.iii, “Anytime Queues”, on page 26
	Express queues	High-priority queue; priority is set to the level signifying that it is an express queue	See section 2.3.5.3.i, “Express Queues”, on page 27
	Anti-express queue	Low-priority queue designed for work that should run only when no other jobs need the resources	See section 4.9.1, “Anti-Express Queues”, on page 107

2.3.2 Basic Queue Use

The simplest form of PBS uses just one queue. The queue is an execution queue named *workq*. This queue is always created, enabled, and started for you during installation. After a basic installation, this queue is ready to hold jobs submitted by users.

2.3.3 Creating Queues

To create a queue, use the `qmgr` command to create it and set its `queue_type` attribute:

```
Qmgr: create queue <queue name>
Qmgr: set queue <queue_name> queue_type = <execution or route>
```

For example, to create an execution queue named *exec_queue*, set its type, start it, and enable it:

```
Qmgr: create queue exec_queue
Qmgr: set queue exec_queue queue_type = execution
Qmgr: set queue exec_queue enabled = True
Qmgr: set queue exec_queue started = True
```

Now we will create a routing queue, which will send jobs to our execution queue:

```
Qmgr: create queue routing_queue
Qmgr: set queue routing_queue queue_type = route
Qmgr: set queue routing_queue route_destinations = exec_queue
```

2.3.4 Enabling, Disabling, Starting, and Stopping Queues

When you *enable* a queue, you allow it to accept jobs, meaning that jobs can be enqueued in the queue. When you *disable* a queue, you disallow it from accepting jobs. Queues are disabled by default. You enable a queue by setting its `enabled` attribute to *True*:

```
Qmgr: set queue <queue name> enabled = True
```

When you *start* a queue, you allow the jobs in the queue to be executed. Jobs are selected to be run according to the scheduling policy. When you *stop* a queue, you disallow jobs in that queue from running, regardless of scheduling policy. Except for the default queue, queues are stopped by default. You start a queue by setting its `started` attribute to *True*:

```
Qmgr: set queue <queue name> started = True
```

2.3.5 Execution Queues

Execution queues are used to run jobs; jobs must be in an execution queue in order to run. PBS does not route from execution queues.

2.3.5.1 Where Execution Queues Get Their Jobs

By default, PBS allows jobs to be moved into execution queues via the `qmove` command, by hooks, from routing queues, and by being submitted to execution queues. You can specify that an execution queue should accept only those jobs that are routed from a routing queue by PBS, by setting the queue's `from_route_only` attribute to *True*:

```
Qmgr: set queue <queue name> from_route_only = True
```

2.3.5.2 Execution Queues for Specific Time Periods

PBS provides a mechanism that allows you to specify that the jobs in an execution queue can run only during specific time periods. PBS provides a different kind of execution queue for each kind of time period. The time periods you can specify are the following:

Reservations

You can create an advance, standing, job-specific, or maintenance reservation. See [section 4.9.37, “Reservations”, on page 196](#).

Dedicated time

Dedicated time is a period of time with a defined beginning and end. You can define multiple dedicated times.

Primetime

Primetime is a recurring time period with a defined beginning and end. You can define primetime to be different for each day of the week.

Non-primetime

Non-primetime is a recurring time period with a defined beginning and end. Non-primetime begins when primetime ends, and vice versa.

Holidays

Holidays are dates defined in the `<sched_priv directory>/holidays` file. PBS provides an example file with everything commented out, and you define your own holidays and primetime. Holiday time is treated like non-primetime, meaning jobs in non-primetime queues run during holiday time.

Anytime queue

The term "anytime queue" means a queue that is not a primetime or a non-primetime queue.

2.3.5.2.i Dedicated Time Queues

The jobs in a dedicated time execution queue can run only during dedicated time. Dedicated time is defined in `<sched_priv directory>/dedicated_time`. See [section 4.9.10, “Dedicated Time”, on page 128](#).

To specify that a queue is a dedicated time queue, you prefix the queue name with the dedicated time keyword. This keyword defaults to `"ded"`, but can be defined in the `dedicated_prefix` scheduler parameter in `<sched_priv directory>/sched_config`. See [“dedicated_prefix” on page 252 of the PBS Professional Reference Guide](#).

2.3.5.2.ii Primetime and Non-Primetime Queues

The jobs in a primetime queue run only during primetime, and the jobs in a non-primetime queue run only during non-primetime. Primetime and non-primetime are defined in `<sched_priv directory>/holidays`. See [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).

To specify that a queue is a primetime or non-primetime queue, you prefix the queue name with the primetime or non-primetime keyword. For primetime, this keyword defaults to `"p_"`, and for non-primetime, the keyword defaults to `"np_"`, but these can be defined in the `primetime_prefix` and `nonprimetime_prefix` scheduler parameters in `<sched_priv directory>/sched_config`. See [“Scheduler Parameters” on page 251 of the PBS Professional Reference Guide](#).

2.3.5.2.iii Anytime Queues

An anytime queue is a queue whose jobs can run at any time. An anytime queue is simply a queue that is not a dedicated time, primetime, or non-primetime queue.

2.3.5.2.iv Reservation Queues

When the `pbs_rsub` command is used to create a reservation or to convert a job into a reservation job, PBS creates a reservation queue. Jobs in the queue run only during the reservation. Do not operate on these queues directly; instead, operate on the reservations. See [section 4.9.37, “Reservations”, on page 196](#).

2.3.5.3 Prioritizing Execution Queues

You can set the priority of each execution queue as compared to the other queues in this complex by specifying a value for the `priority` queue attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

A higher value for priority means the queue has greater priority. There is no limit to the priority that you can assign to a queue, however it must fit within integer size. See [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

For how queue priority is used in scheduling, see [section 4.9.36, “Queue Priority”, on page 195](#).

2.3.5.3.i Express Queues

A queue is an *express queue* if its priority is greater than or equal to the value that defines an express queue. This value is set in the `preempt_queue_prio` parameter in `<sched_priv directory>/sched_config`. The default value for `preempt_queue_prio` is `150`.

You do not need to set `by_queue` to `True` in order to use express queues.

For how express queues can be used, see [section 4.9.18, “Express Queues”, on page 139](#).

2.3.6 Routing Queues

A routing queue is used only to route jobs to other queues; jobs cannot run from a routing queue.

A routing queue has the following properties:

- Can route to multiple destination queues
- For each job, tries destination queues in the order listed, starting at the top of the list.
- Can route to execution queues
- Can route to other routing queues
- Can route to queues in other complexes (at other servers)

Destinations can be specified in the following ways:

```
route_destinations = Q1
route_destinations = Q1@Server1
route_destinations = "Q1, Q2@Server1, Q3@Server2"
route_destinations += Q1
route_destinations += "Q4, Q5@Server3"
```

2.3.6.1 How Routing Works

Whenever a job is in a started routing queue, PBS immediately attempts to route the job to a destination queue. When PBS routes a job, it starts at the top of the destination list and tries each destination in the order listed. The job's destination is the first queue that accepts it. The result is one of the following:

- The job is routed to one of the destination queues.
- The attempt to route is permanently rejected by each destination queue, and the job is deleted.
- Every destination queue rejects the job, but at least one rejection is temporary. In this case, the destination is tried again later, after the amount of time specified in the routing queue's `route_retry_time` attribute.
- If the job exceeds the time set in the queue's `route_lifetime` attribute, the job is deleted.

If there are multiple routing queues containing jobs to be routed, the routing queues are processed in the order in which they are displayed in the output of a `qstat -Q` command.

Queue priority does not play a role in routing jobs.

2.3.6.2 Requirements for Routing Queues

- A routing queue's destination queues must be created before being specified in the routing queue's `route_destinations` attribute.
- A routing queue's `route_destinations` attribute must be specified before enabling and starting the routing queue.
- A routing queue must be enabled in order to route jobs.

2.3.6.3 Caveats and Advice for Routing Queues

- Avoid routing loops. If a job makes more than 20 routing hops, it is discarded, and PBS sends mail to the job owner if the job's `Mail_Points` attribute contains "a" for "abort". Avoid setting a routing queue's destination to be the routing queue itself.
- When routing to a complex that is using failover, it's a good idea to include the names of both primary and secondary servers in a routing destination:
`route_destinations = "destQ@primary_server, destQ@secondary_server"`
- When routing a job between complexes, the job's owner must be able to submit a job to the destination complex.
- When routing to a destination in another complex, the source and destination complexes should use the same version of PBS. If not, you may need a submission hook to modify incoming jobs.
- It is recommended to list the destination queues in order of the most restrictive first, because the first queue which meets the job's requirements and is enabled will be its destination

2.3.6.4 Using Resources to Route Jobs Between Queues

You can use resources to direct jobs to the desired queues. The server will automatically route jobs that are in routing queues, based on job resource requests. The destination queue can be at the local server or at another server. If you have more than one PBS complex, you may want to route jobs between the complexes, depending on the resources available at each complex.

You can set up queues for specific kinds of jobs, for example jobs requesting very little memory, a lot of memory, or a particular application. You can then route jobs to the appropriate queues.

A routing queue tests destination queues in the order listed in the queue's `route_destinations` attribute. The job is placed in the first queue that meets the job's request and is enabled.

Please read all of the subsections for this section.

2.3.6.4.i How Queue and Server Limits Are Applied, Except Running Time

The following applies to all resources except for `min_walltime` and `max_walltime`.

You can set a minimum and a maximum for each resource at each queue using the `resources_min.<resource name>` and `resources_max.<resource name>` queue attributes. Any time a job is considered for entry into a queue, the job's resource request is tested against `resources_min.<resource name>` and `resources_max.<resource name>` for that queue. The job's resource request must be greater than or equal to the value specified in `resources_min.<resource name>`, and less than or equal to the value specified in `resources_max.<resource name>`.

The job is tested only against existing `resources_min.<resource name>` and `resources_max.<resource name>` for the queue.

Only those resources that are specified in the job's resource request are tested, so if a job does not request a particular resource, and did not inherit a default for that resource, the minimum and maximum tests for that resource are not applied to the job.

If you want jobs requesting only a specific value for a resource to be allowed into a queue, set the queue's `resources_min.<resource name>` and `resources_max.<resource name>` to the same value. This resource can be numeric, string, string array, or Boolean.

If you limit queue access using a string array, a job must request one of the values in the string array to be allowed into the queue. For example, if you set `resources_min.strarr` and `resources_max.strarr` to "blue,red,black", jobs can request `-l strarr=blue`, `-l strarr=red`, or `-l strarr=black` to be allowed into the queue.

2.3.6.4.ii How Queue and Server Running Time Limits are Applied

For shrink-to-fit jobs, running time limits are applied to `max_walltime` and `min_walltime`, not `walltime`. To set a running time limit for shrink-to-fit jobs, you cannot use `resources_max` or `resources_min` for `max_walltime` or `min_walltime`. Instead, use `resources_max.walltime` and `resources_min.walltime`. See [section 4.9.42.6, "Shrink-to-fit Jobs and Resource Limits"](#), on page 213.

2.3.6.4.iii Resources Used for Routing and Admittance

You can route jobs using the following kinds of resources:

- Any server-level or queue-level (job-wide) built-in or custom resource, whether it is numeric, string, or Boolean, for example `ncpus` and `software`
When routing jobs with `min_walltime` and/or `max_walltime`, PBS examines the values for `resources_min.walltime` and `resources_max.walltime` at the server or queue. See [section 2.3.6.4.ii, "How Queue and Server Running Time Limits are Applied"](#), on page 29.
- The following built-in chunk-level resources:

```
mem
mpiprocs
ncpus
nodect
vmem
```
- Custom vnode-level (chunk-level) resources that are global and have the `n`, `q`, or `f` flags set
- Any resource in the job's `Resource_List` attribute; see [section 5.9.2, "Resources Requested by Job"](#), on page 241. For string or string array resources, see [section 2.3.6.4.iv, "Using String, String Array, and Boolean Values for Routing and Admittance"](#), on page 30.

When jobs are routed using a chunk-level resource, routing is based on the sum of that resource across all chunks.

2.3.6.4.iv **Using String, String Array, and Boolean Values for Routing and Admittance**

When using strings or string arrays for routing or admittance, you can use only job-wide (server-level or queue-level) string or string array resources. String or string array resources in chunks are ignored. The `resources_min` and `resources_max` attributes work as expected with numeric values. In addition, they can be used with string and Boolean values to force an exact match; this is done by setting both to the same value. For example, to limit jobs entering queue big to those that specify `arch=unicos8`, or that do not specify a value for `arch`:

```
Qmgr: set q ApplQueue resources_max.software=App1
```

```
Qmgr: set q ApplQueue resources_min.software=App1
```

2.3.6.4.v **Examples of Routing Jobs**

You can force all jobs into a routing queue, or you can allow users to request some queues but not others. If you set up the default queue as a routing queue, and make all execution queues accept jobs only from routing queues, all jobs are initially forced into a routing queue.

Alternatively, you can set up one routing queue and a couple of execution queues which accept jobs only from routing queues, but add other queues which can be requested. Or you could allow jobs to request the execution queues, by making the execution queues also accept jobs that aren't from routing queues.

Example 2-1: Jobs can request one execution queue named *WorkQ*. All jobs that do not request a specific queue are routed according to their walltime:

- Create a routing queue *RouteQ* and make it the default queue:

```
Qmgr: create queue RouteQ queue_type = route
Qmgr: set server default_queue = RouteQ
```
- Create two execution queues, *LongQ* and *ShortQ*. One is for long-running jobs, and one is for short-running jobs:

```
Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution
```
- Set `resources_min.walltime` and `resources_max.walltime` on these queues:

```
Qmgr: set queue LongQ resources_min.walltime = 5:00:00
Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
```
- For *LongQ* and *ShortQ*, disallow jobs that are not from a route queue:

```
Qmgr: set queue LongQ from_route_only = True
Qmgr: set queue ShortQ from_route_only = True
```
- Set the destinations for *RouteQ* to be *LongQ* and *ShortQ*:

```
Qmgr: set queue RouteQ route_destinations = "ShortQ, LongQ"
```
- Create a work queue that can be requested:

```
Qmgr: create queue WorkQ queue_type = execution
```
- Enable and start all queues:

```
Qmgr: active queue RouteQ,LongQ,ShortQ,WorkQ
Qmgr: set queue enabled = True
Qmgr: set queue started = True
```
- Set default for walltime at the server so that jobs that don't request it inherit the default, and land in *ShortQ*:

```
Qmgr: set server resources_default.walltime = 4:00:00
```

Example 2-2: Jobs are not allowed to request any queues. All jobs are routed to one of three queues based on the job's walltime request:

- Create a routing queue *RouteQ* and make it the default queue:

```
Qmgr: create queue RouteQ queue_type = route
Qmgr: set server default_queue = RouteQ
```
- Create three execution queues, *LongQ*, *MedQ*, and *ShortQ*. One is for long-running jobs, one is for medium jobs, and one is for short-running jobs:

```
Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue MedQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution
```
- Set `resources_min.walltime` and `resources_max.walltime` on these queues:

```
Qmgr: set queue LongQ resources_min.walltime = 10:00:00
Qmgr: set queue MedQ resources_max.walltime = 9:59:00
Qmgr: set queue MedQ resources_min.walltime = 5:00:00
Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
```
- For *LongQ*, *MedQ*, and *ShortQ*, disallow jobs that are not from a route queue:

```
Qmgr: set queue LongQ from_route_only = True
Qmgr: set queue MedQ from_route_only = True
Qmgr: set queue ShortQ from_route_only = True
```

- Set the destinations for *RouteQ* to be *LongQ*, *MedQ* and *ShortQ*:
`Qmgr: set queue RouteQ route_destinations = "ShortQ, MedQ, LongQ"`
- Enable and start all queues:
`Qmgr: active queue RouteQ,LongQ,ShortQ,MedQ`
`Qmgr: set queue enabled = True`
`Qmgr: set queue started = True`

2.3.6.4.vi Caveats for Queue Resource Limits

If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has `resources_max.<resource name>` set, the job inherits that maximum value. If the queue has `resources_max.<resource name>` set, the job inherits the queue value, and if not, the job inherits the server value.

2.3.6.5 Using Access Control to Route Jobs

You can route jobs based on job ownership by setting access control limits at destination queues. A queue's access control limits specify which users or groups are allowed to have jobs in that queue. Default behavior is to disallow an entity that is not listed, so you need only list allowed entities.

To set the list of allowed users at a queue:

```
Qmgr: set queue <queue name> acl_users = "User1@*.example.com, User2@*.example.com"
```

To enable user access control at a queue:

```
Qmgr: set queue <queue name> acl_user_enable = True
```

To set the list of allowed groups at a queue:

```
Qmgr: set queue <queue name> acl_groups = "Group1, Group2"
```

To enable group access control at a queue:

```
Qmgr: set queue <queue name> acl_group_enable = True
```

For a complete explanation of access control, see [section 11.3, “Using Access Control Lists”, on page 492](#).

2.3.6.6 Allowing Routing of Held or Waiting Jobs

By default, PBS will not route jobs that are held. You can allow a routing queue to route held jobs by setting the queue's `route_held_jobs` attribute to *True*:

```
Qmgr: set queue <queue name> route_held_jobs = True
```

By default, PBS will not route jobs whose `execution_time` attribute has a value in the future. You can allow a routing queue to route jobs whose start time is in the future by setting the queue's `route_waiting_jobs` attribute to *True*:

```
Qmgr: set queue <queue name> route_waiting_jobs = True
```

2.3.6.7 Setting Routing Retry Time

The default time between routing retries is 30 seconds. To set the time between routing retries, set the value of the queue's `route_retry_time` attribute:

```
Qmgr: set queue <queue name> route_retry_time = <value>
```

2.3.6.8 Specifying Job Lifetime in Routing Queue

By default, PBS allows a job to exist in a routing queue for an infinite amount of time. To change this, set the queue's `route_lifetime` attribute:

```
Qmgr: set queue <queue name> route_lifetime = <value>
```

2.3.7 Queue Requirements

- Each queue must have a unique name. The name must be alphanumeric, and must begin with an alphabetic character
- A server may have multiple queues of either or both types, but the server must have at least one execution queue defined.

2.3.8 Queue Configuration Attributes

Queue configuration attributes fall into three groups:

- Those which apply to both types of queues
- Those which apply only to execution queues
- Those which apply only to routing queues

If an "execution queue only" attribute is set for a routing queue, or vice versa, it is ignored. However, as this situation might indicate the administrator made a mistake, the server will write a warning message on `stderr` about the conflict. The same message is written when the queue type is changed and there are attributes that do not apply to the new type.

See [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#) for a table of queue attributes.

2.3.9 Viewing Queue Status

To see the status of a queue, including values for attributes, use the `qstat` command:

```
qstat -Qf <queue name>
```

To see the status of all queues:

```
qstat -Qf
```

The status of the queue is reported in the *State* field. The field shows two letters. One is either *E* (enabled) or *D* (disabled.) The other is *R* (running, same as started) or *S* (stopped.) Attributes with non-default values are displayed. See [“qstat” on page 200 of the PBS Professional Reference Guide](#).

The following queue attributes contain queue status information:

```
total_jobs
state_count
resources_assigned
hasnodes (deprecated)
enabled
started
```

2.3.10 Deleting Queues

Use the `qmgr` command to delete queues.

```
Qmgr: delete queue <queue name>
```

2.3.10.1 Caveats for Deleting Queues

- A queue that has queued or running jobs cannot be deleted.
- The vnode queue attribute is **deprecated**. A queue that is associated with a vnode via that vnode's `queue` attribute cannot be deleted. To remove the association, save the output of `pbsnodes -a` to a file and search for the queue. Unset the `queue` attribute for each associated vnode.

2.3.11 Defining Queue Resources

For each queue, you can define the resources you want to have available at that queue. To set the value for an existing resource, use the `qmgr` command:

```
Qmgr: set queue <queue name> resources_available.<resource name> = <value>
```

For example, to set the value of the Boolean resource `RunsMyApp` to `True` at `QueueA`:

```
Qmgr: set queue QueueA resources_available.RunsMyApp = True
```

For information on how to define a new resource at a queue, see [section 5.14, “Custom Resources”, on page 252](#).

For information on defining default resources at a queue, see [section 5.9.3.3, “Specifying Job-wide Default Resources at Queue”, on page 243](#) and [section 5.9.3.2.ii, “Specifying Chunk Default Resources at Queue”, on page 242](#).

2.3.12 Setting Queue Resource Defaults

The jobs that are placed in a queue inherit the queue's defaults for any resources not specified by the job's resource request. You can specify each default resource for each queue. This is described in [section 5.9.3, “Specifying Job Default Resources”, on page 241](#). Jobs inherit default resources according to the rules described in [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#).

2.3.13 How Default Server and Queue Resources Are Applied When Jobs Move

When a job is moved from one server to another, the following changes happen:

- Any default resources that were applied by the first server are removed
- Default resources from the new server are applied to the job

When a job is moved from one queue to another, the following changes happen:

- Any default resources that were applied by the first queue are removed
- Default resources from the new queue are applied to the job

For more details on how default resources are inherited when a job is moved, see [section 5.9.4.3, “Moving Jobs Between Queues or Servers Changes Defaults”](#), on page 245.

2.3.14 Specifying Default Queue

PBS has a default execution queue named *workq*, where jobs are placed when no queue is requested. You can specify which queue should be the default. To specify the queue which is to accept jobs when no queue is requested, set the server's `default_queue` attribute to the name of the queue:

```
Qmgr: set server default_queue = <queue name>
```

2.3.15 Associating Queues and Vnodes

You can set up vnodes so that they accept jobs only from specific queues. See [section 4.9.2, “Associating Vnodes with Queues”](#), on page 107.

2.3.16 Configuring Access to Queues

You can configure each queue so that only specific users or groups can submit jobs to the queue. See [section 11.3, “Using Access Control Lists”](#), on page 492.

2.3.17 Setting Limits on Usage at Queues

You can set limits on different kinds of usage at each queue:

- You can limit the size of a job array using the `max_array_size` queue attribute
- You can limit the number of jobs or the usage of each resource by each user or group, or overall. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”](#), on page 283

2.3.18 Queues and Failover

For information on configuring routing queues and failover, see [section 8.2.6.1, “Configuring Failover to Work with Routing Queues”](#), on page 384.

2.3.19 Additional Queue Information

For a description of each queue attribute, see [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

For information on using queues for scheduling, see [section 4.6, “Using Queues in Scheduling”](#), on page 103.

Configuring MoMs and Vnodes

3.1 About MoMs

A MoM runs and manages the jobs on each execution host. The `pbs_mom` daemon starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, manages job file transfer, and notifies the server when the job is finished. When the MoM starts a job, she creates a new session that is as identical to the user's login session as is possible. For example, under Linux, if the user's login shell is `csh`, then MoM creates a session in which `.login` and `.cshrc` are run. MoM returns the job's output to the user. The MoM performs any communication with job tasks and with other MoMs. The MoM on the first vnode on which a job is running manages communication with the MoMs on the remaining vnodes on which the job runs. The MoM on the first vnode is called the *primary execution host MoM*.

The MoM writes a log file in `PBS_HOME/mom_logs`. The MoM writes an error message in its log file when it encounters any error. The MoM also writes other miscellaneous information to its log file. If it cannot write to its log file, it writes to standard error.

You start a MoM via the `pbs_mom` command. The executable for `pbs_mom` is in `PBS_EXEC/sbin`, and can be run only by root. For Linux, see [“MoMs: Starting, Stopping, Restarting” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“MoMs: Starting, Stopping, Restarting” on page 158 in the PBS Professional Installation & Upgrade Guide](#).

The MoM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

PBS supplies a hook that you can use to manage cgroups on each execution host, and via the hook, cpusets. See [Chapter 6, “Configuring and Using PBS with Cgroups”, on page 311](#). If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an `execjob_epilogue` hook instead.

3.1.1 Configuring MoMs

3.1.1.1 MoM Configuration File

During the installation process, PBS creates a Version 1 configuration file for each MoM. Each parameter in this file controls some aspect of MoM's behavior. To configure MoM's behavior, edit this file, and set each parameter as desired.

The default location for the Version 1 configuration file is on MoM's host, in `PBS_HOME/mom_priv/config`, or if `PBS_MOM_HOME` is defined, `PBS_MOM_HOME/mom_priv/config`. It can be in a different location; in that case, MoM must be started with the `-c` option. See [“pbs_mom” on page 71 of the PBS Professional Reference Guide](#).

If you add or change anything via a Version 1 configuration file, you can HUP the MoM, but if you remove anything, you must either restart the MoM so that the default value is re-applied, or change the removed value back to its default and then HUP the MoM. If you simply HUP the MoM after removing a line, MoM will not notice the removal.

The Version 1 configuration file must be secure. It must be owned by a user ID and group ID both less than 10 and must not be world-writable.

For a complete description of the syntax and contents of the Version 1 configuration file, see [“MoM Parameters” on page 243 of the PBS Professional Reference Guide](#).

3.1.1.2 Editing Version 1 Files

Use your favorite text editor to edit Version 1 configuration files.

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.1.1.3 Caveats and Restrictions for Configuration Files

- The `pbs_mom -d` option changes where MoM looks for `PBS_HOME`, and using this option will change where MoM looks for all configuration files. If you use the `-d` option, MoM will look in the new location for all MoM and vnode configuration files. Instead, we recommend setting the location of `PBS_HOME` or `PBS_MOM_HOME` in `/etc/pbs.conf` on MoM's host.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.1.1.4 When MoM Reads Configuration Files

MoM reads `pbs.conf` at startup, and her own configuration files at startup and reinitialization. On Linux, this is when `pbs_mom` receives a `SIGHUP` signal or is started or restarted, and on Windows, when MoM is started or restarted.

If you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS, such as a non-PBS process releasing a `cpuset`, you should restart PBS, by typing the following:

```
<path-to-script>/pbs restart
```

The MoM daemon is normally started by the PBS start/stop script.

When MoM is started, it opens its Version 1 configuration file, `mom_priv/config`, in the path specified in `pbs.conf`, if the file exists. If it does not, MoM will continue anyway. The `config` file may be placed elsewhere or given a different name, by starting `pbs_mom` using the `-c` option with the new file and path specified. See [“MoMs: Starting, Stopping, Restarting” on page 158 in the PBS Professional Installation & Upgrade Guide](#).

The files are processed in this order:

1. Version 1 configuration file
2. PBS reserved configuration files
3. Version 2 configuration files

Within each category, the files are processed in lexicographic order.

The contents of a file that is read later will override the contents of a file that is read earlier.

If there is an error in `mom_priv/config`, MoM will not start.

3.1.2 Configuring MoM Polling Cycle

3.1.2.1 Cgroups Hook Can Replace Polling

The cgroups hook (see [Chapter 6, "Configuring and Using PBS with Cgroups", on page 311](#)) can provide accurate accounting information and job resource usage management, so that MoM does not need to perform periodic job resource usage polling. If you use the cgroups hook to manage jobs at a host, MoM does not need to poll throughout the life of the job, and the server and the datastore experience less traffic.

Each time a MoM polls, the server rewrites all of the job's data to the datastore, causing traffic to the data store. If you have smaller jobs, MoM needs to poll often in order to get reasonably accurate information. If you have many of these jobs, this slows the server and reduces throughput.

Each job is always polled at the start and end, regardless of periodic polling. MoM polls at job end, before running the epilogue, when she detects that the last job task is done. If the job was spawned with `tm_spawn`, MoM can get an accurate value for `cput`. If the job was `tm_attached` and the `cgroups` hook is not running on the host, she cannot get an accurate value for `cput`, because a process other than MoM reaped the job. For example, if memory is reaped by something other than MoM, there is no way to get usage. However, if the `cgroups` hook is managing that job, the hook can get accurate usage.

If the `cgroups` hook manages the jobs at a host, MoM does not need to do any periodic job polling at that host.

3.1.2.2 Polling on Linux

MoM's polling cycle is determined by the values of `$min_check_poll` and `$max_check_poll` in the Version 1 configuration file. The interval between each poll starts at `$min_check_poll` and increases with each cycle until it reaches `$max_check_poll`, after which it remains the same. The amount by which the cycle increases is the following:

$$(\text{max_check_poll} - \text{min_check_poll} + 19) / 20$$

The default value for `$max_check_poll` is 120 seconds. The minimum is 1 second.

The default value for `$min_check_poll` is 10 seconds. The minimum is 1 second.

The start of a new job resets the polling for all of the jobs being managed by this MoM.

MoM polls for resource usage for `cput`, `walltime`, `mem` and `ncpus`.

3.1.2.2.i Linux Polling Caveats

Please note that polling intervals cannot be considered to be exact:

- The polling calculation simply provides a minimum amount of time between one poll and the next.
- The actual time between polls can vary. The actual time taken by MoM also depends on the other tasks MoM is performing, such as starting jobs, running a prologue or epilogue, etc.
- The timing of MoM's activities is not completely under her control, because she is a user process.
- The finest granularity for calculating polling is in seconds.

3.1.2.3 Polling on Windows

On Windows, MoM updates job usage at fixed intervals of 10 seconds. The `$min_check_poll` and `$max_check_poll` parameters are not used by MoM on Windows. MoM looks for any job that has exceeded a limit for `walltime`, `mem`, or `cput`, and terminates jobs that have exceeded the limit.

3.1.2.4 How Polling is Used

- Job-wide limits are enforced by MoM using polling. See [section 5.15.2.4.i, “Job Memory Limit Enforcement on Linux”, on page 302](#). MoM can enforce `cpuaverage` and `cpuburst` resource usage. See [section 5.15.2.5.i, “Average CPU Usage Enforcement”, on page 303](#) and [section 5.15.2.5.ii, “CPU Burst Usage Enforcement”, on page 304](#).
- MoM enforces the `$restrict_user` access restrictions on the polling cycle controlled by `$min_check_poll` and `$max_check_poll`. See [section 9.5.7, “Restricting User Access to Execution Hosts”, on page 438](#).
- Cycle harvesting has its own polling interval. See [“\\$kbd idle <idle wait> <min use> <poll interval>” on page 247 of the PBS Professional Reference Guide](#) for information on `$kbd_idle`.

3.1.2.5 Polling for Multi-host Jobs

Polling cycles are different on the primary execution host MoM and sister MoMs.

- The primary execution host MoM polls immediately when a task is started and again after the minimum polling period, then continues polling at each maximum polling period
- Sister MoMs poll a full cycle after the first task is created there

3.1.2.6 Recommendations for Polling Interval

Consider the workload at the host, and the overall workload at the server, when you set polling intervals. MoM's polling period should depend on the length of the typical job, and the importance for your site of accurate accounting. If you have many small jobs, frequent polling can take up a lot of MoM's cycles, and cause heavy traffic for the datastore and the server.

You may want to set `$min_check_poll` and `$max_check_poll` to somewhat higher values than the defaults. For example, for a 1-hour job, you could poll at 10-minute intervals. We do not recommend a value for `$max_check_poll` of less than 30 seconds. We do not recommend setting `$min_check_poll` to less than 10 seconds.

3.1.3 Files and Directories Used by MoM

If `PBS_MOM_HOME` is present in the `pbs.conf` file, `pbs_mom` will use that directory for its "home" instead of `PBS_HOME`.

3.1.3.1 Linux Files and Directories Used by MoM

Under Linux, all files and directories that MoM uses must be owned by root. MoM uses the following files and directories:

Table 3-1: MoM Files and Directories Under Linux

File/Directory	Description	Permissions
<code>/etc/pbs.conf</code>	File	<code>0644</code>
<code>aux</code>	Directory	<code>0755</code>
<code>checkpoint</code>	Directory	<code>0700</code>
<code>checkpoint script</code>	File	<code>0755</code>
<code>mom_logs</code>	Directory	<code>0755</code>
<code>mom_priv</code>	Directory	<code>0751</code>
<code>mom_priv/jobs</code>	Directory	<code>0751</code>
<code>mom_priv/config</code>	File	<code>0644</code>
<code>mom_priv/prologue</code>	File	<code>0755</code>
<code>mom_priv/epilogue</code>	File	<code>0755</code>
<code>pbs_environment</code>	File	<code>0644</code>
<code>spool</code>	Directory	<code>1777 (drwxrwxrwt)</code>
<code>undelivered</code>	Directory	<code>1777 (drwxrwxrwt)</code>

Table 3-1: MoM Files and Directories Under Linux

File/Directory	Description	Permissions
Version 2 configuration files (optional)	Files	0755
PBS reserved configuration files	Files	----
Job temporary directory	Directory	1777

3.1.3.2 Linux Files and Directories Used by MoM

Under Windows, these directories must have at least Full Control permission for the local Administrators group. MoM uses the following files and directories:

Table 3-2: MoM Files and Directories Under Windows

File/Directory	Description	Ownership/Permission
pbs.conf	File	
auxiliary	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
checkpoint	Directory	At least Full Control permission for the local Administrators group
checkpoint script	File	At least Full Control permission for the local Administrators group
mom_logs	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
mom_priv	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
mom_priv/jobs	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
mom_priv/config	File	At least Full Control permission for the local Administrators group
pbs_environment	File	At least Full Control permission for the local Administrators group and read-only to Everyone
spool	Directory	Full access to Everyone
undelivered	Directory	Full access to Everyone
Job's temporary directory	Directory	Writable by Everyone

3.2 About Vnodes: Virtual Nodes

A virtual node, or *vnode*, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, a NUMA node, a nodeboard, or a blade. A single host can be represented by one vnode or multiple vnodes. PBS views hosts as being composed of one or more vnodes, and PBS can manage and schedule each vnode independently. One PBS MoM manages all of the vnodes for each host.

3.2.1 Parent Vnodes and Child Vnodes

Each machine is represented by at least one vnode. The main vnode is called the *parent vnode*. Vnodes that represent machine resources such as CPUs are called *child vnodes*.

For single-vnode machines, the parent vnode is also the child vnode, and this vnode represents all of the machine's resources, including its hardware.

For machines with more than one vnode, the parent vnode does not correspond to any actual hardware; instead, it is a collection of information that applies to the host but not the individual vnodes, such as dynamic host-level resources and shared resources. On multi-vnode machines, resources such as CPUs are represented in child vnodes.

3.3 Creating Vnodes

3.3.1 Overview of Creating Vnodes

1. For each machine, you create one parent vnode using `qmgr`. See ["Creating the Parent Vnode" on page 44](#).
For a single-vnode machine, vnode creation is done.
2. For a machine which will have more than one vnode, after you create the parent vnode, PBS handles creation of the child vnodes:
 - If you run the `cgroups` hook with `vnode_per_numa_node` set to `true`, the `cgroups` hook creates all the local child vnodes. We recommend using the `cgroups` hook for hosts where you need to fence jobs in or take advantage of the topology to keep job processes on nearby resources. See ["Creating Child Vnodes via Cgroups Hook" on page 44](#).
 - If you are not using the `cgroups` hook to create child vnodes, you can have PBS create any child vnodes. You tell MoM which vnodes to create and how to set their attributes and resources by specifying them in a Version 2 configuration file. See ["Creating Child Vnodes via Version 2 Configuration File" on page 44](#).
3. After all vnodes have been created, you can set vnode attributes and resources if necessary. See ["Configuring Vnodes" on page 45](#).

3.3.2 How to Choose Vnode Names

MoM needs to know what name you will use for the parent vnode when she starts up. So if you decide to use a non-default name, define the name before starting MoM.

By default, the `cgroups` hook and MoM use the non-canonicalized hostname returned by `gethostname()` for the host as the vnode name. If you use the hostname, use the part before the first dot. You can use the `hostname()` command without any extra flags to get the hostname:

```
hostname<return>
```

For example, if this returns "myhost.mydomain", use "myhost".

You can choose the name for the parent vnode, such as an alias, or a name bound to another IP address on the host.

You can use the IP address as the name of the parent vnode.

To use any parent vnode name that is not the default, you must specify the name by setting the `PBS_MOM_NODE_NAME` parameter in the host's `/etc/pbs.conf`. For example, if you use a name that has a dot in it and you don't set `PBS_MOM_NODE_NAME`, hooks will fail.

If you've already started MoM, then in order for MoM to be able to use the non-default name, you need to make the name available to her, then restart her. For example, to use the IP address:

1. Add `PBS_MOM_NODE_NAME=<IP address>` to `pbs.conf` on the execution host
2. Restart MoM

When `PBS_MOM_NODE_NAME` is defined, MoM performs a sanity check to ensure that the value is a resolvable host.

3.3.2.1 Names of Child Vnodes

If the cgroups hook creates child vnodes, it creates them with the same name as the parent vnode, with an index number. For example, on a machine with two NUMA nodes where each NUMA node is represented by a vnode, you'll have the parent vnode plus two child vnodes; if the parent vnode is named "myhost", they are named "myhost[0]" and "myhost[1]".

If you create child vnodes via a Version 2 configuration file, each vnode in your complex must have a unique name within the complex. We recommend using or at least including the name of the parent vnode, to ensure uniqueness and to make vnodes recognizable. Do not use square brackets for anything but the index.

3.3.2.2 Caveats for Vnode Names

- Do not change the name of the parent vnode after you create it
- If there is a dot in the name, you must set `resources_available.host` by hand; otherwise the part after the dot is stripped
- If you use an IP address as the name of a vnode, you must set it in `PBS_MOM_NODE_NAME`
- You cannot use a vnode attribute as the name of a vnode
- Vnode names are case-insensitive
- If you create a vnode with a different name from the short name returned by `hostname`, and you don't set it in `PBS_MOM_NODE_NAME`, the following happens:
 - MoM creates a vnode whose name is the short name returned by `hostname ( )`
 - The vnode you created is not recognized by MoM, and is marked stale

3.3.2.3 Errors and Logging for Vnode Names

- If `PBS_MOM_NODE_NAME` is unset and the call to `gethostname ( )` fails, or if `PBS_MOM_NODE_NAME` is set and the value does not conform to RFCs 952 and 1123, the following message is printed to the MoM log:
`Unable to obtain my host name`
- Once the hostname is obtained, MoM ensures the hostname resolves properly. If the hostname fails to resolve, the following message is printed to the MoM log:
`Unable to resolve my host name`

3.3.3 Creating the Parent Vnode

1. Make sure MoM can look up the name of the parent vnode when she starts. Follow the rules in [section 3.3.2, “How to Choose Vnode Names”, on page 42](#). Choose the name for the parent vnode:
 - If you will use the default, make sure that `PBS_MOM_NODE_NAME` is not set, or set it to the default. To get the default name, run this at MoM's host, and use the part before the dot:
`hostname<return>`
 - If you will use a non-default name, set it in `PBS_MOM_NODE_NAME` in `/etc/pbs.conf` on the MoM host. For example, to use the IP address for the name of the vnode, add this to `/etc/pbs.conf` on the execution host:
`PBS_MOM_NODE_NAME=<IP address>`

2. Start MoM using `systemd` or the PBS start/stop script:

```
systemctl start pbs
```

or

```
<path to script>/pbs start
```

For details on starting and stopping MoM, see [“Methods for Starting, Stopping, or Restarting PBS” on page 144 in the PBS Professional Installation & Upgrade Guide](#).

3. Use the `qmgr` command to create the parent vnode:

```
qmgr -c 'create node <vnode name> [<attribute>=<value>]'
```

All comma-separated attribute-value strings must be enclosed in quotes:

```
qmgr -c 'create node <vnode name> ["<attribute>=<value>, <attribute>=<value>"]'
```

Attributes and their possible values are listed in [“Vnode Attributes” on page 321 of the PBS Professional Reference Guide](#).

3.3.4 Creating Child Vnodes for Multi-vnode Machines

3.3.4.1 Creating Child Vnodes via Cgroups Hook

If you are running the cgroups hook with `vnode_per_numa_node` set to `true`, the hook creates the local child vnodes.

1. If you have not done so yet, create the parent vnode; see [Chapter 3, “Creating the Parent Vnode”, on page 44](#).
2. Configure and enable the cgroups hook. Follow all the instructions in [Chapter 6, “Configuring and Using PBS with Cgroups”, on page 311](#).
3. Restart the MoM:

```
<path to PBS start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

3.3.4.2 Creating Child Vnodes via Version 2 Configuration File

1. If you have not done so yet, create the parent vnode; see [Chapter 3, “Creating the Parent Vnode”, on page 44](#).
2. If you are not using the cgroups hook, you can create any child vnodes by defining the child vnodes you want in a Version 2 configuration file, so MoM creates the vnodes for you.

Note that in prior versions of PBS, a Version 2 configuration file was the preferred method for advanced GPU configuration (see ["Advanced GPU Scheduling" on page 280](#)). As of version 2020.1, the cgroups hook makes it much easier for job submitters to request exclusive use of GPUs. However, if you want to continue to use Version 2 configuration files for managing GPUs, you can do so.

See [section 3.4.3.1, "Creating Version 2 Configuration Files", on page 47](#).

3. Restart the MoM:

```
<path to PBS start/stop script>/pbs restart
or
systemctl restart pbs
```

4. Check for stale vnodes. Make sure you spell "Stale" with a capital S:

```
qmgr -c 'print node @default' | grep "Stale"
```

3.3.5 Caveats for Creating Vnodes

When using `qmgr` to create vnodes, create only the parent vnode on each host. Do not use `qmgr` to create child vnodes on a multi-vnode host; MoM will not know about these, and cannot use them.

3.4 Configuring Vnodes

Each vnode has an associated set of attributes and resources, such as CPUs, memory, and partition. Vnode attributes are listed and described in ["Vnode Attributes" on page 321 of the PBS Professional Reference Guide](#). Vnode resources can be built-in or custom (defined by you.) See [Chapter 5, "Using PBS Resources", on page 227](#).

3.4.1 Methods for Configuring Vnodes

You may need to configure vnodes after you create them. You can use the following methods:

- Using `exechost_startup` hooks to set vnode attributes and resources

This method is powerful and flexible. You can interrogate the host; for example, you can check whether a vnode exists before setting values for it. You can use this to set the `sharing` attribute and `resources_available.host`. If the cgroups hook creates your vnodes, make sure that the cgroups hook runs before the hook that configures the vnodes. Your `exechost_startup` hooks run when MoM is restarted. See ["Setting and Unsetting Vnode Resources and Attributes" on page 49 in the PBS Professional Hooks Guide](#).

- Using Version 2 vnode configuration files, either to modify vnodes created by the cgroups hook, or to tell MoM to create the vnodes you specify. See [section 3.4.3, "Version 2 Vnode Configuration Files", on page 46](#).

Make sure that a Version 2 configuration file matches your available vnodes every time MoM is started. If your machine reboots with a missing blade, your earlier placement set information will not make sense because child vnode names will not match the available hardware. You can use a script to regenerate this file each time the machine starts, and run the script before MoM is restarted.

You can use a Version 2 configuration file to set the `sharing` attribute and the value of `resources_available.host` (you cannot set these via `qmgr`).

An advantage of using a Version 2 configuration file is that if you delete and re-create the parent vnode, you don't have to re-create this file. MoM automatically picks up everything in a Version 2 configuration file, whereas if you use `qmgr` you have to re-run all your configuration commands.

If you use the cgroups hook to create child vnodes, and you want to modify these child vnodes, make sure you create the Version 2 configuration file after the vnodes are created, and that you use the exact vnode names that the cgroups hook knows about, by checking the output of `pbsnodes -av`.

If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.

Version 2 configuration files are read when MoM is restarted. See [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).

- Using the `qmgr` command to set vnode attribute and resource values

You can easily use `qmgr` to set values across your complex. You cannot use this to set the `sharing` attribute or `resources_available.host`. If you delete and re-create a vnode, the effects of your configuration commands are lost. Changes take place immediately. See [“qmgr” on page 152 of the PBS Professional Reference Guide](#).

- Using the `pbsnodes -o` or `pbsnodes -r` command to mark all vnodes on a host as offline or not offline

You must use `qmgr` to change the state of a single vnode in a multi-vnode host. Changes take place immediately. See [“pbsnodes” on page 36 of the PBS Professional Reference Guide](#).

3.4.2 Rules for Configuring Vnodes

- If you are using the cgroups hook to create child vnodes and manage subsystems, do not change attribute or resource values that are set by the cgroups hook.
- To set the `sharing` attribute or `resources_available.host`, you must use an `exehost_startup` hook or a Version 2 configuration file. You cannot use `qmgr`. See [section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 50](#).
- Set the `Mom` attribute for the parent vnode only. You can set the initial value only via `qmgr -c 'create node <vnode name>'` to tell the server at what IP address MoM is located. The server will later update it based on the MoM's response. The server only queries for the canonicalized address of the MoM host, unless you let it know via the `Mom` attribute; if you have set `PBS_LEAF_NAME` in `/etc/pbs.conf` to something else, make sure you set the `Mom` attribute at vnode creation.

3.4.3 Version 2 Vnode Configuration Files

Version 2 configuration files contain settings for vnode attributes and resources. For example, to change the `sharing` vnode attribute or `resources_available.host`, you can use a Version 2 configuration file, or an `exehost_startup` hook, but not `qmgr`. You can use more than one Version 2 configuration file per host, but make sure they do not conflict.

PBS places Version 2 configuration files in an area that is private to each installed instance of PBS.

It's best to automate updates to Version 2 configuration files so that they are created at boot time to match available hardware, because a change in hardware may create a mismatch with an old Version 2 configuration file. This ensures that a Version 2 configuration file matches your available hardware every time MoM is started. If your machine reboots with a missing blade, your earlier placement set information will not make sense because child vnode names will be not match the available hardware. You can use a script to regenerate this file each time the machine starts, and run the script before MoM is restarted.

An advantage of using a Version 2 configuration file is that if you delete and re-create the parent vnode, you don't have to re-create this file. MoM automatically picks up everything in a Version 2 configuration file, whereas if you use `qmgr` you have to rerun all your configuration commands.

3.4.3.1 Creating Version 2 Configuration Files

Version 2 configuration files are created by PBS, through a process where you write a source file and then PBS copies it to the location where Version 2 files are used. Instead of editing one of these directly, you create an input file and give it as an argument to the `pbs_mom -s insert` option on the local host (`pbs_mom -N -s insert` on Windows), and PBS creates a new configuration file for you.

You use the `pbs_mom -s insert` command to create Version 2 configuration files. On Windows, use the `pbs_mom` command in standalone mode: `pbs_mom -N -s insert`.

First, you create an input file which is to be the contents of the configuration file. Then, you use the `pbs_mom -s insert` command, on the host you want to configure:

Linux:

```
pbs_mom -s insert <Version 2 filename> <input file>
```

Windows:

```
pbs_mom -N -s insert <Version 2 file> <input file>
```

After you create the new Version 2 configuration file, restart the MoM.

3.4.3.1.i Syntax of Version 2 Configuration Files

In a Version 2 configuration file, you tell PBS that it's a Version 2 MoM configuration file by putting a special tag on the first line:

```
$configversion 2
```

The rest of the file describes vnodes, with one attribute specification per line.

The format of the remaining contents of the file is the following:

```
<vnode name> : <attribute name> = <attribute value>
```

where

<vnode name>

Sequence of characters not including a colon (":"). The vnode name must be unique in this PBS complex. Vnode names are case-insensitive. See [“Vnode Name” on page 358 of the PBS Professional Reference Guide](#).

If you're modifying vnodes created by the cgroups hook, the *vnode name* must exactly match the output of `pbsn-odes -av`.

<attribute name>

Name of the attribute being specified. See [“Non-resource Formats” on page 353 of the PBS Professional Reference Guide](#).

<attribute value>

Value being specified. Sequence of characters not including an equal sign ("="). See [“Resource Formats” on page 359 of the PBS Professional Reference Guide](#).

White space around the colon and equal sign is ignored.

In a Version 2 configuration file, do not use quotes around string array values. This is different from using the `qmgr` command; in the `qmgr` command line, you need to put quotes around the value.

Make sure that the first vnode entry is for the parent vnode. If you don't need to set anything, you can set the `ntype` attribute to "PBS" (the default).

Make sure that there is a newline at the end of the file. Under Windows, the Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

Make sure that entries do not conflict, whether within one file or multiple files.

Do not use Version 1 (MoM configuration file) syntax or contents in Version 2 files, and vice versa.

3.4.3.1.ii Example of Creating Version 2 Configuration File

Example 3-1: If your machine named "myhost" has 4 vnodes, where two are big (myhost[0] and myhost[1]), and two are small (myhost[2] and myhost[3]), and you want big jobs to have exclusive use of myhost[0] and myhost[1], and small jobs to share myhost[2] and myhost[3]:

- a. Set sharing for big and small vnodes by creating a file "set_sharing" containing the following:

```
$configversion 2
myhost: ntype = PBS
myhost[0]: sharing = default_excl
myhost[0]: resources_available.nodetype=big
myhost[1]: sharing = default_excl
myhost[1]: resources_available.nodetype=big
myhost[2]: sharing = default_shared
myhost[2]: resources_available.nodetype=small
myhost[3]: sharing = default_shared
myhost[3]: resources_available.nodetype=small
```

- b. Use the `pbs_mom -s insert <Version 2 filename> <input file>` option at myhost to create its configuration file:

Linux:

```
pbs_mom -s insert sharing_config set_sharing
```

Windows:

```
pbs_mom -N -s insert sharing_config set_sharing
```

PBS creates the new Version 2 configuration file called "sharing_config". Its contents will override previously-read sharing settings.

- c. Restart the MoM after changing the configuration file:

Linux:

```
kill -INT <MoM PID>
PBS_EXEC/sbin/pbs_mom
```

or

```
systemctl restart pbs
```

or

```
<path to start/stop script>/pbs restart
```

Windows:

```
net stop pbs_mom
net start pbs_mom
```

Jobs can then request `nodetype = big` or `nodetype=small`, or you can use a hook to route jobs, etc.

3.4.3.2 Listing and Viewing Version 2 Configuration Files

You can list and view the Version 2 configuration files at each host.

To see the list of Version 2 configuration files:

Linux:

```
pbs_mom -s list
```

Windows:

```
pbs_mom -N -s list
```

To display the contents of a Version 2 configuration file:

Linux:

```
pbs_mom -s show <Version 2 filename>
```

Windows:

```
pbs_mom -N -s show <Version 2 filename>
```

See [“pbs_mom” on page 71 of the PBS Professional Reference Guide](#).

3.4.3.3 Moving Version 2 Configuration Files

To move a set of Version 2 configuration files from one MoM host to another:

1. List the Version 2 files at the source instance:

```
pbs_mom -s list
```

2. Save a copy of each file at the source instance:

```
pbs_mom -s show <Version 2 filename> > <input file>
```

3. Create the new Version 2 configuration files at the destination host. For each file:

```
pbs_mom -s insert <Version 2 filename> <input file>
```

3.4.3.4 Removing Version 2 Configuration Files

You can remove a Version 2 configuration file:

Linux:

```
pbs_mom -s remove <Version 2 filename>
```

Windows:

```
pbs_mom -N -s remove <Version 2 filename>
```

See [“pbs_mom” on page 71 of the PBS Professional Reference Guide](#).

3.4.3.5 Caveats for Version 2 Configuration Files

- If you are using the cgroups hook to create child vnodes at a host, and you use a Version 2 configuration file to modify those child vnodes:
 - Make sure that you use exactly the same vnode names in the Version 2 configuration file as those that the cgroups hook has created; check the output of `pbsnodes -av`.
 - Do not use a Version 2 configuration file to change hardware settings for that host.
- If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.
- The `pbs_mom -d` option changes where MoM looks for `PBS_HOME`, and using this option will change where MoM looks for all configuration files. If you use the `-d` option, MoM will look in the new location for all MoM and vnode configuration files. Instead, we recommend setting the location of `PBS_HOME` or `PBS_MOM_HOME` in `/etc/pbs.conf` on MoM's host.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.4.3.6 PBS Reserved Configuration Files

PBS reserved configuration files are created by PBS and are prefixed with "PBS". You cannot create or modify a configuration file whose name begins with "PBS". Do not move PBS reserved configuration files.

3.4.4 Configuring the Vnode Sharing Attribute

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode's `sharing` attribute and the placement requested by a job.

You can set each vnode's `sharing` attribute so that the vnode or host is always shared, is always exclusive, or so that it honors the job's placement request. If the vnode attribute is set to `force_shared` or `force_excl`, the value of a vnode's `sharing` attribute takes precedence over a job's placement request. If the vnode attribute is set to `default_`, the job request overrides the vnode attribute.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs. If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its `sharing` attribute set to either `default_exclhost` or `force_exclhost`, all vnodes on that host must have the same value for the `sharing` attribute.

For a complete description of the `sharing` attribute, and a table showing the interaction between the value of the `sharing` attribute and the job's placement request, see [“sharing” on page 325 of the PBS Professional Reference Guide](#).

3.4.4.1 Sharing on a Multi-vnode Machine

On a multi-vnode shared-memory machine, a scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

To keep a job in a single vnode, use `-lplace=group=vnode`; if you want to restrict it to larger sets of vnodes, identify those sets using a custom `string` or `string_array` resource and use it in `-lplace=group=<resource>`. If you already have resources used in `node_group_key` you can usually use these.

3.4.4.2 Setting the sharing Vnode Attribute

To set the `sharing` attribute for a vnode, use either:

- An `execlhost_startup` hook; see ["Setting and Unsetting Vnode Resources and Attributes" on page 49 in the PBS Professional Hooks Guide](#)
- A Version 2 configuration file; see [section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 50](#)

3.4.4.3 Viewing Sharing Information

You can use the `qmgr` or `pbsnodes` commands to view sharing information. See [“qmgr” on page 152 of the PBS Professional Reference Guide](#) and [“pbsnodes” on page 36 of the PBS Professional Reference Guide](#).

3.4.4.4 Sharing Caveats

- The term "sharing" is also used to describe the case where MoM manages a resource that is shared among her vnodes, for example an application license shared by the vnodes of a multi-vnode machine.
- The term "sharing" is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are "sharing" a CPU. See [section 8.6.6, “Managing Load Levels on Vnodes”, on page 415](#)
- If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.
- For vnodes with *sharing=default_shared*, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The *exec_vnode* attribute will show this allocation.

3.4.5 Configuring Vnode Resources

Before configuring vnode (host-level) resources, consider how you will use them. When configuring static resources, it is best to configure global static resources. Even though they are global, they can be configured at the host level. Global resources can be operated on via the *qmgr* command and viewed via the *qstat* command. When configuring dynamic resources, if you need the script to run at the execution host, configure local dynamic resources. These resources cannot be operated on via the *qmgr* command or viewed via the *qstat* command. See [section 5.4, “Categories of Resources”, on page 230](#).

3.4.5.1 Configuring Global Static Vnode Resources

You can create global custom static host-level resources that can be reported by MoM and used for jobs. Follow the instructions in [section 5.14.4.2, “Static Host-level Resources”, on page 265](#).

You can set values for built-in and custom global static vnode resources; see [section 3.4.5, “Configuring Vnode Resources”, on page 51](#).

3.4.5.2 Configuring Local Dynamic Vnode Resources

You can create local custom dynamic host-level resources. The primary use of this feature is to add site-specific resources, such as software application licenses or scratch space. Follow the instructions in [section 5.14.4.1, “Dynamic Host-level Resources”, on page 265](#).

3.4.5.3 Rules for Configuring Vnode Resources

- In general, it is not advisable to set `resources_available.ncpus` or `resources_available.mem` to a value greater than PBS has detected on the machine. This is because you do not want MoM to try to allocate more resources than are available. However, if you have lots of I/O-bound jobs, you might get away with oversubscribing CPUs.
- In general, it is safe to set `resources_available.ncpus` or `resources_available.mem` to a value less than PBS has detected. If you are using a Version 2 configuration file, consider setting `ncpus` lower to set aside some of the resource for the operating system.
- For the parent vnode on a multi-vnode machine, set all values for `resources_available.<resource name>` to zero (0), unless the resource is being shared among child vnodes via indirection. Here is an example of the vnode definition for a parent vnode:


```
host03: pnames = cbrick, router
host03: sharing = ignore_excl
host03: resources_available.ncpus = 0
host03: resources_available.mem = 0
host03: resources_available.vmem = 0
```
- When MoM creates a vnode, she automatically sets values for the following resources according to information from the host:


```
resources_available.ncpus
resources_available.arch
resources_available.mem
```
- If you set the value of a resource via `qmgr`, that setting is carried forth across server restarts.
- If you add or change a value via a Version 2 configuration file, you can HUP the MoM. If you remove a value, you must restart MoM so that she uses the default. (Hint: to avoid restarting MoM, use the configuration file to set the default.)
- You can set values for the `sharing` attribute and `resources_available.host` only in an `exehost_startup` or `exehost_periodic` hook, or in a Version 2 configuration file. You cannot use `qmgr` to set these.
- Version 2 configuration files take effect before `exehost_startup` hooks.

3.4.6 Configuring Vnodes via the `qmgr` Command

You can use the `qmgr` command to set attribute and resource values for individual vnodes, for single-vnode and multi-vnode machines.

To set a vnode's attribute:

```
qmgr -c 'set node <vnode name> <attribute> = <value>'
```

We describe the `qmgr` command in [“qmgr” on page 152 of the PBS Professional Reference Guide](#).

3.4.6.1 Caveats for Setting Values via qmgr Command

- When setting hardware resources, be careful about setting these to values that are higher than what MoM or the cgroups hook did. If you have lots of I/O-bound jobs, you might get away with oversubscribing CPUs.
- It is usually safe to set hardware resources to values lower than what MoM or the cgroups hook did.
- If you are not using the cgroups hook, consider setting `ncpus` to a slightly lower value than what MoM reports, to give some to the operating system.
- If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.
- You cannot set the value of `resources_available.host` via the `qmgr` command.

3.4.7 Configuring Vnodes via the pbsnodes Command

You can use the `pbsnodes` command to set the state all of the vnodes on a host to be *offline* or not *offline*. To set the state attribute of one or more hosts to *offline*:

```
pbsnodes -o <hostname [hostname ...]>
```

To remove the *offline* setting from the state attribute of one or more hosts:

```
pbsnodes -r <hostname [hostname ...]>
```

See [“pbsnodes” on page 36 of the PBS Professional Reference Guide](#).

3.4.7.1 Caveats for pbsnodes Command

For multi-vnode hosts, the `pbsnodes` command operates on all of the host's vnodes only. You cannot use it on individual vnodes where those vnodes are on multi-vnode machines. To operate on individual vnodes, use the `qmgr` command:

```
qmgr -c 'set node <vnode name> state = <new state>'
```

When you specify a hostname, the `pbsnodes` command looks for the value of a vnode's `resources_available.host` resource. If this is different from the `PBS_MOM_NODE_NAME` parameter, it may be helpful to use a Version 2 configuration file to set `resources_available.host` to match the `PBS_MOM_NODE_NAME` parameter (you cannot use `qmgr` for this).

Make sure that `resources_available.host` is unique for each host in your complex.

The `pbsnodes -o <target host>` command offlines everything with a matching `resources_available.<target host>`.

3.5 Deleting Vnodes

3.5.1 Deleting the Vnode on a Single-vnode Machine

Use the `qmgr` command to delete the vnode:

```
qmgr: delete node <vnode name>
```

3.5.2 Deleting Vnodes on a Multi-vnode Machine

3.5.2.1 Deleting Vnodes When Not Using Version 2 Configuration File

4. Use the `qmgr` command to delete the vnodes:

```
Qmgr: delete node <vnode name>
```

3.5.2.2 Deleting Vnodes When Using Version 2 Configuration File

To delete one or more vnodes on a multi-vnode machine where there is a Version 2 configuration file, you must first remove the configuration file. Then you can delete the vnodes. You may want to save the existing configuration file and edit it down to just the vnodes you want to preserve. On the local host:

1. To see the list of Version 2 configuration files:

Linux:

```
pbs_mom -s list
```

Windows:

```
pbs_mom -N -s list
```

2. To save the contents of a Version 2 configuration file in "tempconfig":

Linux:

```
pbs_mom -s show <Version 2 filename> > tempconfig
```

Windows:

```
pbs_mom -N -s show <Version 2 filename> > tempconfig
```

3. Edit `tempconfig` so that it describes only the vnodes you want to keep.
4. Use `pbs_mom -s remove` to remove the old Version 2 configuration file:

On Linux:

```
pbs_mom -s remove <Version 2 filename>
```

On Windows:

```
pbs_mom -N -s remove <Version 2 filename>
```

5. Use `pbs_mom -s insert` to create a new Version 2 configuration file describing the vnodes to be retained. If you created `tempconfig`, that is your input file:

On Linux:

```
pbs_mom -s insert <Version 2 filename> <input file>
```

On Windows:

```
pbs_mom -N -s insert <Version 2 filename> <input file>
```

6. Restart the MoM:

```
<path to start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

7. Use the `qmgr` command to remove the vnodes no longer appearing in your configuration file:

```
Qmgr: delete node <vnode name>
```

8. Check for stale vnodes. Make sure you spell "Stale" with a capital S:

```
qmgr -c 'print node @default' | grep "Stale"
```


4

Scheduling

The "[Scheduling Policy Basics](#)" section of this chapter describes what PBS can do, so that you can consider these capabilities when choosing how to schedule jobs. The "[Choosing a Policy](#)" section describes how PBS can meet the scheduling needs of various workloads. The "[Scheduling Tools](#)" section describes each scheduling tool offered by PBS.

4.1 Chapter Contents

4.1	Chapter Contents	57
4.2	Scheduling Each Partition Separately	59
4.2.1	Creating and Configuring a Multisched	59
4.2.2	Starting a Multisched	60
4.2.3	Configuring Your Partitions for Multischeds	61
4.2.4	Using the Default Scheduler with Multischeds	61
4.2.5	Multisched Caveats and Restrictions	62
4.2.6	Attributes Used with Multischeds	62
4.2.7	Multisched Errors and Logging	64
4.2.8	Multisched Deprecations	65
4.3	Scheduling Policy Basics	66
4.3.1	How Scheduling Can Be Used	66
4.3.2	What Is Scheduling Policy?	66
4.3.3	Basic PBS Scheduling Behavior	66
4.3.4	Sub-goals	67
4.3.5	Job Prioritization and Preemption	67
4.3.6	Resource Allocation to Users, Projects & Groups	73
4.3.7	Time Slot Allocation	75
4.3.8	Job Placement Optimization	76
4.3.9	Resource Efficiency Optimizations	79
4.3.10	Overrides	81
4.3.11	Configurable Job Dispatch Rate	82
4.4	Choosing a Policy	83
4.4.1	Overview of Kinds of Policies	83
4.4.2	FIFO: Submission Order	83
4.4.3	Prioritizing Jobs by User, Project or Group	84
4.4.4	Allocating Resources by User, Project or Group	84
4.4.5	Scheduling Jobs According to Size Etc.	86
4.4.6	Scheduling Jobs into Time Slots	88
4.4.7	Default Scheduling Policy	90
4.4.8	Examples of Workload and Policy	92
4.5	About Schedulers	93
4.5.1	Configuring a Scheduler	93
4.5.2	Making a Scheduler Read its Configuration	99
4.5.3	Scheduling on Resources	99
4.5.4	Specifying Scheduler Username	99
4.5.5	Starting, Stopping, and Restarting a Scheduler	99
4.5.6	The Scheduling Cycle	100
4.5.7	How Available Consumable Resources are Counted	101

4.5.8	Improving Scheduler Performance	102
4.6	Using Queues in Scheduling	103
4.7	Scheduling Restrictions and Caveats	103
4.7.1	One Policy Per Scheduler	103
4.7.2	Jobs that Cannot Run on Current Resources	103
4.7.3	Resources Not Controlled by PBS	104
4.7.4	No Pinning of Processes to Cores.	104
4.8	Errors and Logging	104
4.8.1	Logfile for scheduler	104
4.9	Scheduling Tools	104
4.9.1	Anti-Express Queues	107
4.9.2	Associating Vnodes with Queues	107
4.9.3	Using Backfilling	109
4.9.4	Examining Jobs Queue by Queue	113
4.9.5	Checkpointing	114
4.9.6	Organizing Job Chunks	115
4.9.7	cron Jobs	115
4.9.8	Using Custom and Default Resources	116
4.9.9	Using Idle Workstation Cycle Harvesting	117
4.9.10	Dedicated Time	128
4.9.11	Dependencies	129
4.9.12	Dynamic Resources	129
4.9.13	Eligible Wait Time for Jobs	129
4.9.14	Sorting Jobs by Entity Shares (Was Strict Priority)	133
4.9.15	Estimating Job Start Time	133
4.9.16	Calculating Job Execution Priority	136
4.9.17	Calendaring Jobs	138
4.9.18	Express Queues	139
4.9.19	Using Fairshare	139
4.9.20	FIFO Scheduling	150
4.9.21	Using a Formula for Computing Job Execution Priority	151
4.9.22	Gating Jobs at Server or Queue	157
4.9.23	Managing Application Licenses	158
4.9.24	Limits on Per-job Resource Usage	158
4.9.25	Limits on Project, User, and Group Jobs	159
4.9.26	Limits on Project, User, and Group Resource Usage	159
4.9.27	Using Load Balancing	159
4.9.28	Matching Jobs to Resources	159
4.9.29	Node Grouping	161
4.9.30	Overrides	162
4.9.31	Peer Scheduling	164
4.9.32	Placement Sets	168
4.9.33	Using Preemption	180
4.9.34	Using Primetime and Holidays	190
4.9.35	Provisioning	195
4.9.36	Queue Priority	195
4.9.37	Reservations	196
4.9.38	Round Robin Queue Selection	204
4.9.39	Routing Jobs	205
4.9.40	Scheduler Cycle Speedup	209
4.9.41	Shared vs. Exclusive Use of Resources by Jobs	210
4.9.42	Using Shrink-to-fit Jobs	211
4.9.43	SMP Cluster Distribution	217
4.9.44	Using Soft Walltime	218

4.9.45	Sorting Jobs on a Key	220
4.9.46	Sorting Jobs by Requested Priority	222
4.9.47	Sorting Queues into Priority Order	222
4.9.48	Using Strict Ordering	223
4.9.49	Sorting Vnodes on a Key	224

4.2 Scheduling Each Partition Separately

You can leave your complex as a single default partition, or you can divide your complex into partitions, and run a separate scheduler for each partition. PBS automatically creates a default partition containing all queues and vnodes that are not explicitly labeled with a partition name. You can create named partitions, simply by labeling each queue and vnode with the desired partition. You can choose whether to assign each queue and vnode to a specific partition, or to have it remain as part of the default partition.

PBS has two kinds of schedulers:

- A default scheduler that handles the workload for the default partition (all queues and vnodes that have not been explicitly assigned to a named partition)

The default scheduler runs its own scheduling policy.

You cannot assign any non-default partitions to the default scheduler.

The default scheduler runs only on the server host.

- A *multisched* that handles a named, non-default partition

Each multisched runs its own scheduling policy, and can schedule jobs for one named partition.

A multisched requires at least one queue and one vnode in its partition in order to be able to schedule jobs.

Multischeds cannot share partitions.

A multisched can run on any host.

A named partition is a collection of vnodes labeled with a partition name, along with one or more queues also labeled with the same partition name. A vnode can be in at most one partition. You can put some or all of your vnodes into partitions, where they will be scheduled by the multisched assigned to the partition. You can also leave vnodes out of named partitions; those vnodes will be scheduled by the default scheduler.

You can have as many named partitions and multischeds as you want. Each partition can have only one multisched. Each partition requires at least one execution queue.

Each multisched schedules only from the queue(s) in its partition, and only to the vnode(s) in its partition. Jobs do not span partitions.

You can define a unique policy for each scheduler.

4.2.1 Creating and Configuring a Multisched

4.2.1.1 Prerequisites for Creating a Multisched

You must be a PBS administrator or Manager to create a scheduler.

You must supply a name when you create a multisched. The maximum length for the name is 15 characters.

Before you start a multisched, you must create the `sched_priv` and `sched_log` directories for it.

- The default name for the `sched_priv` directory is `sched_priv <multisched name>`, and the default location is on the server/scheduler host, directly under `PBS_HOME`, alongside the `sched_priv` of the default scheduler. You can set the name and location as desired, but we recommend keeping it in `PBS_HOME`. The `sched_priv` directory should have permissions 750, and should be readable and writable by the multisched. It should be owned by root. It cannot be shared with another multisched.
- Populate the `sched_priv` directory with the following:
 - `sched_config`
Required.
 - `holidays`
Required.
 - `resource_group`
Necessary for fairshare tree.
 - `dedicated_time`
Required.

We provide default copies of these files in `PBS_EXEC/etc`.
- The default name and location for the multisched logging directory is `sched_logs_<multisched name>` (note the plural), on the server/scheduler host, directly under `PBS_HOME`, alongside the `sched_logs` of the default scheduler. You can set the name and location as desired, but we recommend keeping it in `PBS_HOME`. The `sched_log` directory should have permissions 755, and should be readable and writable by the multisched. It should be owned by root. It cannot be shared with another multisched.

4.2.1.2 Creating a Multisched

You use the `qmgr` command to create a scheduler:

```
qmgr -c "create sched <multisched name>"
```

For example:

```
qmgr -c "create sched multisched_1"
```

This creates a multisched with its attributes set to the defaults.

4.2.1.3 Configuring a Multisched

You must set the partition multisched attribute:

```
qmgr -c "set sched <multisched name> partition = <partition name>"
```

4.2.1.4 Enabling a Multisched

To enable a multisched, set its scheduling attribute to `True`:

```
qmgr -c "set sched <scheduler name> scheduling = 1"
```

4.2.2 Starting a Multisched

Do not start a multisched until:

- Its `sched_priv` directory is ready. See [section 4.2.1.1, “Prerequisites for Creating a Multisched”, on page 59](#)
- You have assigned it a partition. See [section 4.2.1.3, “Configuring a Multisched”, on page 60](#)

4.2.2.1 Starting a Multisched on Linux

Start a multisched by calling `pbs_sched` and specifying the name you already gave it:

```
pbs_sched -I <name of multisched>
```

For example:

```
pbs_sched -I multisched_1
```

When you start a multisched, you must specify its name.

4.2.3 Configuring Your Partitions for Multischeds

To schedule using partitions, compose each partition and start its multisched:

- Put each vnode into at most one partition, or leave it in the default partition; partitions cannot share vnodes. If putting the vnode into a named partition, set the value of the `partition` vnode attribute to the name of its partition:

```
qmgr -c set node <vnode name> partition=<partition name>
```

For example:

```
qmgr -c set node <vnode1> partition=part1
```

- Assign at least one execution queue to each named partition: set the value of the `partition` queue attribute to the name of its partition:

```
qmgr -c set queue <queue name> partition=<partition name>
```

For example:

```
qmgr -c set queue <queue1> partition=part1
```

- Create the desired multisched. See [section 4.2.1, “Creating and Configuring a Multisched”, on page 59](#).
- Assign a multisched to each named partition: set the value of the `partition` multisched attribute to the name of its partition:

```
qmgr -c "set sched <multisched name> partition=<partition name>"
```

For example:

```
qmgr -c "set sched multisched_1 partition=part1"
```

- Enable the multisched:

```
qmgr -c 'set sched <multisched name> scheduling=1'
```

For example:

```
qmgr -c 'set sched multisched_1 scheduling=1'
```

- Start the multisched:

```
pbs_sched -I <name of multisched>
```

For example:

```
pbs_sched -I multisched_1
```

4.2.4 Using the Default Scheduler with Multischeds

PBS automatically creates the default scheduler; its name is "default". The `sched_priv` directory of the default scheduler is always `$PBS_HOME/sched_priv`. The default scheduler writes its logs in `$PBS_HOME/sched_logs`. If you do nothing, the default scheduler uses the default scheduling policy defined in the default `sched_config` file. Default behavior is described in [section 4.4.7, “Default Scheduling Policy”, on page 90](#). You can set the desired policy for the default scheduler. The default scheduler schedules jobs using queues and vnodes in the default partition.

4.2.4.1 Configuring the Default Scheduler

When you use the `qmgr` command to configure the default scheduler, specify its name:

```
qmgr -c "set sched default <attribute> = <value>"
```

For example:

```
qmgr -c "set sched default job_sort_formula_threshold = <value>"
```

4.2.5 Multisched Caveats and Restrictions

- You cannot delete the default scheduler.
- You cannot change the name of the default scheduler.
- You cannot set `sched_priv` for the default scheduler.
- If you create a new queue or vnode without assigning it to a specific partition, it is scheduled by the default scheduler.
- You cannot assign a new multisched to a partition that is already assigned to a multisched, or vice versa. To make the change, offline the vnodes, wait for jobs to finish running, then un-assign and re-assign the multisched or partition.
- You cannot change a queue to a routing queue when the queue has its `partition` attribute set.
- You cannot associate a vnode and a queue and assign them separate partitions.
- If there is more than one scheduler, job run limits for the entire complex set at the server are not supported. Queue limits are enforced.
- All schedulers in the complex have as a default value for `backfill_depth` the value that is set at the server. For each scheduler, this is overridden by the setting at a scheduler's queue.
- All schedulers in the complex use the same value for `job_sort_formula`, so all schedulers use the same formula.
- If the complex has more than one scheduler, you cannot use complex-wide fairshare. Each scheduler manages its own fairshare tree.

4.2.6 Attributes Used with Multischeds

partition

Scheduler attribute. Partition for which this scheduler is to run jobs. Cannot be set on default scheduler.

Format: *String*

Default: *"None"*

partition

Vnode attribute. Name of partition to which this vnode is assigned. A vnode can be assigned to at most one partition.

Settable by Manager and Operator, viewable by all.

Format: *String*

partition

Queue attribute. Name of partition to which this queue is assigned.

Cannot be set for routing queue.

An execution queue cannot be changed to a routing queue while this attribute is set.

Settable by Manager, administrator, viewable by all.

Format: *String*

sched_log

Scheduler attribute. Directory where this scheduler writes its logs. Permissions should be **755**. Must be owned by root. Cannot be shared with another scheduler. For default scheduler, directory is always `PBS_HOME/sched_log`.

Settable by Manager, administrator, viewable by all.

Default: `$PBS_HOME/sched_logs_<scheduler name>`

sched_host

Scheduler attribute. Hostname on which scheduler runs.

Default value for default scheduler is set by server to server hostname.

Settable by Manager or administrator, viewable by all.

scheduling

Scheduler attribute. Enables scheduling of jobs. If you set the server's **scheduling** attribute, that value is assigned to the default scheduler's **scheduling** attribute, and vice versa.

Settable by Manager or administrator, viewable by all.

Default value for default scheduler: *True*

Default value for multisched: *False*

scheduler_iteration

Scheduler attribute. Time between scheduling iterations. If you set the server's **scheduler_iteration** attribute, that value is assigned to the default scheduler's **scheduler_iteration** attribute, and vice versa.

Settable by Manager, administrator, viewable by all.

Default: *600*

sched_priv

Scheduler attribute. Directory where this scheduler keeps fairshare usage, **resource_group**, **holidays**, and **sched_config**. Must be owned by root. For default scheduler, directory is always `PBS_HOME/sched_priv`.

Settable by Manager or administrator, viewable by all.

Default: `$PBS_HOME/sched_priv_<scheduler name>`

state

Scheduler attribute. State of this scheduler.

Set by server. Readable by all.

Valid values: one of *down*, *idle*, *scheduling*

down: scheduler is not running

idle: scheduler is running and is waiting for a scheduling cycle to be triggered

scheduling: scheduler is running and is in a scheduling cycle

Format: *String*

Default value for default scheduler: *idle*

Default value for multisched: *down*

comment

Scheduler attribute. Can be set by PBS or administrator. For certain scheduler errors, PBS sets the scheduler's **comment** attribute to specific error messages. You can use the **comment** field to notify another administrator of something, but PBS does overwrite the value of **comment** under certain circumstances.

Format: *String*

4.2.6.1 Behavior of Attributes Shared by Server and Scheduler

If you set the server's **scheduling** or **scheduler_iteration** attributes, the changes are applied to the default scheduler, and its corresponding **scheduling** or **scheduler_iteration** attributes are given the new setting(s). The reverse is also true.

4.2.7 Multisched Errors and Logging

4.2.7.1 Multisched Error Messages Appearing in Scheduler Comment

A scheduler's **comment** attribute can be set to specific error messages.

- Setting the **sched_log** attribute to an invalid value produces a scheduler comment. If the log directory is not accessible by the scheduler:
Unable to change the **sched_log** directory
In addition, the **scheduling** attribute is set to *False*.
- Attempting to set the **sched_priv** attribute to an invalid value:
Unable to change the **sched_priv** directory
In addition, the **scheduling** attribute is set to *False*.
- Setting **sched_priv** to a directory that fails validation checking produces a scheduler comment. If the **sched_priv** directory is not accessible by the scheduler, or the scheduler files are not found in the directory:
PBS failed validation checks for **sched_priv** directory
In addition, the **scheduling** attribute is set to *False*.

4.2.7.2 Multisched Error Messages Appearing in Scheduler Logs

- Attempting to start a multisched before assigning it a partition:
Scheduler does not contain a partition
- When the partition has been removed from a multisched, the multisched is shut down.
Scheduler does not contain a partition
- When the scheduler cannot get its attribute information from the server:
Unable to retrieve the scheduler attributes from server

4.2.7.3 Multisched Error Messages Appearing in Server Logs

- Attempting to set `sched_priv` for the default scheduler:
Operation is not permitted on default scheduler
- Attempting to set the partition for the default scheduler:
Operation is not permitted on default scheduler
In addition, the error code is set to 15223.
- Attempting to assign a `sched_priv` to a multisched while that directory is already assigned to another multisched:
Another scheduler has same value for its `sched_priv` directory
In addition, the error code is set to 15216.
- Attempting to assign a `sched_log` to a multisched while that directory is already assigned to another multisched:
Another scheduler has same value for its `sched_log` directory
In addition, the error code is set to 15215.
- If the server is not able to reach a scheduler one of the following messages appears:
Unable to reach scheduler associated with partition [`<partition ID>`]
Unable to reach scheduler associated with job `<job ID>`

4.2.7.4 Multisched Errors Returned by `qmgr` Command

- Attempting to associate a vnode with a queue that has not been assigned to the same partition:
`qmgr obj=<vnode name> svr=<server name>: Partition <partition name> is not part of queue for node`
`qmgr: Error (15220) returned from server`
- Attempting to assign a partition to a vnode when that vnode is associated with a queue and the queue is not assigned to the same partition:
`qmgr obj=<vnode name> svr=<server name>: Queue <queue name> is not part of partition for node`
`qmgr: Error (15219) returned from server`
- When a queue is associated with one or more vnodes, and a partition is assigned to the queue and vnodes, attempting to change the queue's partition:
`qmgr obj=<queue name> svr=<server name>: Invalid partition in queue`
`qmgr: Error (15221) returned from server`
- Attempting to assign a partition to a multisched while that partition is already assigned to another multisched:
Partition `<partition name>` is already associated with scheduler `<scheduler name>`.
- Attempting to set the `partition` attribute for a routing queue:
`qmgr obj=<queue name> svr=<server name>: Cannot assign a partition to route queue`
`qmgr: Error (15217) returned from server`
- Attempting to change an execution queue to a routing queue while the `partition` attribute is set:
`qmgr obj=<queue name> svr=<server name>: Cannot queue_type=route if partition is set`
`qmgr: Error (15218) returned from server`

4.2.8 Multisched Deprecations

Using `qmgr` on the default scheduler, without specifying its name, is deprecated. The old syntax is supported for backward compatibility.

Example of old syntax:

```
qmgr -c "set sched job_sort_formula_threshold = <value>"
```

Same with new syntax:

```
qmgr -c "set sched default job_sort_formula_threshold = <value>"
```

4.3 Scheduling Policy Basics

4.3.1 How Scheduling Can Be Used

You can use the scheduling tools provided by PBS to implement your chosen scheduling policy, so that your jobs run in the way you want.

Your policy can do the following:

- Prioritize jobs according to your specification
- Run jobs according to their relative importance
- Award specific amounts of resources such as CPU time to projects, users, and groups according to rules that you set
- Make sure that resources are not misused
- Optimize how jobs are placed on vnodes, so that jobs run as efficiently as possible
- Use special time slots for particular tasks
- Optimize throughput or turnaround time for jobs

4.3.2 What Is Scheduling Policy?

Scheduling policy determines when each job is run and on which resources. In other words, a scheduling policy describes a goal, or intended behavior. For convenience, we describe a scheduling policy as being a combination of sub-goals, for example a combination of how resources should be allocated and how efficiency should be maximized.

You implement a scheduling policy using the tools PBS provides. A scheduling tool is a feature that allows you control over some aspect of scheduling. For example, the job sorting formula is a tool that allows you to define how you want job execution priority to be computed. Some scheduling tools are supplied by the PBS scheduler(s), and some are supplied by other elements of PBS, such as the hooks, server, queues or resources.

You can group the resources in your complex into partitions, and you can run a separate scheduling policy on each partition.

4.3.3 Basic PBS Scheduling Behavior

The basic behavior of PBS is that it always places jobs where it finds the resources requested by the job. PBS will not place a job where that job would use more resources than PBS thinks are available. For example, if you have two jobs, each requesting 1 CPU, and you have one vnode with 1 CPU, PBS will run only one job at a time on the vnode. You do not have to configure PBS for this basic behavior.

PBS determines what hardware resources are available and configures them for you. However, you do have to inform PBS which custom resources and non-hardware resources are available and where, how much, and whether they are consumable or not. In addition, in order to ensure that jobs are sent to the appropriate vnodes for execution, you also need to make sure that they request the correct resources. You can do this either by having users submit their jobs with the right resource requests, using hooks that set job resources, or by configuring default resources for jobs to inherit.

4.3.4 Sub-goals

Your scheduling policy is the combination that you choose of one or more sub-goals. For example, you might need to meet two particular sub-goals: you might need to prioritize jobs a certain way, and you might need to use resources efficiently. You can choose among various outcomes for each sub-goal. For example, you can choose to prioritize jobs according to size, owner, owner's usage, time of submission, etc.

In the following sections, we describe the tools PBS offers for meeting each of the following sub-goals.

- Job prioritization and preemption; see [section 4.3.5, “Job Prioritization and Preemption”, on page 67](#).
- Resource allocation & limits; see [section 4.3.6, “Resource Allocation to Users, Projects & Groups”, on page 73](#).
- Time slot allocation; see [section 4.3.7, “Time Slot Allocation”, on page 75](#).
- Job placement optimizations; see [section 4.3.8, “Job Placement Optimization”, on page 76](#).
- Resource efficiency optimizations; see [section 4.3.9, “Resource Efficiency Optimizations”, on page 79](#).
- Overrides; see [section 4.3.10, “Overrides”, on page 81](#).

4.3.5 Job Prioritization and Preemption

Job prioritization is any technique you use to come up with a ranking of each job's relative importance. You can specify separate priority schemes for both execution and preemption.

4.3.5.1 Job Priority Terminology

Execution Priority

Used in determining order in which jobs are executed. Calculated by PBS. Determined by job class and then by sorting parameters specified by administrator; this can include the job sorting formula. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#) and [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

Preemption Priority

Used in determining which jobs can preempt other jobs. Specified by administrator. See [section 4.9.33, “Using Preemption”, on page 180](#).

The Priority Job Attribute

Value that can be assigned by job submitter in order to indicate relative job priority. Can be used as input for the following:

- When calculating execution and/or preemption priority
- In the job sorting formula
- As a sort key for job sorting jobs via the `job_sort_key` scheduler parameter

The `job_priority` Keyword

This keyword represents the value of the Priority job attribute, and is used in the job sorting formula and the `job_sort_key` scheduler parameter.

4.3.5.2 Where PBS Uses Job Priority

PBS calculates job priority for two separate tasks: job execution and job preemption. Job execution priority is used with other factors to determine when to run each job. Job preemption priority is used to determine which queued jobs are allowed to preempt which running jobs in order to use their resources and run. These two tasks are independent, and it is important to make sure that you do not make them work at cross-purposes. For example, you do not want to have a class of jobs having high execution priority and low preemption priority; these jobs would run first, and then be preempted first.

Preemption comes into play when a scheduler examines the top job and determines that it cannot run now. If preemption is enabled, a scheduler checks to see whether the top job has sufficient preemption priority to be able to preempt any running jobs, and then if it does, whether preempting jobs would yield enough resources to run the top job. If both are true, a scheduler preempts running jobs and runs the top job.

If you take no action to configure how jobs should be prioritized, they are considered in submission order, one queue at a time. If you don't prioritize queues, the queues are examined in an undefined order.

4.3.5.3 Overview of Prioritizing Jobs

PBS provides several tools for setting job execution priority. There are queue-based tools for organizing jobs, moving them around, and specifying the order in which groups of jobs should be examined. There are tools for sorting jobs into the order you want. There is a meta-tool (strict ordering) that allows you to specify that the top job must go next, regardless of whether the resources it requires are available now.

A scheduler can use multiple sorting tools, in succession. You can combine your chosen sorting tools with queue-based tools to give a wide variety of behaviors. Most of the queue-based tools can be used together. A scheduler can treat all jobs as if they are in a single queue, considering them all with respect to each other, or it can examine all queues that have the same priority as a group, or it can examine jobs queue by queue, comparing each job only to other jobs in the same queue.

You can change how execution priority is calculated, depending on which time slot is occurring. You can divide time up into primetime, non-primetime, and dedicated time.

When a scheduler calculates job execution priority, it uses a built-in system of job classes. PBS runs special classes of jobs before it considers queue membership. These classes are for reservation, express, and preempted jobs. Please see [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#). After these jobs are run, a scheduler follows the rules you specify for queue behavior. Within each queue, jobs are sorted according to the sorting tools you choose.

4.3.5.4 Using Queue-based Tools to Prioritize Jobs

4.3.5.4.i Using Queue Order to Affect Order of Consideration

When a scheduler examines queued jobs, it can consider all of the jobs in its partition as a whole, it can round-robin through groups of queues where the queues are grouped by priority, or it can examine jobs in only one queue at a time. These three systems are incompatible. Queues are always sorted by priority.

The `by_queue` scheduler parameter controls whether a scheduler runs all the jobs it can from the highest-priority queue before moving to the next, or treats all jobs as if they are in a single queue. By default, this parameter is set to *True*. When examining jobs one queue at a time, a scheduler runs all of the jobs it can from the highest-priority queue first, then moves to the next highest-priority queue and runs all the jobs it can from that queue, and so on. See [section 4.9.4, “Examining Jobs Queue by Queue”, on page 113](#).

The `round_robin` scheduler parameter controls whether or not a scheduler round-robins through queues. When a scheduler round-robins through queues, it groups the queues by priority, and round-robins first through the highest-priority group, then the next highest-priority group, and so on, running all of the jobs that it can from that group. So within each group, if there are multiple queues, a scheduler runs the top job from one queue, then the top job from the next queue, and so on, then goes back to the first queue, runs its new top job, goes to the next queue, runs its new top job, and so on until it has run all of the jobs it can from that group. All queues in a group must have exactly the same priority. The order in which queues within a group are examined is undefined. If all queues have different priorities, a scheduler starts with the highest-priority queue, runs all its jobs, moves to the next, runs its jobs, and so on until it has run all jobs from each queue. This parameter overrides `by_queue`. See [section 4.9.38, “Round Robin Queue Selection”, on page 204](#).

If you want queues to be considered in a specific order, you must assign a different priority to each queue. Queues are always sorted by priority. See [section 4.9.47, “Sorting Queues into Priority Order”, on page 222](#). Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. If you want groups of queues to be considered together for round-robinning, give all queues in each group one priority, and all queues in the next group a different priority. If the queues don't have priority assigned to them, the order in which they are considered is undefined. To set a queue's priority, use the `qmgr` command to assign a value to the `priority` queue attribute. See [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#).

4.3.5.4.ii Using Express Queues in Job Priority Calculation

You can create express queues, and route jobs into them, if you want to give those jobs special priority.

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the `preempt_queue_prio` scheduler attribute. For information on configuring express queues, see [section 2.3.5.3.i, “Express Queues”, on page 27](#).

When calculating execution priority, a PBS scheduler uses a built-in job class called “*Express*” which contains all jobs that have a preemption level greater than that of the `normal_jobs` level. By default, those jobs are jobs in express queues. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

You can create preemption levels that include jobs in express queues. Jobs in higher preemption levels are allowed to preempt jobs in lower levels. See [section 4.9.33, “Using Preemption”, on page 180](#).

4.3.5.4.iii Routing Jobs into Queues

You can configure PBS to automatically put each job in the most appropriate queue. There are several approaches to this. See [section 4.9.39, “Routing Jobs”, on page 205](#).

4.3.5.4.iv Using Queue Priority when Computing Job Priority

You can configure a scheduler so that job priority is partly determined by the priority of the queue in which the job resides. See [section 4.9.36, “Queue Priority”, on page 195](#).

4.3.5.5 Using Job Sorting Tools to Prioritize Jobs

A scheduler can use multiple job sorting tools in succession to determine job execution priority. A scheduler groups all jobs waiting to run into classes, and then applies the sorting tools you choose to all jobs in each class.

- You can create a formula that a scheduler uses to sort jobs. A scheduler applies this formula to all jobs in its partition, using it to calculate a priority for each job. For example, you can specify in the formula that jobs requesting more CPUs have higher priority. If the formula is defined, it overrides fairshare and sorting jobs on keys. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).
- You can use the fairshare algorithm to sort jobs. This algorithm allows you to set a resource usage goal for users or groups. Jobs are prioritized according to each entity's usage; jobs whose owners have used the smallest percentage of their allotment go first. For example, you can track how much CPU time is being used, and allot each group a percentage of the total. See [section 4.9.19, “Using Fairshare”, on page 139](#).
- You can sort jobs according to the same usage allotments you set up for fairshare. In this case, jobs whose owners are given the highest allotment go first. See [section 4.9.14, “Sorting Jobs by Entity Shares \(Was Strict Priority\)”, on page 133](#).
- You can sort jobs on one or more keys, for example, you can sort jobs first by the number of CPUs they request, then by the amount of memory they request. You can specify that either the high or the low end of the resulting sort has higher priority.

You can create a custom resource, and use a hook to set a value for that resource for each job, and then sort on the resource.

See [section 4.9.45, “Sorting Jobs on a Key”, on page 220](#).

- You can run jobs in the order in which they were submitted. See [section 4.9.20, “FIFO Scheduling”, on page 150](#).
- You can run jobs according to the priority requested for each job at submission time. This priority can be set via a hook. See [section 4.9.46, “Sorting Jobs by Requested Priority”, on page 222](#) and the PBS Professional Hooks Guide.

4.3.5.6 Prioritizing Jobs by Wait Time

You can use the amount of time a job has been waiting to run in the priority calculation. You use eligible waiting time, which is how long a job has been waiting to run due to a shortage of resources, rather than because its owner isn't allowed to run jobs now. See [section 4.9.13, “Eligible Wait Time for Jobs”, on page 129](#).

You can use a job's eligible waiting time in the job sorting formula. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

4.3.5.7 Calculating Preemption Priority

Execution priority and preemption priority are two separate systems of priority.

By default, if the top job cannot run now, and it has high preemption priority, a scheduler will use preemption to run the top job. A scheduler will preempt jobs with lower preemption priority so that it can use the resources to run the top job. The default definition of jobs with high preemption priority is jobs in express queues. You can configure many levels of preemption priority, specifying which levels can preempt which other levels. See [section 4.9.33, “Using Preemption”, on page 180](#).

4.3.5.8 Making Preempted Jobs Top Jobs

You can specify that a scheduler should make preempted jobs be top jobs. See [section 4.9.3.6, “Configuring Backfilling”, on page 111](#).

4.3.5.9 Preventing Jobs from Being Preempted

You may have jobs that should not be preempted, regardless of their priority. These can be jobs which cannot be effectively preempted, so that preempting them would waste resources. To prevent these jobs from being preempted, do one or both of the following:

- Set a value for the `preempt_targets` resource at all jobs that specify a value for a custom resource. For example, define a Boolean resource named `Preemptable`, and add `"Resource_List.Preemptable=true"` to `preempt_targets` for all jobs. Then set the value of `Resource_List.Preemptable` to *False* for the jobs you don't want preempted.

For example, if we want JobA and JobB to be able to preempt Job1 and Job2, but not Job3:

Define a Boolean resource named "Preemptable"

For Job1 and Job2, set `Resource_List.Preemptable` to *True*:

```
qsub ... -l Preemptable=True ...
```

or

```
qalter -l Preemptable=True Job1 Job2
```

For Job3, set `Resource_List.Preemptable` to *False*:

```
qalter -l Preemptable=False Job3
```

For JobA and JobB, set `Resource_List.preempt_targets` to `"Preemptable=True"`:

```
qalter -l preempt_targets=Resource_List.Preemptable=True JobA JobB
```

- Route jobs you don't want preempted to one or more specific queues, and then use a hook to make sure that no jobs specify these queues in their `preempt_targets`.

4.3.5.10 Meta-priority: Running Jobs Exactly in Priority Order

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. If a job cannot run now because the resources required are unavailable, the default behavior is to skip the job and move to the next in order of priority.

You can tell PBS to use a different behavior called *strict ordering*. This means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

You can see that using strict ordering could lead to decreased throughput and idle resources. In order to prevent idle resources, you can tell PBS to run small filler jobs while it waits for the resources for the top job to become available. These small filler jobs do not change the start time of the top job. See [section 4.9.48, "Using Strict Ordering", on page 223](#) and [section 4.9.3, "Using Backfilling", on page 109](#).

4.3.5.11 Using Different Calculations for Different Time Periods

PBS allows you to divide time into two kinds, called *primetime* and *non-primetime*. All time is covered by one or the other of these two kinds of time. The times are arbitrary; you can set them up however you like. You can also choose not to define them, and instead to treat all time the same.

You can configure two separate, independent ways of calculating job priority for primetime and non-primetime. The same calculations are used during dedicated time; dedicated time is a time slot made up of primetime and/or non-primetime. Many scheduler parameters are prime options, meaning that they can be configured separately for primetime and non-primetime. For example, you can configure fairshare as your sorting tool during primetime, but sort jobs on a key during non-primetime.

If you use the formula, it is in force all of the time.

See [section 4.9.34, "Using Primetime and Holidays", on page 190](#).

4.3.5.12 When Priority Is Not Enough: Overrides

Sometimes, the tools available for setting job priority don't do everything you need. For example, it may be necessary to run a job right away, regardless of what else is running. Or you may need to put a job on hold. Or you might need to tweak the way the formula works for the next N jobs. See [section 4.9.30, “Overrides”, on page 162](#).

4.3.5.13 Elements to Consider when Prioritizing Jobs

- Whether users, groups, or projects affect job priority: for techniques to use user, group, or project to affect job priority, see [section 4.4.3, “Prioritizing Jobs by User, Project or Group”, on page 84](#).
- Reservation jobs: jobs in reservations cannot be preempted.
- Express jobs: PBS has a built-in execution priority for express jobs. You can set the preemption priority for express jobs; see [section 4.9.33, “Using Preemption”, on page 180](#).
- Preempted jobs: PBS has a built-in execution priority for preempted jobs. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- Large or small jobs: you may want to give large and/or small jobs special treatment. See [section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 86](#).
- User's priority request for job: the job submitter can specify a priority for the job at submission. You can sort jobs according to each job's specified priority. See [section 4.9.46, “Sorting Jobs by Requested Priority”, on page 222](#).
- Whether the top job must be the next to run, regardless of whether it can run now; see [section 4.9.48, “Using Strict Ordering”, on page 223](#).

4.3.5.14 List of Job Sorting Tools

4.3.5.14.i Queue-based Tools for Organizing Jobs

- Queue-by-queue: PBS runs all the jobs it can from the first queue before moving to the next queue. Queue order is determined by queue priority. See [section 4.9.4, “Examining Jobs Queue by Queue”, on page 113](#).
- Round-robin job selection: PBS can select jobs from queues with the same priority in a round-robin fashion. See [section 4.9.38, “Round Robin Queue Selection”, on page 204](#).
- Queue priority: Queues are always ordered according to their priority; jobs in higher-priority queues are examined before those in lower-priority queues. See [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#).
- Sorting queues: PBS always sorts queues into priority order. See [section 4.9.47, “Sorting Queues into Priority Order”, on page 222](#).
- Express queues: Jobs in express queues are assigned increased priority. See [section 2.3.5.3.i, “Express Queues”, on page 27](#), and [section 4.3.5.4.ii, “Using Express Queues in Job Priority Calculation”, on page 69](#).
- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See [section 4.9.39, “Routing Jobs”, on page 205](#).

4.3.5.14.ii Job Sorting Tools

You can use multiple job sorting tools, one at a time in succession. You can use different sorting tools for primetime and non-primetime.

- Job sorting formula: You create a formula that PBS uses to calculate each job's priority. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).
- Fairshare: PBS tracks past usage of specified resources, and starts jobs based on specified usage ratios. See [section 4.9.19, “Using Fairshare”, on page 139](#).
- Sorting jobs on keys: PBS can sort jobs according to one or more keys, such as requested CPUs or memory; see [section 4.9.45, “Sorting Jobs on a Key”, on page 220](#).
- Entity shares (strict priority): Jobs are prioritized according to the owner's fairshare allocation. See [section 4.9.14, “Sorting Jobs by Entity Shares \(Was Strict Priority\)”, on page 133](#).
- FIFO: Jobs can be run in submission order. See [section 4.9.20, “FIFO Scheduling”, on page 150](#).
- Job's requested priority: you can sort jobs on the priority requested for the job; see [section 4.9.46, “Sorting Jobs by Requested Priority”, on page 222](#).

4.3.5.14.iii Other Job Prioritization Tools

- Strict ordering: you can specify that jobs must be run in priority order, so that a job that cannot run because resources are unavailable is not skipped. See [section 4.9.48, “Using Strict Ordering”, on page 223](#).
- Waiting time: PBS can assign increased priority to jobs that have been waiting to run. See [section 4.9.13, “Eligible Wait Time for Jobs”, on page 129](#).
- Setting job execution priority: PBS can set job execution priority according to a set of rules. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- Preemption: PBS preempts lower-priority jobs in order to run higher-priority jobs. See [section 4.9.33, “Using Preemption”, on page 180](#).
- Preventing preemption: You can prevent certain jobs from being preempted. See [section 4.3.5.9, “Preventing Jobs from Being Preempted”, on page 71](#).
- Making preempted jobs top jobs: PBS can backfill around preempted jobs. See [section 4.9.3.5, “Backfilling Around Preempted Jobs”, on page 110](#).
- Behavior overrides: you can intervene manually in how jobs are run. See [section 4.9.30, “Overrides”, on page 162](#).

4.3.6 Resource Allocation to Users, Projects & Groups

If you need to ensure fairness, you may need to make sure that resources are allocated fairly. If different users, groups, or projects own or pay for different amounts of hardware or machine time, you may need to allocate resources according to these amounts or proportions.

You can allocate hardware-based resources such as CPUs or memory, and/or time-based resources such as **walltime** or CPU time, according to the agreed amounts or proportions. You can also control who starts jobs.

4.3.6.1 Limiting Amount of Resources Used

4.3.6.1.i Allocation Using Resource Limits

You can use resource limits as a way to enforce agreed allocation amounts. This is probably the most straightforward way, and the easiest to explain to your users. PBS provides a system for limiting the total amount of each resource used by projects, users, and groups at the server and at each queue. For example, you can set a limit on the number of CPUs that any generic user can use at one time at QueueA, but set three different individual limits for each of three users that have special requirements, at the same queue. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

4.3.6.1.ii Allocation Using Fairshare

The PBS fairshare tool allows you to start jobs according to a formula based on resource usage by job owners. You can designate who the valid job owners are, which resources are being tracked, and how much of the resources each owner is allowed to be using. Fairshare uses a moving average of resource usage, so that a user who in the recent past has not used their share can use more now. For example, you can track usage of the `cpur` resource, and give one group 40 percent of usage, one 50 percent, and one group, 10 percent. See [section 4.9.19, “Using Fairshare”, on page 139](#).

4.3.6.1.iii Allocation Using Routing

If you do not want to place usage limits directly on projects, users, or groups, you can instead route their jobs to specific queues, where those queues have their own resource usage limits.

To route jobs this way, force users to submit jobs to a routing queue, and set access control limits at each execution queue. See [section 11.3, “Using Access Control Lists”, on page 492](#). Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

Using this method, you place a limit for total resource usage at each queue, for each resource you care about. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

You can also route jobs to specific queues, where those queues can send jobs only to specific vnodes. See [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).

4.3.6.2 Limiting Jobs

4.3.6.2.i Limiting Number of Jobs per Project, User, or Group

You can set limits on the numbers of jobs that can be run by projects, users, and groups. You can set these limits for each project, user, and group, and you can set them at the server and at each queue. You can set a generic limit for all projects, users, or groups, and individual limits that override the generic limit. For example, you can set a limit that says that no user at its partition can run more than 8 jobs. Then you can set a more specific limit for QueueA, so that users at QueueA can run 4 jobs. Then you can set a limit for User1 and User2 at QueueA, so that they can run 6 jobs. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

4.3.6.2.ii Allocation Using Round-robin Queue Selection

PBS can select jobs from queues by examining groups of queues in round-robin fashion, where all queues in each group have the same priority. When using the round-robin method, a scheduler considers the first queue in a group, tries to run the top job from that queue, then considers the next queue, tries to run the top job from that queue, then considers the next queue, and so on, in a circular fashion. A scheduler runs all the jobs it can from the highest-priority group first, then moves to the group with the next highest priority.

If you want a simple way to control how jobs are started, you can use round-robin where each queue in a group belongs to a different user or entity. See [section 4.9.38, “Round Robin Queue Selection”, on page 204](#).

4.3.6.2.iii Limiting Resource Usage per Job

If you are having trouble with large jobs taking up too much of a resource, you can limit the amount of the resource being used by individual jobs. You can set these limits at each queue, and at the server. See [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#).

4.3.6.3 Resource Allocation Tools

The following is a list of scheduling tools that you can use for allocating resources or limiting resources or jobs:

- Matching: PBS places jobs where the available resources match the job's resource requirements; see [section 4.9.28, “Matching Jobs to Resources”, on page 159](#).
- Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See [section 4.9.37, “Reservations”, on page 196](#).
- Fairshare: PBS tracks past usage of specified resources, and starts jobs based on specified usage ratios. See [section 4.9.19, “Using Fairshare”, on page 139](#).
- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See [section 2.3.6, “Routing Queues”, on page 27](#) and [section 4.9.39, “Routing Jobs”, on page 205](#).
- Limits on resource usage by projects, users, and groups: You can set limits on user and group resource usage. See [section 4.9.26, “Limits on Project, User, and Group Resource Usage”, on page 159](#).
- Round-robin job selection: PBS can select jobs from queues that have the same priority in a round-robin fashion. See [section 4.9.38, “Round Robin Queue Selection”, on page 204](#).
- Sorting queues: PBS always sorts queues into priority order. See [section 4.9.47, “Sorting Queues into Priority Order”, on page 222](#).
- Limits on number of jobs for projects, users, and groups: You can set limits on the numbers of jobs that can be run by projects, users, and groups. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).
- Limits on resources used by each job: You can set limits on the amount of each resource that any job can use. See [section 4.9.24, “Limits on Per-job Resource Usage”, on page 158](#).
- Using custom resources to limit resource usage: You use custom resources to manage usage. See [section 4.9.8, “Using Custom and Default Resources”, on page 116](#).
- Gating and admission requirements: You can specify admission requirements for jobs. See [section 4.9.22, “Gating Jobs at Server or Queue”, on page 157](#).
- Making jobs inherit default resources: You can use default resources to manage jobs. See [section 4.9.8, “Using Custom and Default Resources”, on page 116](#).

4.3.7 Time Slot Allocation

Time slot allocation is the process of creating time slots within which only specified jobs are allowed to run.

4.3.7.1 Why Allocate Time Slots

You may want to set up blocks of time during which only certain jobs are allowed to run. For example, you might need to ensure that specific high-priority jobs have their own time slot, so that they are guaranteed to be able to run and finish before their results are required.

You may want to divide jobs into those that run at night, when no one is around, and those that run during the day, because their owners need the results then.

You might want to run jobs on desktop clusters only at night, when the primary users of the desktops are away.

When you upgrade PBS, a chunk of dedicated time can come in very handy. You set up dedicated time for a time period that is long enough for you to perform the upgrade, and you make sure the time slot starts far enough out that no jobs will be running.

You may want to run different scheduling policies at different times or on different days.

4.3.7.2 How to Allocate Time Slots

Time slots are controlled by queues: primetime queues, non-primetime queues, dedicated time queues, and reservation queues. For this, you use your favorite routing method to move jobs into the desired queues. See [section 4.9.39, “Routing Jobs”, on page 205](#).

4.3.7.2.i Allocation Using Primetime and Holidays

You can specify how to divide up days or weeks, and designate each time period to be either primetime or non-primetime. You can use this division in the following ways:

- You can run a different policy during primetime from that during non-primetime
- You can run specific jobs during primetime, and others during non-primetime

See [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).

4.3.7.2.ii Allocation Using Dedicated Time

Dedicated time is a time period where the only jobs that are allowed to run are the ones in dedicated time queues. The policy you use during dedicated time is controlled by the normal primetime and non-primetime policies; those times overlap dedicated time.

If you don't allow any jobs into a dedicated time queue, you can use it to perform maintenance, such as an upgrade.

See [section 4.9.10, “Dedicated Time”, on page 128](#).

4.3.7.2.iii Allocation Using Reservations

You and any other PBS user can create advance and standing reservations. These are time periods with a defined start and end, for a specific, defined set of resources. Reservations are used to make sure that specific jobs can run on time. See [section 4.9.37, “Reservations”, on page 196](#).

4.3.7.2.iv Allocation Using `cron` Jobs

You can use `cron` to run jobs at specific times. See [section 4.9.7, “cron Jobs”, on page 115](#).

4.3.7.3 Time Slot Allocation Tools

The following is a list of scheduling tools that you can use to create time slots:

- Primetime and holidays: You can specify days and times that are to be treated as prime execution time. See [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).
- Dedicated time: You can set aside blocks of time reserved for certain system operations. See [section 4.9.30.6, “Using Dedicated Time”, on page 164](#).
- `cron` jobs: You can use `cron` to run jobs. See [section 4.9.30.7, “Using cron Jobs”, on page 164](#).
- Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See [section 4.9.37, “Reservations”, on page 196](#).

4.3.8 Job Placement Optimization

PBS automatically places jobs where they can run, but you can refine how jobs are placed.

Optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of where jobs can be run.

PBS places jobs according to placement optimization settings in tools to specify how vnodes should be organized, how jobs should be distributed, and how resources should be used.

4.3.8.1 Why Optimize Placement

PBS automatically places jobs where they can run, matching jobs to resources, so why optimize placement?

- You can help PBS refine its understanding of hardware topology, so that PBS can place jobs where they will run most efficiently.
- If you have some vnodes that are faster than others, you can preferentially place jobs on those vnodes.
- You may need to place jobs according to machine ownership, so that for example only jobs owned by a specific group run on a particular machine.
- You can take advantage of unused workstation computing capacity.
- You can balance the workload between two or more PBS partitions or complexes, trading jobs around depending on the workload on each partition or complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

4.3.8.2 Matching Jobs to Resources

By default, PBS places jobs where the available resources match the job's resource requirements. See [section 4.9.28, “Matching Jobs to Resources”, on page 159](#).

4.3.8.3 Organizing and Selecting Vnodes

By default, the order in which PBS examines vnodes is undefined. The default setting for vnode sorting is the following:

```
node_sort_key: "sort_priority HIGH all"
```

However, `sort_priority` means sort on each vnode's `priority` attribute, but by default, that attribute is unset.

PBS can organize vnodes into groups. By default, PBS does not organize vnodes into groups.

By default, when PBS chooses vnodes for a job, it runs down its list of vnodes, searching until it finds vnodes that can supply the job with the requested resources. You can improve this in two ways:

- PBS provides a way to organize your vnodes so that jobs can run on groups of vnodes, where the selected group of vnodes provides the job with good connectivity. This can improve memory access and interprocess communication timing. PBS then searches through these groups of vnodes, called *placement sets*, looking for the smallest group that satisfies the job's requirements. Each placement set is a group of vnodes that share a value for a resource. An illustrative example is a group of vnodes that are all connected to the same high speed switch, so that all of the vnodes have the same value for the switch resource. For detailed information on how placement sets work and how to configure them, see [section 4.9.32, “Placement Sets”, on page 168](#).
- By default, the order in which PBS examines vnodes, whether in or outside of placement sets, is undefined. PBS can sort vnodes on one or more keys. Using this tool, you can specify which vnodes should be selected first. For information on sorting vnodes on keys, see [section 4.9.49, “Sorting Vnodes on a Key”, on page 224](#).

You can sort vnodes in conjunction with placement sets.

4.3.8.4 Distributing Jobs

All of the following methods for distributing jobs can be used together.

4.3.8.4.i Filtering Jobs to Specific Vnodes

If you want to run certain kinds of jobs on specific vnodes, you can route those jobs to specific execution queues, and tie those queues to the vnodes you want. For example, if you want to route jobs requesting large amounts of memory to your large-memory machines, you can set up an execution queue called LMemQ, and associate that queue with the large-memory vnodes. You can route any kind of job to its own special execution queue. For example, you can route jobs owned by the group that owns a cluster to a special queue which is associated with the cluster. For details on routing jobs, see [section 4.9.39, “Routing Jobs”, on page 205](#). For details on associating vnodes and queues, see [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).

4.3.8.4.ii Running Jobs at Least-loaded Partition or Complex

You can set up cooperating PBS partitions and complexes that automatically run jobs from each other's queues. This allows you to dynamically balance the workload across multiple, separate PBS partitions and complexes. See [section 4.9.31, “Peer Scheduling”, on page 164](#).

4.3.8.4.iii Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won't interfere with the user's operation. See [section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 117](#).

4.3.8.4.iv Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See [section 4.9.27, “Using Load Balancing”, on page 159](#).

4.3.8.4.v Placing Job Chunks on Desired Hosts

You can tell PBS to place each job on as few hosts as possible, to place each chunk of a job on a separate host, a separate vnode, or on any vnode. You can specify this behavior for the jobs at a queue and at the server.

You can do the following

- Set default placement behavior for the queue or server: jobs inherit placement if they do not request it; see [section 5.9.3.5, “Specifying Default Job Placement”, on page 243](#)
- Use a hook to set each job's placement request (`Resource_List.place`). See the PBS Professional Hooks Guide

For more on placing chunks, see [section 4.9.6, “Organizing Job Chunks”, on page 115](#).

For information on how jobs request placement, see [section 2.57.2.6, “Requesting Resources and Placing Jobs”, on page 219](#).

4.3.8.5 Shared or Exclusive Resources and Vnodes

PBS can give jobs their own vnodes, or fill vnodes with as many jobs as possible. A scheduler uses a set of rules to determine whether a job can share resources or a host with another job. These rules specify how the vnode `sharing` attribute should be combined with a job's placement directive. The vnode's `sharing` attribute supersedes the job's placement request.

You can set each vnode's `sharing` attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job's placement request. See [section 4.9.41, “Shared vs. Exclusive Use of Resources by Jobs”, on page 210](#).

4.3.8.6 Tools for Organizing Vnodes

- Placement sets: PBS creates sets of vnodes organized by the values of multiple resources. See [section 4.9.32, “Placement Sets”, on page 168](#).
- Sorting vnodes on keys: PBS can sort vnodes according to specified keys. See [section 4.9.49, “Sorting Vnodes on a Key”, on page 224](#).

4.3.8.7 Tools for Distributing Jobs

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See [section 2.3.6, “Routing Queues”, on page 27](#) and [section 4.9.39, “Routing Jobs”, on page 205](#).
- Associating vnodes with queues: You can specify that jobs in a given queue can run only on specific vnodes, and vice versa. See [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See [section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 117](#).
- Peer scheduling: PBS partitions and complexes can exchange jobs. See [section 4.9.31, “Peer Scheduling”, on page 164](#).
- Load balancing: PBS can place jobs so that machines have balanced loads. See [section 4.9.27, “Using Load Balancing”, on page 159](#).
- SMP cluster distribution (**deprecated**): PBS can place jobs in a cluster as you specify. See [section 4.9.43, “SMP Cluster Distribution”, on page 217](#).

4.3.9 Resource Efficiency Optimizations

PBS automatically runs each job where the resources required for the job are available. You can refine the choices PBS makes.

Resource optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of how resources are used.

Before reading this section, please make sure you understand how resources are used by reading [section 4.9.28, “Matching Jobs to Resources”, on page 159](#).

4.3.9.1 Why Optimize Use of Resources

You may want to take advantage of the following:

- If you are using strict ordering, you can prevent resources from standing idle while the top job waits for its resources to become available
- PBS can estimate the start times of jobs, so that users can stay informed
- PBS can provision vnodes with the environments that jobs require
- PBS can track resources that are outside of the control of PBS, such as scratch space
- You can take advantage of unused workstation computing capacity
- You can balance the workload between two or more PBS partitions or complexes, trading jobs around depending on the workload on each partition or complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- Users can specify that jobs that are dependent on the output of other jobs run only after the other jobs complete
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

4.3.9.2 How to Optimize Resource Use

4.3.9.2.i Backfilling Around Top Jobs

PBS creates a list of jobs ordered by priority, and tries to run the jobs in order of priority. You can force all jobs to be run in exact order of their priority, using strict ordering. See [section 4.9.48, “Using Strict Ordering”, on page 223](#). However, this can reduce resource utilization when the top job cannot run now and must wait for resources to become available, idling the entire partition or complex. You can offset this problem by using backfilling, where PBS tries to fit smaller jobs in around the top job that cannot run. The start time of the top job is not delayed. Job walltimes are required in order to use backfilling. You can specify the number of jobs around which to backfill. You can also disable this feature. See [section 4.9.3, “Using Backfilling”, on page 109](#).

PBS can shrink the walltime of shrink-to-fit jobs into available time slots. These jobs can be used to backfill around top jobs and time boundaries such as dedicated time or reservations. See [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

If you do not use strict ordering, PBS won't necessarily run jobs in exact priority order. PBS will instead run jobs so that utilization is maximized, while trying to preserve priority order.

4.3.9.2.ii Using Dependencies

Job submitters can specify dependencies between jobs. For example, if you have a data analysis job that must run after data collection and cleanup jobs, you can specify that. See [section 4.9.11, “Dependencies”, on page 129](#).

4.3.9.2.iii Estimating Start Time for Jobs

You can tell PBS to estimate start times and execution vnodes for either the number of jobs being backfilled around, or all jobs. Users can then see when their jobs are estimated to start, and the vnodes on which they are predicted to run. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).

4.3.9.2.iv Provisioning Vnodes with Required Environments

PBS can provision vnodes with environments (applications or operating systems) that jobs require. This means that a job can request a particular environment that is not yet on a vnode, but is available to be instantiated there. See [section 4.9.35, “Provisioning”, on page 195](#).

4.3.9.2.v Tracking Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for application licenses and scratch space. You can represent elements that are available to the entire partition or PBS complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see [section 5.14.3.1.iii, “Example of Configuring Dynamic Server-level Resource”, on page 264](#). For an example of configuring a dynamic host-level resource, see [section 5.14.4.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 265](#).

For a complete description of how to create and use dynamic resources, see [section 5.14, “Custom Resources”, on page 252](#).

4.3.9.3 Optimizing Resource Use by Job Placement

4.3.9.3.i Sending Jobs to Partition or Complex Having Lightest Workload

You can set up cooperating PBS partitions or complexes that automatically run jobs from each other's queues. This allows you to dynamically balance the workload across multiple, separate partitions or complexes. See [section 4.9.31, “Peer Scheduling”, on page 164](#).

4.3.9.3.ii Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won't interfere with the user's operation. See [section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 117](#).

4.3.9.3.iii Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See [section 4.9.27, “Using Load Balancing”, on page 159](#).

4.3.9.4 Resource Efficiency Optimization Tools

The following is a list of scheduling tools that you can use to optimize how resources are used:

- Backfilling around most important job(s): PBS can place small jobs in otherwise-unused blocks of resources. See [section 4.9.3, “Using Backfilling”, on page 109](#).
- Dependencies: Users can specify requirements that must be met by previous jobs in order for a given job to run. See [section 4.9.11, “Dependencies”, on page 129](#).
- Estimating start time of jobs: PBS can estimate when jobs will start, so that users can be informed. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).
- Provisioning vnodes with required environments: PBS can provision vnodes with the environments that jobs require. See [section 4.9.35, “Provisioning”, on page 195](#).
- Using dynamic resources: PBS can track resources such as scratch space and licenses. See [section 4.9.12, “Dynamic Resources”, on page 129](#).
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See [section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 117](#).
- Peer scheduling: PBS partitions and complexes can exchange jobs. See [section 4.9.31, “Peer Scheduling”, on page 164](#).
- Load balancing: PBS can place jobs so that machines have balanced loads. See [section 4.9.27, “Using Load Balancing”, on page 159](#).

4.3.10 Overrides

Overrides are the techniques you use to override the specified scheduling behavior of PBS.

4.3.10.1 Why and How to Override Scheduling

- If you need to run a job immediately, you can tell PBS to run a job now. You can optionally specify the vnodes and resources to run it. See [section 4.9.30.1, “Run a Job Manually”, on page 162](#).
- If you need to prevent a job from running, you can tell PBS to place a hold on a job. See [section 4.9.30.2, “Hold a Job Manually”, on page 163](#).
- If you need to change how the formula computes job priority, you can make on-the-fly changes to how the formula is computed. See [section 4.9.30.5, “Change Formula On the Fly”, on page 164](#).
- If you need a block of time where you can control what's running, for example for upgrading PBS, you can create dedicated time. See [section 4.9.30.6, “Using Dedicated Time”, on page 164](#).
- If you need to submit jobs at a certain time, you can use `cron` to run jobs. See [section 4.9.30.7, “Using cron Jobs”, on page 164](#).
- If you need to change job resource requests, programs, environment, or attributes, you can use hooks to examine jobs and alter their characteristics. See the *PBS Professional Hooks Guide*.
- If you need to prevent a scheduler from calendaring jobs, you can set their `topjob_ineligible` attribute to *True*. See [section 4.9.17, “Calendaring Jobs”, on page 138](#).

4.3.11 Configurable Job Dispatch Rate

4.3.11.1 The `job_run_wait` Scheduler Attribute

You can use the `job_run_wait` scheduler attribute to instruct the scheduler to dispatch jobs without waiting, after waiting only for any `runjob` hooks to execute, or after waiting for any `execjob_*` hooks to execute. Allowed values for `job_run_wait`:

none: run next job immediately

runjob_hook: run next job after waiting for any `runjob` hooks to execute

execjob_hook: run next job after waiting for any `execjob_*` hooks to execute

4.3.11.2 Recommendations for Configurable Job Dispatch Rate

The `job_run_wait` scheduler attribute replaces the `throughput_mode` scheduler attribute. We recommend using `job_run_wait`, and we recommend not setting or using `throughput_mode`.

If you set `job_run_wait` to *none* or *runjob_hook*, PBS changes `throughput_mode` to *True*.

If you set `job_run_wait` to *execjob_hook*, PBS changes `throughput_mode` to *False*.

If you set `throughput_mode` to *True*, PBS changes `job_run_wait` to *runjob_hook*.

If you set `throughput_mode` to *False*, PBS changes `job_run_wait` to *execjob_hook*.

4.4 Choosing a Policy

4.4.1 Overview of Kinds of Policies

You can tune PBS to produce any of a wide selection in scheduling behaviors. You can choose from a wide variety of behaviors for each sub-goal, resulting in many possible scheduling policies. However, policies can be grouped into the following kinds:

- FIFO, where you essentially run jobs in the order in which they were submitted; see [section 4.4.2, “FIFO: Submission Order”, on page 83](#)
- According to user or group priority, where the job's priority is determined by the owner's priority; see [section 4.4.3, “Prioritizing Jobs by User, Project or Group”, on page 84](#)
- According to resource allocation rules, where jobs are run so that they use resources following a set of rules for how resources should be awarded to users or groups; see [section 4.4.4, “Allocating Resources by User, Project or Group”, on page 84](#)
- According to the size of the job, for example measured by CPU or memory request; see [section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 86](#)
- By setting up time slots for specific uses; see [section 4.4.6, “Scheduling Jobs into Time Slots”, on page 88](#)

4.4.2 FIFO: Submission Order

If you want jobs to run in the order in which they are submitted, use FIFO. You can use FIFO across the entire partition or complex, or within each queue.

If it's important that jobs run exactly in submission order, use FIFO with strict ordering. However, if you don't want resources to be idle while a top job is stuck, you can use FIFO with strict ordering and backfilling.

To run jobs in submission order, see [section 4.9.20.1, “Configuring Basic FIFO Scheduling”, on page 150](#).

To run jobs in submission order across the entire partition or complex, see [section 4.9.20.2, “FIFO for Entire Partition Or Complex”, on page 150](#).

To run jobs in submission order, examining queues in order of queue priority, see [section 4.9.20.3, “Queue by Queue FIFO”, on page 151](#).

To run jobs in submission order, with strict ordering, see [section 4.9.20.4, “FIFO with Strict Ordering”, on page 151](#).

To run jobs in submission order, with strict ordering and backfilling, see [section 4.9.20.5, “FIFO with Strict Ordering and Backfilling”, on page 151](#).

4.4.3 Prioritizing Jobs by User, Project or Group

If you need to run jobs from some users, groups, or projects before others, you can prioritize jobs using the following techniques:

- Routing each entity's jobs to its own execution queue, assigning the queue the desired priority, and examining jobs queue by queue. See the following:
 - For routing: [section 2.3.6, “Routing Queues”, on page 27](#)
 - For setting queue priority: [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#)
 - For examining jobs queue by queue: [section 4.9.4, “Examining Jobs Queue by Queue”, on page 113](#)
- Routing each entity's jobs to its own execution queue, where the jobs inherit a custom resource that you use in the job sorting formula. See the following:
 - For routing: [section 2.3.6, “Routing Queues”, on page 27](#)
 - For inherited resources: [section 10.3, “Allocating Resources to Jobs”, on page 455](#)
 - For the job sorting formula: [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#)
- Using a hook to allocate a custom resource to each job, where the hook sets the value according to the priority of the job's owner, group, or project, then using the resource in the job sorting formula. See the following:
 - For hooks: the PBS Professional Hooks Guide
 - For custom resources: [section 5.14, “Custom Resources”, on page 252](#)
 - For the job sorting formula: [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#)
- Assigning a greater fairshare allocation in the fairshare tree to the users or groups whose jobs must run first, and running jobs according to entity shares. See the following:
 - For fairshare: [section 4.9.19, “Using Fairshare”, on page 139](#)
 - For entity shares: [section 4.9.14, “Sorting Jobs by Entity Shares \(Was Strict Priority\)”, on page 133](#)

4.4.4 Allocating Resources by User, Project or Group

When you want to divide up hardware usage among users, groups, or projects, you can make sure you allocate resources along those lines. You can do this in the following ways:

- Allocate portions of the entire partition or complex to each entity; see [section 4.4.4.1, “Allocating Portions of Partition Or Complex”, on page 84](#)
- Allocate portions of all machines or clusters to each entity, or use controlled allocation for some hardware, with a free-for-all elsewhere; see [section 4.4.4.2, “Allocating Portions of Machines or Clusters”, on page 85](#)
- Lock entities into using specific hardware; see [section 4.4.4.3, “Locking Entities into Specific Hardware”, on page 86](#)

4.4.4.1 Allocating Portions of Partition Or Complex

4.4.4.1.i Allocating Specific Amounts

To allocate specific amounts of resources across the entire partition or complex, you can use resource limits at the server. These limits set the maximum amount that can be used, ensuring that projects, users, or groups stay within their bounds. You can set a limit for each resource, and make it different for each project, user, and group. You can set a different limit for each project, user, and group, for each resource.

For example, you can set a limit of 48 CPUs in use at once by most groups, but give groupA a limit of 96 CPUs. You can give each individual user a limit of 8 CPUs, but give UserA a limit of 10 CPUs, and UserB a limit of 4 CPUs.

To set limits for usage across the entire partition or complex, set the limits at the server.

See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

4.4.4.1.ii Allocating Percentages

To allocate a percentage of the resources being used in the partition managed by a scheduler, you can use fairshare. Fairshare tracks a moving average of resource usage, so it takes past use into account. You choose which resources to track. You can tune the influence of past usage.

To use fairshare across the entire partition or complex, make sure that both `by_queue` and `round_robin` are *False*.

Fairshare is described in [section 4.9.19, “Using Fairshare”, on page 139](#).

4.4.4.2 Allocating Portions of Machines or Clusters

You can allocate fixed amounts of a machine or groups of machines. You can do this for as many machines as you want. For example, on HostA, you can give GroupA 100 CPUs, GroupB 150 CPUs, and GroupC 50 CPUs, while at HostB, GroupA gets 10, GroupB gets 8, and GroupC gets 25.

To allocate fixed portions of a specific machine or group of machines, you use these tools in combination:

- Create an execution queue for this machine; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Route jobs belonging to the users or groups who share this machine into a queue. Each machine or cluster that requires controls gets its own queue. See [section 4.9.39, “Routing Jobs”, on page 205](#).
- Associate the queue with the vnodes in question; see [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).
- Set a limit at the queue for each resource that you care about, for each project, user, or group. These limits control use of the vnodes associated with the queue only. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

You can prevent unauthorized usage by setting generic project, user, and group limits for the machine's queue to zero. However, you probably don't want users to submit their jobs to a queue where they are not allowed to run, only to have those jobs languish. You can avoid this by doing the following:

- Setting up a routing queue; see [section 2.3.6, “Routing Queues”, on page 27](#).
- Making the routing queue be the default queue:
`Qmgr: set server default_queue = <routing queue name>`
- Making the routing queue the only queue that accepts job submission: set `from_route_only` to *True* on execution queues tied to hardware. See [section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 25](#).
- Using queue access control to limit which jobs are routed into the execution queue; see [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).

You can either set up allocations for every machine, or you can set up allocations for only some machines, leaving a free-for-all for the others. If you want access to be unrestricted for some machines, do not set limits at the server.

4.4.4.3 Locking Entities into Specific Hardware

You can send all jobs from some projects, users, or groups to designated hardware, essentially limiting them to a sandbox. To do this, do the following:

- Create an execution queue for the sandbox hardware; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Create at least one other execution queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Create a routing queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Make the routing queue be the default queue:
`Qmgr: set server default_queue = <routing queue name>`
- Force all users to submit jobs to the routing queue: set `from_route_only` to `True` on all other queues. See [section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 25](#).
- Use queue access control to route according to user or group: route jobs from the controlled users or groups into the sandbox queue only. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).
- Use a job submission hook to route according to project: route the jobs from the desired project(s) to the sandbox queue. See the PBS Professional Hooks Guide.
- Associate the sandbox queue with the sandbox vnodes. See [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).

Note that you can either allow all projects, users, or groups into the sandbox queue, or allow only the controlled projects, users, or groups into the sandbox queue.

4.4.5 Scheduling Jobs According to Size Etc.

You may need to treat jobs differently depending on their size or other characteristics. For example, you might want to run jobs differently depending on the number of CPUs or amount of memory requested by the job, or whether the job requests GPUs.

- Give special priority to a group of jobs
- Run a group of jobs on designated hardware
- Run a group of jobs in designated time slots: reservations, dedicated time, and primetime or non-primetime

There are two main approaches to doing this. You can route jobs into queues, or you can use hooks to set values. Here is an outline:

- Route certain kinds of jobs into their own queues, in order to treat each kind differently. This works for priority, hardware, and time slots. See [section 4.4.5.1, “Special Treatment via Routing”, on page 86](#)
 - Route each kind to its own queue, using queue-based routing or a submission hook;
 - Use queue-based methods to set job priority or to run the jobs on certain hardware or in certain time slots
- Use hooks to set priority for jobs or to set a custom resource that will send jobs to certain hardware. This does not work for time slots. See [section 4.4.5.2, “Special Treatment via Hooks”, on page 88](#).
 - Use a submission hook to set each job's `Priority` attribute, or set a value for a custom resource used in the job sorting formula
 - Use a submission hook to set a custom host-level resource value for each job, where the value matches the value at the desired hardware

4.4.5.1 Special Treatment via Routing

Use a routing queue or a hook to route jobs into a special queue, where the jobs are given special priority, or are run on special hardware, or are run in special time slots.

4.4.5.1.i Routing via Queues

- Create your destination queues. See [section 2.3.3, “Creating Queues”, on page 25](#).
- Set limits at the destination queues, so that each queue receives the correct jobs. See [section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 28](#).
- Create a routing queue, and set its destination queues. See [section 2.3.6, “Routing Queues”, on page 27](#).
- Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

4.4.5.1.ii Using Hooks to Route Jobs

You can use a submission hook to move jobs into the queues you want. See [section 4.9.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 207](#).

4.4.5.1.iii Giving Routed Jobs Special Priority

You can give routed jobs special priority in the following ways:

- Have the jobs inherit a custom resource from the special queue, and use this resource in the job sorting formula.
 - For how to have jobs inherit custom resources, see [section 10.3, “Allocating Resources to Jobs”, on page 455](#).
 - For how to use the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).
- Give the queue itself special priority, and use queue priority in the job sorting formula.
 - For how to assign priority to queues, see [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#)
 - For how to use the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

4.4.5.1.iv Running Jobs on Special Vnodes

Now that the special jobs are routed to a special queue, associate that queue with the special vnodes. See [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).

4.4.5.1.v Running Jobs in Special Time Slots

If you want to run jobs during dedicated time, route the jobs into one or more dedicated time queues. In the same way, for primetime or non-primetime, route jobs into primetime or non-primetime queues. You can also route jobs into reservation queues for reservations that you have created for this purpose.

For using dedicated time, see [section 4.9.10, “Dedicated Time”, on page 128](#)

For using primetime and non-primetime, see [section 4.9.34, “Using Primetime and Holidays”, on page 190](#)

For using reservations, see [section 4.9.37, “Reservations”, on page 196](#)

4.4.5.2 Special Treatment via Hooks

4.4.5.2.i Setting Job Priority Via Hook

You can set a job's Priority attribute using a hook. Note that users can `qalter` the job's Priority attribute. Use a job submission hook to set the job's priority, by doing one of the following:

- Set a custom numeric resource for the job, and use the resource in the job sorting formula
 - For how to use hooks, see the PBS Professional Hooks Guide
 - For how to use the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).
- Set the job's Priority attribute, and sort jobs on a key, where the key is the job's Priority attribute.
 - For how to set job attributes, see the PBS Professional Hooks Guide
 - For how to sort jobs on a key, see [section 4.9.45, “Sorting Jobs on a Key”, on page 220](#)

4.4.5.2.ii Routing Jobs to Hardware via Hooks

You can send jobs to particular hardware without using a particular queue, by using a hook. See [section 4.9.39.4.i, “Using Hooks to Tag Jobs”, on page 208](#).

4.4.6 Scheduling Jobs into Time Slots

You can schedule jobs in time slots in the following ways:

- Set aside time slots for specific entities; see [section 4.4.6.1, “Setting Aside Time Slots for Entities”, on page 88](#)
- Lock entities into specific time slots; see [section 4.4.6.2, “Locking Entities into Time Slots”, on page 89](#)

4.4.6.1 Setting Aside Time Slots for Entities

You can set aside time slots that are reserved exclusively for certain users or groups. You can use reservations, dedicated time, primetime, or non-primetime.

4.4.6.1.i Reservations

Reservations set aside one or more blocks of time on the requested resources. Users can create their own reservations, or you can create them and set their access control to allow only specified users to submit jobs to them. See [section 4.9.37, “Reservations”, on page 196](#).

4.4.6.1.ii Dedicated Time

During dedicated time, the only jobs allowed to run are those in dedicated queues. The drawback to dedicated time is that it applies to the entire partition or complex. If you want to set aside one or more dedicated time slots for a user or group, do the following:

- Create a dedicated queue. See [section 2.3.5.2.i, “Dedicated Time Queues”, on page 26](#).
- Define dedicated time. See [section 4.9.10, “Dedicated Time”, on page 128](#).
- Set access control on the dedicated queue so that only the particular users or groups you want can submit jobs to the queue. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).
- If you want to limit access on a dedicated queue to a specific project, set the generic limit for queued jobs for projects at that queue to zero, and then set the individual limit for the specific project higher.

4.4.6.1.iii Non-primetime

You can set up primetime and non-primetime so that one of them, for example, non-primetime, is used as a special time slot allocated to particular users or groups. The advantage of using non-primetime is that you can set up a separate scheduling policy for it, for example, using fairshare during non-primetime and sorting jobs on a key during primetime. Note that the formula, if defined, is in force all of the time. To use non-primetime, do the following:

- Create a non-primetime queue; see [section 2.3.3, “Creating Queues”, on page 25](#) and [section 2.3.5.2.ii, “Primetime and Non-Primetime Queues”, on page 26](#).
- Define primetime and non-primetime; see [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).
- Set access control on the non-primetime queue so that only the particular users or groups you want can submit jobs to the queue. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).
- Make sure that the scheduling policy you want is in force during non-primetime. See [section 4.9.34.1, “How Primetime and Holidays Work”, on page 190](#).

4.4.6.2 Locking Entities into Time Slots

You can make all jobs from some users or groups run during designated time slots. You can run them during a reservation, dedicated time, or non-primetime.

4.4.6.2.i Locking Entities into Reservations

To allow a user to submit jobs only into a reservation, do the following:

- Create a reservation for the resources and time(s) you want the controlled user(s) to use. When creating the reservation, set access control to allow the controlled user(s). See [section 4.9.37, “Reservations”, on page 196](#) and [section 11.3.8.3, “Setting and Changing Reservation Access”, on page 502](#).
- Set access control on all queues except the reservation's queue to deny the controlled user(s); see [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).

4.4.6.2.ii Locking Entities into Dedicated Time

You can create a dedicated time queue, and send all jobs from controlled projects, users, or groups to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more projects, users, or groups into one or more dedicated time slots, do the following:

- Create a dedicated time queue; see [section 2.3.3, “Creating Queues”, on page 25](#) and [section 2.3.5.2.i, “Dedicated Time Queues”, on page 26](#).
- Create at least one other execution queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Create a routing queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Prevent controlled users from submitting to non-dedicated time execution queues: set `from_route_only` to `True` on the non-dedicated time execution queues. See [section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 25](#).
- Use queue access control to allow jobs from the controlled users or groups into the dedicated time queue only. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).
- Use a job submission hook to route jobs from controlled projects into the dedicated time queue. See the PBS Professional Hooks Guide
- Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

Note that you can either allow all users into the dedicated time queue, or allow only the controlled users into the dedicated time queue.

4.4.6.2.iii Locking Entities into Non-primetime

You can create a non-primetime queue, and send all jobs from controlled users, groups, or projects to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more users, groups, or projects into one or more non-primetime slots, do the following:

- Create a non-primetime queue; see [section 2.3.3, “Creating Queues”, on page 25](#) and [section 2.3.5.2.ii, “Primetime and Non-Primetime Queues”, on page 26](#).
- Create at least one other execution queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Create a routing queue; see [section 2.3.3, “Creating Queues”, on page 25](#).
- Prevent controlled users from submitting to primetime execution queues: set `from_route_only` to *True* on the primetime execution queues. See [section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 25](#).
- Make the routing queue be the default queue:
Qmgr: `set server default_queue = <routing queue name>`
- Use queue access control to allow jobs from the controlled users or groups into the non-primetime queue only. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).
- Use a job submission hook to route jobs from controlled projects into the non-primetime queue. See the PBS Professional Hooks Guide
- Define primetime and non-primetime; see [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).
- Make sure that the scheduling policy you want is in force during non-primetime. See [section 4.9.34.1, “How Primetime and Holidays Work”, on page 190](#).

Note that you can either allow all users into the non-primetime queue, or allow only the controlled users into the non-primetime queue.

4.4.7 Default Scheduling Policy

The default scheduling policy is determined by the default settings for all of the attributes, parameters, etc. that determine a scheduler's behavior. For a list of all of these elements, see [section 4.5.1, “Configuring a Scheduler”, on page 93](#).

The default behavior of a scheduler is the following:

- A scheduler matches jobs with available resources. This means that a scheduler places each job only where that job has enough resources to run. See [section 4.9.28, “Matching Jobs to Resources”, on page 159](#).
- A scheduler will not over-allocate the resources that are listed in the scheduler's `resources` parameter. The defaults for these are `ncpus`, `mem`, `arch`, `host`, `vnode`, `aoe`. See [section 4.9.28.1, “Scheduling on Consumable Resources”, on page 159](#).

- A scheduler sorts vnodes according to its `node_sort_key` parameter, whose default setting is the following:
`node_sort_key: "sort_priority HIGH all"`

This means that vnodes are sorted by the value of their `priority` attribute, with high-priority vnodes used first. A scheduler places jobs first on vnodes that are first in the sorted list.

Note that all vnodes have the same default priority upon creation, so the default sorted order for vnodes is undefined.

See [section 4.9.49, “Sorting Vnodes on a Key”, on page 224](#).

- Queues are sorted according to the value of their `priority` attribute, so that queues with a higher priority are considered before those with a lower priority. See [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#).
- Jobs are considered according to the priority of their queues. A scheduler runs all of the jobs that it can from the highest-priority queue before moving to the next queue, and so on. See [section 4.9.4, “Examining Jobs Queue by Queue”, on page 113](#).
- Within each queue, jobs are considered in submission order.
- Jobs in an express queue are placed in the `express_queue` preemption priority level. They are also placed in the *Express* execution priority class. The default priority for a queue to be an express queue is 150. See [section 2.3.5.3.i, “Express Queues”, on page 27](#).
- Queued jobs are sorted according to their priority. Special jobs are all prioritized ahead of normal jobs, without regard to the queue in which they reside. The order for job priority for special jobs, highest first, is reservation jobs, jobs in express queues, preempted jobs. After this, a scheduler looks at normal jobs, queue by queue. All jobs in express queues, and all preempted jobs are considered before a scheduler looks at the individual queues.

See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

- A scheduler will preempt lower-priority jobs in order to run higher-priority jobs (`preemptive_sched` is *True* by default). By default, it has two levels of job priority, `express_queue`, and `normal_jobs`, where `express_queue` jobs can preempt `normal_jobs`. This is set in the scheduler's `preempt_prio` attribute.

When a scheduler chooses among jobs of the same priority for a job to preempt, it uses the only setting for `preempt_sort`, which is `min_time_since_start`, choosing jobs that have been running for the shortest time.

When a scheduler chooses how to preempt a job, it uses the default setting for its `preempt_order` attribute, which is *SCR*, meaning that first it will attempt suspension, then checkpointing, then if necessary requeueing.

See [section 4.9.33, “Using Preemption”, on page 180](#).

- A scheduler will do its best to backfill smaller jobs around the job it has decided is the most important job. See [section 4.9.3, “Using Backfilling”, on page 109](#).
- Primetime by default is 24/7. Any holiday is considered non-primetime. You can define primetime and holidays in the file `<sched_priv_directory>/holidays`. These dates should be adjusted yearly to reflect your local holidays. See [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).
- A scheduler runs every 10 minutes unless a new job is submitted or a job finishes execution. See [section 4.5.6, “The Scheduling Cycle”, on page 100](#).
- A scheduler runs with the `job_run_wait` scheduler attribute set to `runjob_hook` by default, so the scheduler runs asynchronously, and doesn't wait for each job to be accepted by MoM, which means it also doesn't wait for an `execjob_begin` hook to finish. Especially for short jobs, this can give better scheduling performance. See [section 4.3.11, “Configurable Job Dispatch Rate”, on page 82](#).

4.4.8 Examples of Workload and Policy

- If you need to have high-priority jobs run soon, and nothing distinguishes the high-priority jobs from the rest:
 - Create advance reservations for the high-priority jobs, and have users submit those jobs to the reservations; see [section 4.9.37, “Reservations”, on page 196](#)
- If you want to run jobs in submission order:
 - FIFO; see [section 4.9.20, “FIFO Scheduling”, on page 150](#)
- If you have low-priority jobs that should run only when other jobs don't need the resources:
 - Set up an anti-express queue; see [section 4.9.1, “Anti-Express Queues”, on page 107](#)
- If you have a mix of jobs, and want to run big jobs first:
 - Sort jobs on a key, using `ncpus` as the key, to run big jobs first; see [section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 86](#)
- If you have a mix of jobs, and want to give big jobs high priority, but avoid having idle resources:
 - Sort jobs on a key, using `ncpus` as the key, to run big jobs first; see [section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 86](#)
 - Use backfilling; see [section 4.9.3, “Using Backfilling”, on page 109](#)
- If you want to have all users start about the same number of jobs:
 - Use round robin, give each user their own queue, and give each queue the same priority; see [section 4.9.38, “Round Robin Queue Selection”, on page 204](#)
- If you want to always give each user access to a certain amount of a resource, but allow more if no one else is using it:
 - Use soft limits for the amount each user can use; see [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#) and [section 4.9.33, “Using Preemption”, on page 180](#)
- If your partition or site has more than one funding source:
 - See [section 4.4.4, “Allocating Resources by User, Project or Group”, on page 84](#)
- If you have lots of users in a partition or complex:
 - Use resource limits; see [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#), or
 - Use fairshare; see [section 4.9.19, “Using Fairshare”, on page 139](#)
- If you have jobs that must run at the end of the day:
 - Use dependencies for end-of-day accounting; see [section 4.9.11, “Dependencies”, on page 129](#)
- If you need to ensure that jobs run in certain hours on desktops:
 - Use cycle harvesting; see [section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 117](#), or
 - Use primetime & non-primetime for nighttime; see [section 4.9.34, “Using Primetime and Holidays”, on page 190](#)
- If you want to be sure a job will run:
 - Create an advance reservation; see [section 4.9.37, “Reservations”, on page 196](#)
- If you have more than one partition or complex, and you want to balance the workload across the partitions or complexes:
 - Use peer scheduling; see [section 4.9.31, “Peer Scheduling”, on page 164](#)
- If you have some jobs that should prefer to run on one set of vnodes, and other jobs that should prefer to run on another set of vnodes, but if the preferred vnodes are busy, a job can run on the non-preferred vnodes:
 - Use peer scheduling. Set up two partitions or complexes, give the pulling queues low priority, and use queue

priority in the job sorting formula. See [section 4.9.31, “Peer Scheduling”, on page 164](#), [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#), and [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#). You can use a routing queue to initially send jobs to the correct partition or complex. See [section 2.3.6, “Routing Queues”, on page 27](#)

- If you have two (or more) sets of vnodes, and jobs should run on one set or the other, but not both. Additionally, jobs should not have to request where they run. For example, one set of vnodes is new, and one is old:
 - Use a routing queue and two execution queues. Associate each execution queue with one set of vnodes. Put the execution queue for the preferred set of vnodes first in the routing list, but put a limit on the number of queued jobs in the execution queues, so that both queues will fill up. Otherwise the routing queue will preferentially fill the first in its routing list. See [section 2.3.6, “Routing Queues”, on page 27](#), and [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#)
- If you need to apportion a single vnode or cluster according to ownership:
 - See [section 4.4.4, “Allocating Resources by User, Project or Group”, on page 84](#)
- If you have more than one high-priority queue, and at least one low-priority queue, and you want all jobs in high-priority queues to be considered as one group, and run in submission order:
 - Use the job sorting formula to sort jobs on queue priority:


```
set server job_sort_formula = queue_priority
```
 - Give all queues whose jobs should be considered together the same priority
 - Set the `by_queue` scheduler attribute to *False*
- If you want to place jobs on the vnodes with the fewest CPUs first, saving bigger vnodes for larger jobs:
 - Sort vnodes so that those with fewer CPUs come first:


```
node_sort_key: "ncpus LOW"
```

4.5 About Schedulers

Each scheduler, `pbs_sched`, implements its own scheduling policy.

4.5.1 Configuring a Scheduler

4.5.1.1 Where a Scheduler Gets Its Information

Each scheduler has its own `sched_priv` directory, where it keeps scheduler-specific files. For a multisched, this is `$PBS_HOME/sched_priv_<scheduler name>`; for the default scheduler, it is always `$PBS_HOME/sched_priv/`.

The behavior of a scheduler is controlled by the information provided by the following sources:

PBS_est

Hook that runs estimator process which calculates estimated start time and vnodes for jobs. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).

<sched_priv directory>/resource_group

Contains the description of the fairshare tree. Created by you. Can be edited. Read on startup and HUP of scheduler.

<sched_priv directory>/usage

Contains the usage database. Do not edit. Instead, use the `pbsfs` command while a scheduler is stopped; see [“pbsfs” on page 32 of the PBS Professional Reference Guide](#).

<sched_priv directory>/sched_config

Contains scheduler configuration options, also called scheduler parameters, e.g. `fairshare_decay_time`, `job_sort_key`. Read on startup and HUP.

Can be edited. Each entry must be a single, unbroken line. Entries must be double-quoted if they contain whitespace.

See [“Scheduler Parameters” on page 251 of the PBS Professional Reference Guide](#).

<sched_priv directory>/dedicated_time

Contains definitions of dedicated time. Can be edited. Read on startup and HUP.

<sched_priv directory>/holidays

Where you define primetime, non-primetime, and holidays. Can be edited. Read on startup and HUP.

/etc/pbs.conf

Contains scheduler parameters:

PBS_DAEMON_SERVICE_USER

Sets the username under which scheduler(s) run. Default: root

PBS_SCHED_THREADS

Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable.

Overridden by `pbs_sched -t` option and `PBS_SCHED_THREADS` environment variable.

Default: 1

Options to `pbs_sched` command

Control some scheduler behavior. Set on invocation. See [“pbs_sched” on page 105 of the PBS Professional Reference Guide](#).

Scheduler attributes

Control some scheduler behavior. Can be set using `qmgr`. Read every scheduling cycle. See [“Scheduler Attributes” on page 299 of the PBS Professional Reference Guide](#).

Server attributes

Several server attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the server attributes that affect scheduling, along with a brief description. Read every scheduling cycle.

Some limit attributes are marked as "old". These are incompatible with, and are replaced by, the new limit attributes described in [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”](#), on page 283.

For a complete description of each attribute, see [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
<code>backfill_depth</code>	Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around.
<code>default_queue</code>	Specifies queue for jobs that don't request a queue
<code>eligible_time_enable</code>	Enables accruing wait time for jobs

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
est_start_time_freq	Obsolete. Not used. Interval at which PBS calculates estimated start times and vnodes for all jobs.
job_sort_formula	Formula for computing job priorities.
max_group_res	Old. The maximum amount of the specified resource that any single group may consume in this PBS complex.
max_group_res_soft	Old. The soft limit for the specified resource that any single group may consume in this complex.
max_group_run	Old. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.
max_group_run_soft	Old. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time.
max_queued	The maximum number of jobs allowed to be queued or running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_queued_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs queued or running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run	The maximum number of jobs allowed to be running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_res_soft.<resource name>	Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_soft	Soft limit on the number of jobs allowed to be running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_running	Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.
max_user_res	Old. The maximum amount within this complex that any single user may consume of the specified resource.
max_user_res_soft	Old. The soft limit on the amount of the specified resource that any single user may consume within a complex.
max_user_run	Old. The maximum number of jobs owned by a single user allowed to be running within the partition managed by a scheduler at one time.
max_user_run_soft	Old. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time.

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
node_fail_requeue	Controls whether running jobs are automatically requeued or are deleted when the primary execution host fails. Number of seconds to wait after losing contact with the primary execution host MoM before requeueing or deleting jobs. See “node fail requeue” on page 290 of the PBS Professional Reference Guide .
node_group_enable	Specifies whether node grouping is enabled.
node_group_key	Specifies the resource to use for node grouping.
resources_available	The list of available resources and their values defined on the server.
resources_max	The maximum amount of each resource that can be requested by any single job in this complex, if there is not a <code>resources_max</code> value defined for the queue at which the job is targeted.
scheduler_iteration deprecated	The time between scheduling iterations.
scheduling deprecated	Enables scheduling of jobs.
resources_assigned	The total of each type of resource allocated to jobs running and exiting in this complex, plus the total of each type of resource allocated to any started reservations.

Vnode attributes

Several vnode attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the vnode attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see [“Vnode Attributes” on page 321 of the PBS Professional Reference Guide](#).

Table 4-2: Vnode Attributes Involved in Scheduling

Attribute	Effect
current_aoe	This attribute identifies the AOE currently instantiated on this vnode
no_multinode_jobs	Controls whether jobs which request more than one chunk are allowed to execute on this vnode
partition	The partition to which this vnode is assigned
pcpus	The number of physical CPUs on the vnode
priority	The priority of this vnode compared with other vnodes
provision_enable	Controls whether this vnode can be provisioned
queue deprecated	The queue with which this vnode is associated
resources_assigned	The total amount of each resource allocated to running and exiting jobs and started reservations running on this vnode
resources_available	The list of resources and the amounts available on this vnode

Table 4-2: Vnode Attributes Involved in Scheduling

Attribute	Effect
sharing	Specifies whether more than one job at a time can use the resources of the vnode or the vnode's host.
state	Shows or sets the state of the vnode.

Queue attributes

Several queue attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the queue attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

Table 4-3: Queue Attributes Involved in Scheduling

Attribute	Effect
backfill_depth	Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around.
enabled	Specifies whether this queue accepts new jobs.
from_route_only	Specifies whether this queue accepts jobs only from routing queues.
max_array_size	The maximum number of subjobs that are allowed in an array job.
max_group_res	Old. The maximum amount of the specified resource that any single group may consume in this queue.
max_group_res_soft	Old. The soft limit for the specified resource that any single group may consume in this queue.
max_group_run	Old. The maximum number of jobs owned by the users in one group allowed to be running within this queue at one time.
max_group_run_soft	Old. The maximum number of jobs owned by the users in one group allowed to be running in this queue at one time.
max_queueable	Old. The maximum number of jobs allowed to reside in the queue at any given time.
max_queued	The maximum number of jobs allowed to be queued in or running from the queue. Can be specified for users, groups, or all.
max_queued_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from the queue. Can be specified for users, groups, or all.
max_run	The maximum number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.
max_run_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.

Table 4-3: Queue Attributes Involved in Scheduling

Attribute	Effect
max_run_res_soft.<resource name>	Soft limit on the amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.
max_run_soft	Soft limit on the number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.
max_running	Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.
max_user_res	Old. The maximum amount of the specified resource that the jobs of any single user may consume.
max_user_res_soft	Old. The soft limit on the amount of the specified resource that any single user may consume in this queue.
max_user_run	Old. The maximum number of jobs owned by a single user allowed to be running from the queue at one time.
max_user_run_soft	Old. The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time.
node_group_key	Specifies the resource to use for node grouping.
Priority	The priority of this queue compared to other queues of the same type in this PBS partition or complex.
resources_assigned	The total of each type of resource allocated to jobs running and exiting in this queue
resources_available	The list of available resources and their values defined on the queue.
resources_max	The maximum amount of each resource that can be requested by any single job in this queue.
resources_min	The minimum amount of each resource that can be requested by a single job in this queue.
route_destinations	The list of destinations to which jobs may be routed.
route_held_jobs	Specifies whether jobs in the held state can be routed from this queue.
route_lifetime	The maximum time a job is allowed to reside in a routing queue. If a job cannot be routed in this amount of time, the job is aborted.
route_retry_time	Time delay between routing retries. Typically used when the network between servers is down.
route_waiting_jobs	Specifies whether jobs whose execution_time attribute value is in the future can be routed from this queue.
started	Specifies whether jobs in this queue can be scheduled for execution.
state_count	The number of jobs in each state currently residing in this queue.

List of jobs and server-level resources queried from server

Read every scheduling cycle.

Resources in Resource_List job attribute

Read every scheduling cycle.

List of host-level resources queried from MoMs

Read every scheduling cycle.

4.5.1.2 Reference Copies of Files

PBS is installed with a reference copy of the holidays file in which everything is commented out, in `PBS_EXEC/etc/pbs_holidays`.

4.5.2 Making a Scheduler Read its Configuration

If you change a scheduler's configuration file, the scheduler must re-read it for the changes to take effect. To get a scheduler to re-read its configuration information, without stopping the scheduler, you can HUP the scheduler:

```
kill -HUP <scheduler PID>
```

If you set a scheduler attribute using `qmgr`, the change takes effect immediately and you do not need to HUP the scheduler.

4.5.3 Scheduling on Resources

A scheduler honors all resources listed in the `resources:` line in `<sched_priv directory>/sched_config`. If this line is not present, a scheduler honors all resources, built-in and custom. It is more efficient to list just the resources that you want a scheduler to schedule on.

4.5.4 Specifying Scheduler Username

By default, the PBS daemons run as root. However, you can specify that the scheduler should run as some other user. You can do this either by setting `PBS_DAEMON_SERVICE_USER` in the environment when doing an `rpm` install, or by specifying the username in the `PBS_DAEMON_SERVICE_USER` parameter in `/etc/pbs.conf`. See [section 9.1, “Specifying Scheduler Username”, on page 420](#).

4.5.5 Starting, Stopping, and Restarting a Scheduler

4.5.5.1 When and How to Start a Scheduler

During normal operation, startup of the scheduler is handled automatically. The PBS daemons are started automatically at bootup by the PBS start/stop script. During failover, the secondary server automatically tries to use the primary scheduler, and if it cannot, it starts its own scheduler.

To start the default scheduler by hand:

```
PBS_EXEC/sbin/pbs_sched [options]
```

See [“pbs_sched” on page 105 of the PBS Professional Reference Guide](#).

For how to start a multisched, see [section 4.2.2, “Starting a Multisched”, on page 60](#).

4.5.5.2 When and How to Stop a Scheduler

You must stop a scheduler for the following operations:

- (Recommended) Using the `pbsfs` command; see [“pbsfs” on page 32 of the PBS Professional Reference Guide](#).
- Upgrading PBS Professional; see [“Upgrading” on page 67 in the PBS Professional Installation & Upgrade Guide](#).

A scheduler traps signals during the scheduling cycle. You can kill a scheduler at the end of the cycle, or if necessary, immediately. A scheduler does not write the fairshare usage file when it is killed with -9, but it does write the file when it is killed without -9.

You must be root on the scheduler's host.

To stop a scheduler at the end of a cycle:

```
kill <scheduler PID>
```

To stop a scheduler immediately:

```
kill -9 <scheduler PID>
```

4.5.5.3 When and How to Restart a Scheduler

Under most circumstances, when you restart a scheduler, you do not need to specify any options to the `pbs_sched` command. See [“pbs_sched” on page 105 of the PBS Professional Reference Guide](#). Start a scheduler this way:

```
PBS_EXEC/sbin/pbs_sched [options]
```

4.5.6 The Scheduling Cycle

A scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in each scheduler's `scheduler_iteration` server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in each scheduler's `sched_cycle_length` attribute. A scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes. A scheduler does not include the time it takes to query dynamic resources in its cycle measurement.

4.5.6.1 Triggers for Scheduling Cycle

A scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded
- A hook restarts the scheduling cycle

4.5.6.1.i Logging Scheduling Triggers

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See [section 9.4.4.2, “Scheduler Commands”, on page 433](#).

4.5.6.2 Actions During Scheduling Cycle

The following is a list of a scheduler's actions during a scheduling cycle. The list is not in any special order.

- A scheduler gets the state of the world:
 - A scheduler queries the server for the following:
 - Status of jobs in queues
 - All global server, queue, and host-level resources
 - Server, queue, vnode, and scheduler attribute settings
 - Reservations
 - A scheduler runs dynamic server resource queries for resources listed in the "server_dyn_res" line in sched_config
- A scheduler logs a message at the beginning of each scheduling cycle saying whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is of this form:


```
"It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```

or

```
"It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```
- Given scheduling policy, available jobs and resources, and scheduling cycle length, a scheduler examines as many jobs as it can, and runs as many jobs as it can.

4.5.7 How Available Consumable Resources are Counted

When a scheduler checks for available consumable resources, it uses the following calculation:

resources_available.<resource name> - total resources assigned for this resource

total resources assigned is the total amount of `resources_assigned.<resource name>` for all other running and exiting jobs and, at the server and vnodes, for started reservations.

For example, if a scheduler is calculating available memory, and two other jobs are running, each with 2GB of memory assigned, and `resources_available.mem` is 8GB, the scheduler figures that it has 4GB to work with.

4.5.8 Improving Scheduler Performance

4.5.8.1 Configuring Job Dispatch Rate via the `job_run_wait` Scheduler Attribute

You can use the `job_run_wait` scheduler attribute to instruct the scheduler to dispatch jobs without waiting, after waiting only for any `runjob` hooks to execute, or after waiting for any `execjob_*` hooks to execute. Allowed values for `job_run_wait`:

none: run next job immediately

runjob_hook: run next job after waiting for any `runjob` hooks to execute

execjob_hook: run next job after waiting for any `execjob_*` hooks to execute

4.5.8.1.i Recommendations for Configuring Job Dispatch Rate

The `job_run_wait` scheduler attribute replaces the `throughput_mode` scheduler attribute. We recommend using `job_run_wait`, and we recommend not setting or using `throughput_mode`.

If you set `job_run_wait` to *none* or *runjob_hook*, PBS changes `throughput_mode` to *True*.

If you set `job_run_wait` to *execjob_hook*, PBS changes `throughput_mode` to *False*.

If you set `throughput_mode` to *True*, PBS changes `job_run_wait` to *runjob_hook*.

If you set `throughput_mode` to *False*, PBS changes `job_run_wait` to *execjob_hook*.

4.5.8.2 Limiting Number of Jobs Queued in Execution Queues

If you limit the number of jobs queued in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the `queued_jobs_threshold` queue or server attribute. See [section 5.15.1.9, “How to Set Limits at Server and Queues”, on page 292](#).

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See [section 4.9.39, “Routing Jobs”, on page 205](#).

4.5.8.3 Setting Number of Scheduler Threads

By default, each scheduler starts one thread on its host. You can modify the number of threads a scheduler starts, either by starting the scheduler with `pbs_sched -t <num threads>`, or by setting the `PBS_SCHED_THREADS` configuration parameter in `pbs.conf`, or the `PBS_SCHED_THREADS` environment variable. The `pbs_sched -t` option overrides the environment variable, which overrides the value in `pbs.conf`.

4.6 Using Queues in Scheduling

A queue is a PBS mechanism for holding jobs. PBS has queue-based tools for handling jobs; for example, you can set queue-based limits on resource usage by jobs. PBS uses queues for a variety of purposes. Before reading this section, please familiarize yourself with the mechanics of creating and configuring queues, by reading [section 2.3, “Queues”, on page 23](#).

Queues are used in the following ways:

- Holding submitted jobs
- Prioritizing jobs and ordering job selection:
 - PBS provides tools for selecting jobs according to the queue they are in; see [section 4.3.5.4, “Using Queue-based Tools to Prioritize Jobs”, on page 68](#)
 - Queue priority can be used in calculating job priority; see [section 4.9.36, “Queue Priority”, on page 195](#)
- Providing tools for managing time slots
 - Reservations: you can reserve specific resources for defined time slots. Queues are used for advance and standing reservations; see [section 4.9.37, “Reservations”, on page 196](#), and [“Reserving Resources”, on page 137 of the PBS Professional User’s Guide](#)
 - Dedicated time; see [section 4.9.10, “Dedicated Time”, on page 128](#)
 - Primetime and holidays; see [section 4.9.34, “Using Primetime and Holidays”, on page 190](#)
- Routing jobs: Many ways to route jobs are listed in [section 4.9.39, “Routing Jobs”, on page 205](#)
- Providing tools for managing resources
 - Managing resource usage by users; see [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#)
 - Managing resource usage by jobs; see [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#)
 - Setting resource and job limits used for preemption: you can specify how much of a resource or how many jobs a user or group can use before their jobs are eligible to be preempted. See [section 5.15.1.4, “Hard and Soft Limits”, on page 286](#) and [section 4.9.33, “Using Preemption”, on page 180](#).
 - Assigning default resources to jobs; see [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#)

4.7 Scheduling Restrictions and Caveats

4.7.1 One Policy Per Scheduler

Each scheduler runs a single scheduling policy.

4.7.2 Jobs that Cannot Run on Current Resources

A scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. A scheduler counts resources only from those vnodes that are on line. If a vnode is marked *offline*, its resources are not counted.

A scheduler determines whether a job cannot run on current resources only when backfilling is used. If backfilling is turned off, then a scheduler won't determine whether or not a job has requested more than can be supplied by current resources. It decides only that it can't run now. If the job cannot run now because vnodes are unavailable, there is no log message. If the job requests more than is available in the partition managed by a scheduler, there is a log message. In both cases, the job stays queued.

4.7.3 Resources Not Controlled by PBS

When a scheduler runs each cycle, it gets the state of its world, including dynamic resources outside of the control of PBS. If non-PBS processes are running on the vnodes PBS uses, it is possible that another process will use enough of a dynamic resource such as scratch space to prevent a PBS job that requested that resource from running.

4.7.4 No Pinning of Processes to Cores

PBS does not pin processes to cores. This can be accomplished in the job launch script using, for example, `taskset` or `dplace`.

4.8 Errors and Logging

4.8.1 Logfile for scheduler

You can set a scheduler's logging to record different kinds of events. See [section 9.4.3.1.iii, “Specifying Scheduler Log Events”, on page 431](#).

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See [section 9.4.4.2, “Scheduler Commands”, on page 433](#).

4.9 Scheduling Tools

In this section (all of [section 4.9, “Scheduling Tools”, on page 104](#), and its subsections), we describe each scheduling tool, including how to configure it.

The following table lists PBS scheduling tools, with links to descriptions:

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Anti-express queue	soft queue limits	See section 4.9.1, “Anti-Express Queues”, on page 107
Associating vnodes with queues		See section 4.9.2, “Associating Vnodes with Queues”, on page 107
Backfilling	fairshare or preemption w/backfilling+strict ordering	See section 4.9.3, “Using Backfilling”, on page 109
Examining jobs queue-by-queue	round robin, queues as fair-share entities	See section 4.9.4, “Examining Jobs Queue by Queue”, on page 113
Checkpointing		See section 4.9.5, “Checkpointing”, on page 114
Organizing job chunks		See section 4.9.6, “Organizing Job Chunks”, on page 115
cron jobs		See section 4.9.7, “cron Jobs”, on page 115
Custom resources		See section 4.9.8, “Using Custom and Default Resources”, on page 116

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Cycle harvesting	reservations	See section 4.9.9, “Using Idle Workstation Cycle Harvesting” , on page 117
Dedicated time		See section 4.9.10, “Dedicated Time” , on page 128
Default resources		See section 4.9.8, “Using Custom and Default Resources” , on page 116
Dependencies		See section 4.9.11, “Dependencies” , on page 129
Dynamic resources (server & host)		See section 4.9.12, “Dynamic Resources” , on page 129
Eligible wait time for jobs		See section 4.9.13, “Eligible Wait Time for Jobs” , on page 129
Entity shares (was strict priority)	formula, fairshare, FIFO	See section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)” , on page 133
Estimating job start time		See section 4.9.15, “Estimating Job Start Time” , on page 133
Calculating job execution priority		See section 4.9.16, “Calculating Job Execution Priority” , on page 136
Express queues		See section 4.9.18, “Express Queues” , on page 139
Fairshare	strict ordering, using the fairshare_perc option to job_sort_key	See section 4.9.19, “Using Fairshare” , on page 139
FIFO		See section 4.9.20, “FIFO Scheduling” , on page 150
Formula		See section 4.9.21, “Using a Formula for Computing Job Execution Priority” , on page 151
Gating jobs at server or queue		See section 4.9.22, “Gating Jobs at Server or Queue” , on page 157
Managing application licenses		See section 4.9.23, “Managing Application Licenses” , on page 158
Limits on per-job resource usage		See section 4.9.24, “Limits on Per-job Resource Usage” , on page 158
Limits on project, user, and group jobs		See section 4.9.25, “Limits on Project, User, and Group Jobs” , on page 159
Limits on project, user, and group resource usage		See section 4.9.26, “Limits on Project, User, and Group Resource Usage” , on page 159
Load balancing	node_sort_key using unused or assigned options,	See section 4.9.27, “Using Load Balancing” , on page 159
Matching jobs to resources		See section 4.9.28, “Matching Jobs to Resources” , on page 159
Node grouping		See section 4.9.29, “Node Grouping” , on page 161

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Overrides		See section 4.9.30, “Overrides”, on page 162
Peer scheduling		See section 4.9.31, “Peer Scheduling”, on page 164
Placement sets		See section 4.9.32, “Placement Sets”, on page 168
Preemption	cgroups hook cannot be used with suspend/resume	See section 4.9.33, “Using Preemption”, on page 180
Preemption targets		See section 4.9.33.4, “Using Preemption Targets”, on page 182
Primetime and holidays		See section 4.9.34, “Using Primetime and Holidays”, on page 190
Provisioning		See section 4.9.35, “Provisioning”, on page 195
Queue priority		See section 4.9.36, “Queue Priority”, on page 195
Advance and standing reservations	cycle harvesting	See section 4.9.37, “Reservations”, on page 196
Round robin queue examination	by_queue	See section 4.9.38, “Round Robin Queue Selection”, on page 204
Routing jobs		See section 4.9.39, “Routing Jobs”, on page 205
Shared or exclusive vnodes and hosts		See section 4.9.41, “Shared vs. Exclusive Use of Resources by Jobs”, on page 210
Shrinking jobs to fit		See section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211
SMP cluster distribution	avoid_provision	See section 4.9.43, “SMP Cluster Distribution”, on page 217
Soft walltime		See section 4.9.44, “Using Soft Walltime”, on page 218 .
Sorting jobs using job_sort_key		See section 4.9.45, “Sorting Jobs on a Key”, on page 220
Sorting jobs on job's requested priority		See section 4.9.46, “Sorting Jobs by Requested Priority”, on page 222
Sorting queues (deprecated in 13.0)		See section 4.9.47, “Sorting Queues into Priority Order”, on page 222
Strict ordering	Backfilling combined with fairshare	See section 4.9.48, “Using Strict Ordering”, on page 223
Sorting vnodes on a key	smp_cluster_dist set to other than pack, or load balancing, with unused or assigned options to node_sort_key	See section 4.9.49, “Sorting Vnodes on a Key”, on page 224

4.9.1 Anti-Express Queues

An anti-express queue is a preemptable low-priority queue, designed for jobs that should run only when no other jobs need the resources. These jobs are preempted if any other job needs the resources. An anti-express queue has the lowest priority of all queues in this queue's partition. Jobs in this queue have a soft limit of zero, so that any job running from this queue is over its queue soft limit.

See [section 4.9.33, “Using Preemption”, on page 180](#).

4.9.1.1 Configuring Anti-express Queues via Priority

To configure an anti-express queue by using queue priority, do the following:

- Create an execution queue called *lowprio*:

```
Qmgr: create queue lowprio
Qmgr: set queue lowprio queue_type=e
Qmgr: set queue lowprio started=true
Qmgr: set queue lowprio enabled=true
```
- By default, all new queues have a priority of zero. Make sure all queues have a value set for priority, and that lowprio has the lowest priority:

```
Qmgr: set queue workq priority=10
```
- Set the soft limit on the number of jobs that can run from that queue to zero for all users:

```
Qmgr: set queue lowprio max_run_soft = "[u:PBS_GENERIC=0]"
```
- Make sure that jobs over their queue soft limits have lower preemption priority than normal jobs. Edit `<sched_priv directory>/sched_config`, and do the following:
 - Put "normal_jobs" before "queue_softlimits". For example:

```
preempt_prio: "express_queue, normal_jobs, queue_softlimits"
```
 - Use preemption:

```
preemptive_sched: True ALL
```

4.9.1.2 Configuring Anti-express Queues via Preemption Targets

To use preemption targets, include this queue in `Resource_List.preempt_targets` for all jobs. You can do this with a hook, with server and/or queue defaults, or by qaltering the jobs. Set each job's `Resource_List.preempt_targets=queue=<name of anti-express queue>`.

4.9.1.3 Anti-express Queue Caveats

If you use soft limits on the number of jobs that users can run at other queues, jobs that are over their soft limits at other queues will also have the lowest preemption priority.

4.9.2 Associating Vnodes with Queues

You can associate each vnode with one or more queues. When a vnode is associated with a queue, that means it accepts jobs from that queue only. You can associate one or more vnodes with multiple queues.

You do not need to associate vnodes with queues in order to have jobs run on the vnodes that have the right application, as long as the application is a resource that can be requested by jobs.

You can use custom host-level resources to associate one or more vnodes with more than one queue. A scheduler will use the resources for scheduling just as it does with any resource.

In order to map a vnode to more than one queue, you must define a new host-level string array custom resource. This string array holds a string that has the same value for the queue and vnode you wish to associate. The mechanism of association is that a job that lands in the queue inherits that value for the resource, and then the job can run only on vnodes having a matching value for the resource. You can associate more than one queue with a vnode by setting the resource to the same value at each queue.

In some cases, you can use the same resource to route jobs and to associate vnodes with queues. For the method described here, you use host-level resources to associate vnodes with queues. The rules for which resources can be used for routing are given in [section 2.3.6.4.iii, “Resources Used for Routing and Admittance”, on page 29](#). How jobs inherit resources is described in [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#).

4.9.2.1 Procedure to Associate Vnodes with Queues

To associate one or more vnodes with one or more queues, do the following:

1. Define the new host-level resource:

```
qmgr -c 'create resource <new resource> type=string_array, flag=h'
```

2. Instruct the scheduler to honor the resource. Add the new resource to `$<sched_priv directory>/sched_config`:

```
resources: "ncpus, mem, arch, host, vnode, <new resource>"
```

3. HUP the scheduler:

```
kill -HUP <scheduler PID>
```

4. Set each queue's `default_chunk` for the new resource to the value you are using to associate it with vnodes:

```
Qmgr: set queue <queue name> default_chunk.<new resource> = <value>
```

For example, if one queue is "MathQ" and one queue is "SpareQ", and the new resource is "Qlist", and you want to associate a set of vnodes and queues based on ownership by the math department, you can make the queue resource value be "math":

```
Qmgr: set queue MathQ default_chunk.Qlist = math
```

```
Qmgr: set queue SpareQ default_chunk.Qlist = math
```

5. Set the value for the new resource at each vnode:

```
Qmgr: set node <vnode name> resources_available.<new resource> = <associating value>
```

For example, to have the vnode named "Vnode1" associated with the queues owned by the math department:

```
Qmgr: set node Vnode1 resources_available.Qlist = math
```

4.9.2.2 Example of Associating Multiple Vnodes with Multiple Queues

Now, as an example, assume you have 2 queues: "PhysicsQ" and "ChemQ", and you have 3 vnodes: `vn[1]`, `vn[2]`, and `vn[3]`. You want Physics jobs to run on `vn[1]` and `vn[2]`, and you want Chem jobs to run on `vn[2]` and `vn[3]`. Each department gets exclusive use of one vnode, but both must share a vnode.

To achieve the following mapping:

```
PhysicsQ -->vn[1], vn[2]
```

```
ChemQ --> vn[2], vn[3]
```

Which is the same as:

```
vn[1] <-- PhysicsQ
```

```
vn[2] <-- PhysicsQ, ChemQ
```



```
vn[3] <-- ChemQ
```

1. Define the new host-level resource:

```
Qmgr: create resource Qlist type=string_array, flag=h
```

2. Instruct the scheduler to honor the resource. Add the new resource to `$<sched_priv directory>/sched_config`:

```
resources: "ncpus, mem, arch, host, vnode, Qlist"
```

3. HUP the scheduler:

```
kill -HUP <scheduler PID>
```

4. Add queue to vnode mappings:

```
Qmgr: s n vn[1] resources_available.Qlist="PhysicsQ"
```

```
Qmgr: s n vn[2] resources_available.Qlist= "PhysicsQ,ChemQ"
```

```
Qmgr: s n vn[3] resources_available.Qlist="ChemQ"
```

5. Force jobs to request the correct Qlist values:

```
Qmgr: s q PhysicsQ default_chunk.Qlist=PhysicsQ
```

```
Qmgr: s q ChemQ default_chunk.Qlist=ChemQ
```

4.9.3 Using Backfilling

Backfilling means fitting smaller jobs around the higher-priority jobs that a scheduler is going to run next, in such a way that the higher-priority jobs are not delayed. When a scheduler is using backfilling, the scheduler considers highest-priority jobs *top jobs*. Backfilling changes the algorithm that a scheduler uses to run jobs:

- When backfilling is not being used, a scheduler looks at each job in priority order, tries to run the job now, and if it cannot, it moves on to the next-highest-priority job.
- When backfilling is being used, a scheduler tries to run the top job now, and if it cannot, it makes sure that no other job that it runs in this cycle will delay the top job. It also fits smaller jobs in around the top job.

Backfilling allows you to keep resources from becoming idle when the top job cannot run.

Backfilling applies all of the time; it is not a prime option.

4.9.3.1 Glossary

Top job

A top job has the highest execution priority according to scheduling policy, and a scheduler plans resources and start time for this job first. Top jobs exist only when a scheduler is using backfilling.

Filler job

Smaller job that fits around top jobs. Running a filler job does not change the start time or resources for a top job. This job runs next only when backfilling is being used (meaning that a top job cannot start next because insufficient resources are available for the top job, but whatever is available is enough for the filler job).

4.9.3.2 Backfilling Separately at the Server and Queues

You can configure the number of top jobs that PBS backfills around by setting the value of the `backfill_depth` server and queue attributes. For example, if you set `backfill_depth` to 3, PBS backfills around the top 3 jobs. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

You can specify a different number of top jobs for each queue. You can also specify the number of top jobs for the server. Any queues that do not have their own backfill depth share in the server's backfill depth count. For example, you have three queues Q1, Q2, and Q3, and you set the backfill depth at Q1 to be 5 and the backfill depth at the server to be 3. In this example, the top 5 jobs in Q1 will run as soon as possible, and be backfilled around, but there are only 3 top job slots allocated to the jobs in Q2 and Q3.

If you do not set a value for the backfill depth at the server, it defaults to 1.

4.9.3.3 How Backfilling Works

A scheduler makes a list of jobs to run in order of priority, for any queue that has an individual backfill depth, for the server if there are queues without a backfill depth set. These lists are composed according to execution priority described in [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#). These are top jobs.

If you use backfilling, a scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in each list. A scheduler looks in each prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

A scheduler creates a fresh list of top jobs at every scheduling cycle, so if a new higher-priority job has been submitted, it will be considered.

You can use shrink-to-fit jobs to backfill into otherwise unusable time slots. PBS checks whether a shrink-to-fit job could shrink into the available slot, and if it can, runs it. See [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

Backfilling is useful in the following circumstances:

- When the `strict_ordering` scheduler parameter is turned on, filler jobs are fitted around higher-priority jobs. Without backfilling, no job runs if the top job cannot run. See [section 4.9.48, “Using Strict Ordering”, on page 223](#)

4.9.3.4 Backfilling Around *N* Jobs

You can configure the number of top jobs that PBS backfills around by setting the value of the `backfill_depth` server attribute. For example, if you set `backfill_depth` to 3, PBS backfills around the top 3 jobs. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

4.9.3.5 Backfilling Around Preempted Jobs

When you set the `sched_preempt_enforce_resumption` scheduler attribute to *True*, a scheduler adds preempted jobs to the set of jobs around which it backfills. A scheduler ignores `backfill_depth` when backfilling around jobs in the *Pre-empted* execution class. By default the `sched_preempt_enforce_resumption` scheduler attribute is *False*.

4.9.3.6 Configuring Backfilling

To configure backfilling, do the following:

1. Choose how many jobs to backfill around. If you want to backfill around more than 1 job, set the `backfill_depth` server attribute to the desired number. The default is `1`. Set this parameter to less than `100`.
2. Choose whether you want any queues to share the list of top jobs at the server. Do not set `backfill_depth` at those queues. If you want any queues to share this list, set the server's `backfill_depth` attribute to the desired value. The default is `1`. Set this parameter to less than `100`.
3. For the queues where you want a separate backfill depth, choose how many jobs to backfill around at each queue. Set the `backfill_depth` queue attribute to the desired number.
4. Make sure that jobs request `walltime` by making them inherit a `walltime` resource if they don't explicitly request it. For options, see [section 4.9.3.10.i, “Ensure Jobs Are Eligible for Backfilling”, on page 112](#).
5. Choose whether you want to backfill around preempted jobs. To do this, set the `sched_preempt_enforce_resumption` scheduler attribute to `True`.
6. Make sure that the `strict_ordering` scheduler parameter is set to `True` for all time if you use backfilling.

When most jobs become top jobs, they are counted toward the limit set in `backfill_depth`. Some top jobs are not counted toward `backfill_depth`. The following table shows how backfilling can be configured and which top jobs affect `backfill_depth`. Unless explicitly stated, top jobs are counted towards `backfill_depth`. A scheduler stops considering jobs as top jobs when it has reached `backfill_depth`, except for preempted jobs, which do not count toward that limit. When backfill is off, a scheduler does not have a notion of "top jobs".

Table 4-5: Configuring Backfilling

Parameter and Attribute Settings			When Classes Are Top Jobs		
<code>backfill_depth</code>	<code>strict_ordering</code>	<code>sched_preempt_enforce_resumption</code>	Express	Preempted	Normal
<code>>0</code>	<code>T</code>	<code>T</code>	Top jobs	Top jobs, not counted in <code>backfill_depth</code>	Top jobs
<code>>0</code>	<code>T</code>	<code>T</code>	Top jobs	Top jobs, not counted in <code>backfill_depth</code>	Top jobs
<code>>0</code>	<code>T</code>	<code>F</code>	Top jobs	Top jobs	Top jobs
<code>>0</code>	<code>T</code>	<code>F</code>	Top jobs	Top jobs	Top jobs
<code>>0</code>	<code>F</code>	<code>T</code>	No	Top jobs, not counted in <code>backfill_depth</code>	No
<code>>0</code>	<code>F</code>	<code>T</code>	No	Top jobs, not counted in <code>backfill_depth</code>	No
<code>>0</code>	<code>F</code>	<code>F</code>	No	No	No
<code>>0</code>	<code>F</code>	<code>F</code>	No	No	No

4.9.3.7 Backfilling and Strict Ordering

When you use strict ordering, a scheduler runs jobs in exactly the order of their priority. If `backfill_depth` is set to zero and the top job cannot run, no job is able to run. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See [section 4.9.48, “Using Strict Ordering”, on page 223](#).

4.9.3.8 Backfilling and Scheduler Cycle Speed

You can choose a trade-off between scheduling cycle speed and the fineness of the granularity with which estimated start times are calculated. You do this by setting the `opt_backfill_fuzzy` scheduler attribute via `qmgr`. You can choose *off*, *low*, *medium*, or *high*. For no speedup, choose *off*. For maximum speedup, choose *high*.

Qmgr: `set sched opt_backfill_fuzzy [off | low | medium | high]`

See [section 4.9.40, “Scheduler Cycle Speedup”, on page 209](#).

4.9.3.9 Attributes and Parameters Affecting Backfilling

`backfill_depth`

Server and queue attribute. Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#) and [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

`opt_backfill_fuzzy`

Scheduler attribute. You can use this setting to trade between scheduling cycle speed and estimated start time granularity. See [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

`sched_preempt_enforce_resumption`

Scheduler attribute. When this attribute is *True* and `backfill_depth` is greater than zero, a scheduler treats preempted jobs like top jobs and backfills around them. This effectively increases the value of `backfill_depth` by the number of preempted jobs.

The configuration parameters `backfill_prime` and `prime_exempt_anytime_queues` do not relate to backfilling. They control the time boundaries of regular jobs with respect to primetime and non-primetime. See [section 4.9.34, “Using Primetime and Holidays”, on page 190](#).

4.9.3.10 Backfilling Recommendations and Caveats

4.9.3.10.i Ensure Jobs Are Eligible for Backfilling

When calculating backfilling, PBS treats a job that has no `walltime` specified as if its `walltime` is eternity. A scheduler will never use one of these jobs as a filler job. You can avoid this by ensuring that each job has a realistic `walltime`, by using the following methods:

- At `qsub` time via a hook
- By setting the queue's `resources_default.walltime` attribute
- By setting the server's `resources_default.walltime` attribute
- At `qsub` time via the server's `default_qsub_arguments`

4.9.3.10.ii Number of Jobs to Backfill Around

The more jobs being backfilled around, the longer the scheduling cycle takes.

4.9.3.10.iii Dynamic Resources and Backfilling

Using dynamic resources and backfilling may result in some jobs not being run because a dynamic resource is temporarily unavailable. This may happen when a job requesting a dynamic resource is selected as the top job. A scheduler must estimate when resources will become available, but it can only query for resources available at the time of the query, not resources already in use, so it will not be able to predict when resources in use become available. Therefore the scheduler won't be able to schedule the job. In addition, since dynamic resources are outside of the control of PBS, they may be consumed between the time a scheduler queries for the resource and the time it starts a job.

4.9.3.10.iv Avoid Using Strict Ordering, Backfilling, and Fairshare

It is inadvisable to use strict ordering and backfilling with fairshare.

The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that while a large job waits for a slot, jobs from the same entity or group will be chosen as the filler jobs, and the usage from these small jobs will lower the priority of the large job.

For example, if a user has a large job that is the most deserving but cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from ever being likely to run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

4.9.3.10.v Using Preemption, Strict Ordering, and Backfilling

Using preemption with strict ordering and backfilling may reshuffle the top job(s) if high-priority jobs are preempted.

4.9.3.10.vi Warning About Backfilling and Provisioning

A scheduler will not run a job requesting an AOE on a vnode that has a top job scheduled on it in the future.

A scheduler will not use a job requesting an AOE as a top job.

4.9.3.10.vii Backfilling and Estimating Job Start Time

When a scheduler is backfilling around jobs, it estimates the start times and execution vnodes for the top jobs being back-filled around. See [section 4.9.15, “Estimating Job Start Time”, on page 133](#).

4.9.3.10.viii Using Strict Ordering and Backfilling with Only One of Primetime or Non-primetime

If you use backfilling, it is used all of the time. However, you can use strict ordering during primetime, non-primetime, or all the time. When PBS is using strict ordering and backfilling, a scheduler saves a spot for each high-priority job around which it is backfilling. If you configure PBS to use strict ordering and backfilling for only one of primetime or non-primetime, and you have large jobs that must wait a long time before enough resources are available, the saved spots can be lost in the transition.

4.9.4 Examining Jobs Queue by Queue

When a scheduler examines waiting jobs, it can either consider all of the jobs in its partition as a whole, or it can consider jobs queue by queue. When considering jobs queue by queue, a scheduler runs all the jobs it can from the first queue before examining the jobs in the next queue, and so on. This behavior is controlled by the `by_queue` scheduler parameter.

When the `by_queue` scheduler parameter is set to *True*, jobs in the highest-priority queue are evaluated as a group, then jobs in the next-highest priority queue are evaluated. In this case, PBS runs all the jobs it can from each queue before moving to the next queue, with the following exception: if there are jobs in the *Reservation*, *Express*, or *Preempted* job execution classes, those are considered before any queue. These classes are described in [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

The `by_queue` parameter applies to all of the queues in a scheduler's partition. This means that either all jobs are scheduled as if they are in one large queue, or jobs are scheduled queue by queue.

All queues are always sorted by queue priority. To set queue priority, set each queue's `priority` attribute to the desired value. A queue with a higher value is examined before a queue with a lower value. If you do not assign priorities to queues, their ordering is undefined. See [section 4.9.36, “Queue Priority”, on page 195](#).

The `by_queue` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

See [“by_queue” on page 252 of the PBS Professional Reference Guide](#).

4.9.4.1 Configuring PBS to Consider Jobs Queue by Queue

- Set the `by_queue` scheduler parameter to *True*
- Assign a priority to each queue
- Choose whether you want queue by queue during primetime, non-primetime, or both. If you want separate behavior for primetime and non-primetime, list `by_queue` twice. For example:
`by_queue True prime`
`by_queue False non_prime`

4.9.4.2 Parameters and Attributes Affecting Queue by Queue

- The `by_queue` scheduler parameter; see [“by_queue” on page 252 of the PBS Professional Reference Guide](#).
- The priority queue attribute; see [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

4.9.4.3 Caveats and Advice for Queue by Queue

- The `by_queue` scheduler parameter is overridden by the `round_robin` scheduler parameter when `round_robin` is set to *True*.
- When `by_queue` is *True*, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.
- When `by_queue` is *True*, job execution priority may be affected. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- The `by_queue` parameter is not required when using express queues.
- You can have FIFO scheduling for all your jobs across a given scheduler's partition, if you are using a single execution queue or have `by_queue` set to *False*. However, you can have FIFO scheduling for the jobs within each queue if you set `by_queue` to *True* and specify a different priority for each queue. See [section 4.9.20, “FIFO Scheduling”, on page 150](#).

4.9.5 Checkpointing

You can use checkpointing as a scheduling tool, by including it as a preemption method, an aid in recovery, a way to capture progress from a shrink-to-fit job, and when using the `qhold` command.

For a complete description of how to use and configure checkpointing, see [section 8.3, “Checkpoint and Restart”, on page 387](#).

4.9.5.1 Checkpointing as a Preemption Method

When a job is preempted via checkpointing, MoM runs the `checkpoint_abort` script, and PBS kills and requeues the job. When a scheduler elects to run the job again, the MoM runs the `restart` script to restart the job from where it was checkpointed. See [section 4.9.33, “Using Preemption”, on page 180](#).

4.9.5.2 Checkpointing as a Way to Capture Progress and Help Recover Work

When you use checkpointing to capture a job's progress before the job is terminated, for example when a shrink-to-fit job's wall time is exceeded, MoM runs the `snapshot checkpoint` script, and the job continues to run. See [section 8.3, “Checkpoint and Restart”, on page 387](#).

4.9.5.3 Checkpointing When Using the `qhold` Command

When the `qhold` command is used to hold a checkpointable job, MoM runs the `checkpoint_abort` script, and PBS kills, requeues, and holds the job. A job with a hold on it must have the hold released via the `qrls` command in order to be eligible to run. For a discussion of the use of checkpointing for the `qhold` command, see [section 8.3.7.6, “Holding a Job”, on page 399](#). See [“qhold” on page 150 of the PBS Professional Reference Guide](#) and [“qrls” on page 183 in the PBS Professional Installation & Upgrade Guide](#).

4.9.6 Organizing Job Chunks

You can specify how job chunks should be organized onto hosts or vnodes. Jobs can request their placement arrangement, and you can set defaults at queues and at the server to be inherited by jobs that do not request a placement. You can tell PBS to do the following:

- Put all chunks from a job onto a single host using the `place=pack` statement.
- Put each chunk on a separate host using the `place=scatter` statement. The number of chunks must be fewer than or equal to the number of hosts.
- Put each chunk on a separate vnode using the `place=vscatter` statement. The number of chunks must be fewer than or equal to the number of vnodes.
- Put each chunk anywhere using the `place=free` statement.

To specify a placement default, set `resources_default.place=<arrangement>`, where *arrangement* is *pack*, *scatter*, *vscatter*, or *free*. For example, to have the default at QueueA be *pack*:

```
Qmgr: set queue QueueA resources_default.place=pack
```

You can specify that job chunks must be grouped in a certain way. For example, to require that chunks all end up on a shared router, use this:

```
place=group=router
```

For more about jobs requesting placement, see [“Requesting Resources and Placing Jobs” on page 219 of the PBS Professional Reference Guide](#).

4.9.6.1 Caveats for Organizing Job Chunks

A placement specification for arrangement, sharing, and grouping is treated as one package by PBS. This means that if a job requests only one, any defaults set for the others are not inherited. For example, if you set a default of `place=pack:excl:group=router`, and a job requests only `place=pack`, the job does not inherit `excl` or `group=router`. See [“Requesting Resources and Placing Jobs” on page 219 of the PBS Professional Reference Guide](#).

4.9.7 cron Jobs

You can use `cron` jobs to make time-dependent modifications to settings, where you are scheduling according to time slots. For example, you can change settings for primetime and non-primetime configurations, making the following changes:

- Set nodes *offline* or not *offline*
- Change the number of `ncpus` on workstations
- Change the priority of queues, for example to change preemption behavior
- Start or stop queues
- Set primetime & non-primetime options

4.9.7.1 Caveats for cron Jobs

- Make sure that your `cron` jobs behave correctly when PBS is not running.
- Be careful when changing available resources, such as when offlining vnodes. You might prevent jobs from running that would otherwise run. For details, see [section 4.7.2, “Jobs that Cannot Run on Current Resources”, on page 103](#).

If PBS is down when your `cron` job runs, the change specified in the `cron` job won't happen. For example, if you use `cron` to offline a vnode and then bring it online later, it won't come online if PBS is down during the second operation.

4.9.8 Using Custom and Default Resources

The information in this section relies on understanding how jobs are allocated resources via inheriting defaults or via hooks. Before reading this section, please read [section 10.3, “Allocating Resources to Jobs”, on page 455](#).

For complete details of how to configure and use custom resources, please see [section 5.14, “Custom Resources”, on page 252](#).

You can use custom and default resources for several purposes:

- Routing jobs to the desired vnodes; see [section 4.9.8.2, “Using Custom Resources to Route Jobs”, on page 116](#)
- Assigning execution priority to jobs; see [section 4.9.8.3, “Using Custom Resources to Assign Job Execution Priority”, on page 117](#)
- Tracking and controlling the allocation of resources; see [section 4.9.8.4, “Using Custom Resources to Track and Control Resource Allocation”, on page 117](#)
- Representing elements such as GPUs, FPGAs, and switches; see [section 4.9.8.5, “Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.”, on page 117](#)
- Shrinking job `walltimes` so that they can run in time slots that are less than the expected maximum. See [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

4.9.8.1 Techniques for Allocating Custom Resources to Jobs

In addition to using custom resources to represent physical elements such as GPUs, you can use custom resources as tags that you attach to jobs in order to help schedule the jobs. You can make these custom resources into tools that can be used only for managing jobs, by making them unalterable and unrequestable, and if desired, invisible to users.

For how to assign custom and default resources to jobs, see [section 10.3, “Allocating Resources to Jobs”, on page 455](#).

4.9.8.2 Using Custom Resources to Route Jobs

You can use several techniques to route jobs to the desired queues and/or vnodes. Depending on your partition's or site's configuration, you may find it helpful to use custom resources with one or more of these techniques.

- You can force users to submit jobs to the desired queues by setting resource limits at queues. You can use custom resources to represent arbitrary elements, for example, department. In this case you could limit which department uses each queue. You can set a default value for the department at the server, or create a hook that assigns a value for the department.

For how queue resource limits are applied to jobs, see [section 2.3.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 29](#).

- Use default resources or a hook to assign custom resources to jobs when the jobs are submitted. Send the jobs to routing queues, then route them, using the resources, to other queues inside or outside the PBS partition or complex. Again, custom resources can represent arbitrary elements.

For how routing queues work, see [section 2.3.6, “Routing Queues”, on page 27](#)

- Use peer scheduling to send jobs between PBS partitions or complexes. You can set resource limits on the furnishing queue in order to limit the kinds of jobs that are peer scheduled. You can assign custom resources to jobs to represent arbitrary elements, for example peer queueing only those jobs from a specific project. You can assign the custom resource by having the job inherit it or via a hook.

For how to set up peer scheduling, see [section 4.9.31, “Peer Scheduling”, on page 164](#)

- You can route jobs from specific execution queues to the desired vnodes, by associating the vnodes with the queues. See [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).
- You can create placement sets so that jobs are placed according to resource values. Placement sets are created where vnodes share a value for a resource; you can use custom resources to create the placement sets you want. See [section 4.9.32, “Placement Sets”, on page 168](#).

4.9.8.3 Using Custom Resources to Assign Job Execution Priority

You can use custom resources as coefficients in the job sorting formula. You can assign custom resources to jobs using the techniques listed in [section 10.3, “Allocating Resources to Jobs”, on page 455](#). The value of each custom resource can be based on a project, an application, etc.

For example, you can create a custom resource called "ProjPrio", and the jobs that request the "Bio" project can be given a value of 5 for ProjPrio, and the jobs that request the "Gravel" project can be given a value of 2 for ProjPrio. You can assign this value in a hook or by routing the jobs into special queues from which the jobs inherit the value for ProjPrio.

For information on using the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

4.9.8.4 Using Custom Resources to Track and Control Resource Allocation

You can use resources to track and control usage of things like CPUs and memory. For example, you might want to limit the number of jobs using a particular vnode. See [section 5.10, “Using Resources to Track and Control Allocation”, on page 249](#).

4.9.8.5 Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.

You can use custom resources to represent GPUs, FPGAs, high performance switches, etc. For examples, see [section 5.14.7, “Using GPUs”, on page 279](#), and [section 5.14.8, “Using FPGAs”, on page 282](#).

4.9.9 Using Idle Workstation Cycle Harvesting

You can configure workstations at your partition or site so that PBS can run jobs on them when their "owners" are away and they are idle. This is called *idle workstation cycle harvesting*. This can give your partition or site additional resources to run jobs during nights and weekends, or even during lunch.

You can configure PBS to use the following methods to decide when a workstation is not being used by its owner:

- Keyboard/mouse activity
- X-Window monitoring
- Load average (not recommended)

On some systems cycle harvesting is simple to implement, because the console, keyboard, and mouse device access times are periodically updated by the operating system. The PBS MoM process can track this information, and mark the vnode *busy* if any of the input devices is in use. On other systems, however, this data is not available: on some machines, PBS can monitor the X-Window system in order to obtain interactive idle time, and on others, PBS itself monitors keyboard and mouse activity.

Jobs on workstations that become *busy* are not migrated; they remain on the workstation until they complete execution, are rerun, or are deleted.

4.9.9.1 Platforms Supporting Cycle Harvesting

Due to different operating system support for tracking mouse and keyboard activity, the availability and method of support for cycle harvesting varies based on the computer platform in question. The following table lists the method and support for each platform.

Table 4-6: Cycle Harvesting Support Methods

System	Status	Method	Reference
Linux	supported	keyboard/mouse	section 4.9.9.3, “Cycle Harvesting Based on Keyboard/Mouse Activity”, on page 119
Windows	supported	keyboard/mouse	section 4.9.9.4, “Cycle Harvesting on Windows”, on page 119

4.9.9.2 The \$kbd_idle MoM Configuration Parameter

Cycle harvesting based on keyboard/mouse activity and X-Windows monitoring is controlled by the `$kbd_idle` MoM configuration parameter in `PBS_HOME/mom_priv/config` on the workstation in question. This parameter has the following format:

`$kbd_idle <idle_wait> <min_use> <poll_interval>`

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use.

idle_wait

Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.

Must be set to value greater than 0 for cycle harvesting to be enabled.

Format: Integer

No default

min_use

Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.

Format: Integer

Default: 10

poll_interval

Interval, in seconds, at which MoM checks for keyboard and mouse activity.

Format: Integer

Default: 1

4.9.9.3 Cycle Harvesting Based on Keyboard/Mouse Activity

PBS can monitor a workstation for keyboard and mouse activity, and run batch jobs on the workstation when the keyboard and mouse are not being used. PBS sets the state of the vnode to either *free* or *busy*, depending on whether or not there is keyboard or mouse activity, and runs jobs only when the state of the vnode is *free*. PBS sets the state of the vnode to *free* when the vnode's mouse and keyboard have shown no activity for the specified amount of time. If PBS determines that the vnode is being used, it sets the state of the vnode to *busy* and suspends any running jobs, setting their state to *U (user busy)*.

This method is used for Linux operating systems.

4.9.9.3.i Configuring Cycle Harvesting Using Keyboard/Mouse Activity

To configure cycle harvesting using keyboard and mouse activity, do the following:

1. Set the `$kbd_idle` MoM configuration parameter by editing the `$kbd_idle` parameter in `PBS_HOME/mom_priv/config` on the workstation.
2. HUP the MoM on the workstation:

```
kill -HUP <pbs_mom PID>
```

4.9.9.3.ii Example of Cycle Harvesting Using Keyboard/Mouse Activity

The following is an example setting for the parameter:

```
$kbd_idle 1800 10 5
```

This setting for the parameter in MoM's `config` file specifies the following:

- PBS marks the workstation as *free* if the keyboard and mouse are idle for 30 minutes (1800 seconds)
- PBS marks the workstation as *busy* if the keyboard or mouse is used for 10 consecutive seconds
- The states of the keyboard and mouse are to be checked for activity every 5 seconds

Here, we walk through how this example would play out, to show the roles of the arguments to the `$kbd_idle` parameter:

Let's start with a workstation that has been in use for some time by its owner. The workstation is in state *busy*.

Now the owner goes to lunch. After 1800 seconds (30 minutes), PBS changes the workstation's state to *free* and starts a job on the workstation.

Some time later, someone walks by and moves the mouse or enters a command. Within the next 5 seconds (idle poll period), `pbs_mom` notes the activity. The job is suspended and placed in state *U*, and the workstation is marked *busy*.

If 10 seconds pass and there is no additional keyboard/mouse activity, the job is resumed and the workstation again is either *free* (if any CPUs are available) or *job-busy* (if all CPUs are in use.)

However, if keyboard/mouse activity continues during that 10 seconds, the workstation remains *busy* and the job remains suspended for at least the next 1800 seconds.

4.9.9.3.iii Caveats for Cycle Harvesting Using Keyboard/Mouse Activity

- There is no default for `idle_wait`; you must set it to a value greater than 0 in order to enable cycle harvesting using keyboard/mouse activity.

4.9.9.4 Cycle Harvesting on Windows

A process called `pbs_idled` monitors keyboard and mouse activity and keeps MoM informed of user activity. The user being monitored can be sitting at the machine, or using a remote desktop.

The `pbs_idled` process is managed in one of two ways. PBS can use a service called *PBS_INTERACTIVE* to monitor the user's session. If the *PBS_INTERACTIVE* service is registered, MoM starts the service, and the service starts and stops `pbs_idled`. The *PBS_INTERACTIVE* service runs under a local system account. PBS uses the *PBS_INTERACTIVE* service only where partition or site policy allows a local system account to be a service account. If this is not allowed (so the service is not registered), `pbs_idled` is started and stopped using the log on/log off script. Do not use both the *PBS_INTERACTIVE* service and a log on/log off script.

A `pbs_idled` process monitors the keyboard and mouse activity while a user is logged in. This process starts when the user logs on, and stops when the user logs off. Only a user with administrator privileges, or the user being monitored, can stop `pbs_idled`.

MoM uses two files to communicate with `pbs_idled`:

- MoM creates `PBS_HOME/spool/idle_poll_time` and writes the value of her `$kbd_idle` polling interval parameter to it. The `pbs_idled` process reads the value of the polling interval from `idle_poll_time`.
- MoM creates `PBS_HOME/spool/idle_touch`. The `pbs_idled` process updates the time stamp of the `idle_touch` file when a user is active, and MoM reads the time stamp.

4.9.9.4.i Configuring Cycle Harvesting on Windows

To configure cycle harvesting, do the following:

1. Make sure that you are a user with administrator privileges.
2. Set the `$kbd_idle` MoM configuration parameter by editing the `$kbd_idle` parameter in `PBS_HOME/mom_priv/config` on the workstation.
3. Configure how `pbs_idled` starts:
 - a. If your policy allows a local system account to be a service account, register the *PBS_INTERACTIVE* service:
`pbs_interactive -R`
 - b. If your policy does not allow a local system account to be a service account:
 1. Configure the log on script as described in [section 4.9.9.4.ii, “Configuring pbs_idled in Log On Script in Domain Environment”, on page 121](#).
 2. Configure the log off script as described in [section 4.9.9.4.iii, “Configuring pbs_idled in Log Off Script in Domain Environment”, on page 122](#).
4. Restart the MoM.

4.9.9.4.ii Configuring pbs_idled in Log On Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open *Administrator Tools*.
3. In *Administrator Tools*, open *Active Directory Users and Computers*.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on *Properties*.
6. Go to the *Group Policy* tab under the *Properties* window.
7. Click on *New*.
8. Type "LOG-IN-OUT-SCRIPT" as the name of the policy.
9. Select the Group Policy Object you have just created; click *Edit*. The Group Policy Object editing window will open.
10. Open *Window Settings* in *User Configuration*.
11. Open *Scripts (Logon/Logoff)*.
12. Open *Logon*. A *Logon Properties* window will open.
13. Open Notepad in another window. In Notepad, you create the command that starts the pbs_idled process:

```
pbs_idled start
```
14. Save that document as "*pbs_idled_logon.bat*".
15. In the *Logon Properties* window, click on *Show Files*. A logon script folder will open in a new window.
16. Copy *pbs_idled_logon.bat* into the logon script folder and close the logon script folder window.
17. In the *Logon Properties* window, click on *Add*, and then click on *Browse*. Select *pbs_idled_logon.bat* and then click on *Open*.
18. Click on *OK*, then *Apply*, then again *OK*.
19. Close the Group Policy Object editor and the *Properties* window.
20. Close the *Active Directory Users and Computers* window.
21. Close the *Administrator Tools* window.

4.9.9.4.iii Configuring pbs_idled in Log Off Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open *Administrator Tools*.
3. In *Administrator Tools*, open *Active Directory Users and Computers*.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on *Properties*.
6. Go to the *Group Policy* tab under the *Properties* window.
7. Click on *New*.
8. Type "LOG-IN-OUT-SCRIPT" as the name of the policy.
9. Select the Group Policy Object you have just created; click *Edit*. The Group Policy Object editing window will open.
10. Open *Window Settings* in *User Configuration*.
11. Open *Scripts (Logon/Logoff)*.
12. Open *Logoff*. A *Logoff Properties* window will open.
13. Open Notepad in another window. In Notepad, you create the command that stops the pbs_idled process:
`pbs_idled stop`
14. Save that document as "*pbs_idled_logoff.bat*".
15. In the *Logoff Properties* window, click on *Show Files*. A logoff script folder will open in a new window.
16. Copy *pbs_idled_logoff.bat* into the logoff script folder and close the logoff script folder window.
17. In the *Logoff Properties* window, click on *Add*, and then click on *Browse*. Select *pbs_idled_logoff.bat* and then click on *Open*.
18. Click on *OK*, then *Apply*, then again *OK*.
19. Close the Group Policy Object editor and the *Properties* window.
20. Close the *Active Directory Users and Computers* window.
21. Close the *Administrator Tools* window.

4.9.9.4.iv The PBS_INTERACTIVE Service

The PBS_INTERACTIVE service starts the pbs_idled process, as the current user, in the current active user's session. Each time a user logs on, the service starts a pbs_idled for that user, and when that user logs off, the service stops that user's pbs_idled process.

The service runs under a local system account. If your policy allows a local system account to be a service account, you can use PBS_INTERACTIVE. Otherwise you must configure pbs_idled in log on/log off scripts.

If you have configured the \$kbd_idle MoM parameter, and you have registered the service, MoM starts the service. The service cannot be started manually.

If you will use PBS_INTERACTIVE, you must register the service. The installer cannot register the service.

- To register the PBS_INTERACTIVE service:

```
pbs_interactive -R
```

Upon successful execution of this command, the following message is displayed:

"Service PBS_INTERACTIVE installed successfully"

- To unregister the PBS_INTERACTIVE service:

pbs_interactive -U

Upon successful execution of this command, the following message is displayed:

"Service PBS_INTERACTIVE uninstalled successfully"

- To see the version number for PBS_INTERACTIVE service:

pbs_interactive --version

4.9.9.4.v Errors and Logging

If the \$kbd_idle MoM parameter is configured, MoM attempts to use cycle harvesting. MoM looks for the PBS_INTERACTIVE service in the Service Control Manager. If she finds the service, she starts it.

1. If she cannot find the service, MoM logs the following message at event class 0x0002:
"Can not find PBS_INTERACTIVE service, Continuing Cycle Harvesting with Logon/Logoff Script"
2. MoM looks for PBS_HOME/spool/idle_touch. If she finds it, she uses cycle harvesting.
3. If she cannot find the file, MoM disables cycle harvesting and logs the following message at event class 0x0002:
"Cycle Harvesting Failed, Please contact Admin"

MoM logs the following messages at event class 0x0001.

- If MoM fails to open the Service Control Manager:
"OpenSCManager failed for PBS_INTERACTIVE"
- If MoM fails to open the PBS_INTERACTIVE service:
"OpenService failed for PBS_INTERACTIVE"
- If MoM fails to start the PBS_INTERACTIVE service:
"Could not start PBS_INTERACTIVE service"
- If MoM fails to get status information about the PBS_INTERACTIVE service:
"Can not get information about PBS_INTERACTIVE service"
- If MoM fails to send a stop control message to the PBS_INTERACTIVE service:
"Could not stop PBS_INTERACTIVE service"
- If the PBS_INTERACTIVE service does not respond in a timely fashion:
"PBS_INTERACTIVE service did not respond in timely fashion"
- If MoM fails to create idle_touch and idle_poll_time in PBS_HOME/spool directory:
"Can not create file < full path of idle file >"
- If MoM fails to write the idle polling interval into PBS_HOME/spool/idle_poll_time:
"Can not write idle_poll time into < full path of idle_poll_time file > file"

4.9.9.4.vi Caveats for Cycle Harvesting on Windows

- Under Windows, if the pbs_idled process is killed, cycle harvesting will not work.
- Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in, and users are not employing Switch User.
- Do not use both the PBS_INTERACTIVE service and a log on/log off script.

4.9.9.5 Cycle Harvesting by Monitoring X-Windows

On Linux machines where the OS does not periodically update console, keyboard, and mouse device access times, PBS can monitor X-Window activity instead. PBS uses an X-Window monitoring process called `pbs_idled`. This process runs in the background and monitors X and reports to the `pbs_mom` whether or not the vnode is idle. `pbs_idled` is located in `$PBS_EXEC/sbin`.

To configure PBS for cycle harvesting by monitoring X-Windows, perform the following steps:

1. Create a directory for `pbs_idled`. This directory must have the same permissions as `/tmp` (i.e. mode `1777`). This will allow the `pbs_idled` program to create and update files as the user, which is necessary because the program runs as the user. For example:

```
mkdir PBS_HOME/spool/idledir
chmod 1777 PBS_HOME/spool/idledir
```

2. Turn on keyboard idle detection in the MoM config file:

```
$kbd_idle <idle wait value>
```

3. Include `pbs_idled` as part of the X-Windows startup sequence.

The best and most secure method of starting `pbs_idled` is via the system-wide `Xsession` file. This is the script which is run by `xdm` (the X login program) and sets up each user's X-Windows environment.

You **must** place the startup line for `pbs_idled` before that of the window manager.

You **must** make sure that `pbs_idled` runs in the background.

On systems that use `Xsession` to start desktop sessions, insert a line invoking `pbs_idled` near the top of the file.

For example, insert the following line in a Linux `Xsession` file:

```
/usr/pbs/sbin/pbs_idled &
```

If access to the system-wide `Xsession` file is not available, you can add `pbs_idled` to every user's personal `.xsession` or `.xinitrc` file, depending on the local OS requirements for starting X-windows programs upon login.

4.9.9.6 Cycle Harvesting Based on Load Average

As of version 2022.1, the `load_balancing` scheduler parameter is **removed**.

Cycle harvesting based on load average means that PBS monitors each workstation's load average, runs jobs where workstations have loads below a specified level, and suspends any batch jobs on workstations whose load has risen above the limit you set. When a workstation's owner uses the machine, the workstation's load rises.

When you configure cycle harvesting based on load average, you are performing the same configuration as for load balancing using load average. For a complete description of load balancing, see [section 4.9.27, “Using Load Balancing”, on page 159](#).

4.9.9.6.i Attributes and Parameters Affecting Cycle Harvesting Based on Load Average

`$ideal_load <load>`

MoM parameter. Defines the load below which the vnode is not considered to be *busy*. Used with the `$max_load` directive.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

\$max_load <load> [suspend]

MoM parameter. Defines the load above which the vnode is considered to be *busy*. Used with the `$ideal_load` directive. No new jobs are started on a *busy* vnode.

The optional `suspend` directive tells PBS to suspend jobs running on the node if the load average exceeds the `$max_load` number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

We recommend setting this to a slightly higher value than your target load (which is typically the number of CPUs), for example `.25 + ncpus`.

Example:

```
$max_load 3.25
```

Format: *Float*

Default: number of CPUs

resv_enable

Vnode attribute. Controls whether the vnode can be used for advance and standing reservations. When set to *True*, this vnode can be used for reservations.

Format: *Boolean*

Default: *True*

no_multinode_jobs

Vnode attribute. Controls whether jobs which request more than one chunk are allowed to execute on this vnode. When set to *True*, jobs requesting more than one chunk are not allowed to execute on this vnode.

Format: *Boolean*

Default: *False*

4.9.9.6.ii How Cycle Harvesting Based on Load Average Works

Cycle harvesting based on load average means that PBS monitors the load average on each machine. When the load on a workstation is below what is specified in the `$ideal_load` MoM parameter, PBS sets the state of the workstation to *free*. A scheduler will run jobs on vnodes whose state is *free*. When the load on a workstation exceeds the setting for `$max_load`, PBS sets the state of the workstation to *busy*, and suspends jobs running on the workstation. PBS does not start jobs on a vnode whose state is *busy*. When the load drops below the setting for `$ideal_load`, PBS sets the state to *free*, and resumes the jobs that were running on the workstation.

PBS thinks that a 1-CPU job raises a vnode's load by 1. On machines being used for cycle harvesting, you set the values for `$max_load` and `$ideal_load` to reasonable limits. On other machines, you set these to values that will never be exceeded, so that load is effectively ignored.

On machines where these parameters are unset, the vnode's state is not set according to its load, so jobs are not suspended because a vnode is busy. However, if `$max_load` and `$ideal_load` are unset, they are treated as if they have the same value as `resources_available.ncpus`, and because there is usually a small background load, PBS will lose the use of a CPU's worth of load.

4.9.9.6.iii Configuring Cycle Harvesting Based on Load Average

To set up cycle harvesting for idle workstations based on load average, perform the following steps:

1. If PBS is not already installed on the target execution workstations, do so now, selecting the execution-only install option. See the PBS Professional Installation & Upgrade Guide.
2. Edit the `PBS_HOME/mom_priv/config` configuration file on each target execution workstation, adding the `$max_load` and `$ideal_load` configuration parameters. Make sure they have values that will not interfere with proper operation. See [section 4.9.9.6.v, “Caveats for Cycle Harvesting Based on Load Average”, on page 126](#).

```
$max_load <load limit that allows jobs to run>
```

```
$ideal_load <load at which to start jobs>
```

3. Edit the `PBS_HOME/mom_priv/config` configuration file on each machine where you are not using cycle harvesting, adding the `$max_load` and `$ideal_load` configuration parameters. Make sure they have values that will never be exceeded.

```
$max_load <load limit that will never be exceeded>
```

```
$ideal_load <load limit that will never be exceeded>
```

4. HUP the MoM:

```
kill -HUP <pbs_mom PID>
```

5. If you wish to oversubscribe the vnode's CPU(s), set its `resources_available.ncpus` to a higher number. Do this only on single-vnode machines. You must be cautious about matching `ncpus` and `$max_load`. See ["Caveats for Cycle Harvesting Based on Load Average" on page 126 in the PBS Professional Administrator's Guide](#).

6. HUP the scheduler:

```
kill -HUP <pbs_sched PID>
```

7. Set the vnode's `resv_enable` attribute to *False*, to prevent the workstation from being used for reservations.

```
Qmgr: set node <vnode name> resv_enable = False
```

8. Set the vnode's `no_multinode_jobs` attribute to *True*, to prevent the workstation from stalling multi-chunk jobs.

```
Qmgr: set node <vnode name> no_multinode_jobs = True
```

4.9.9.6.iv Viewing Load Average Information

You can see the state of a vnode using the `pbsnodes -a` command.

4.9.9.6.v Caveats for Cycle Harvesting Based on Load Average

- Be careful with the settings for `$ideal_load` and `$max_load`. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working.
- For information about keeping your partition or site running smoothly using `$max_load` and `$ideal_load`, see [section 8.6.6, “Managing Load Levels on Vnodes”, on page 415](#)
- If you set `ncpus` higher than the number of actual CPUs, and set `$max_load` higher to match, keep in mind that the workstation user could end up with an annoyingly slow workstation. This can happen when PBS runs jobs on the machine, but the combined load from the jobs and the user is insufficient for MoM to report being busy.

4.9.9.7 Cycle Harvesting and File Transfers

The cycle harvesting feature interacts with file transfers in one of two different ways, depending on the method of file transfer:

- If the user's job includes file transfer commands (such as `rscp` or `scp`) within the job script, and such a command is running when PBS decides to suspend the job on the vnode, then the file transfer is suspended as well.
- If the job has PBS file staging parameters (i.e. `stagein=`, `stageout=file1...`), and the load goes above `$max_load`, the file transfer is not suspended. This is because the file staging is not part of the job script execution, and is not subject to suspension. See ["Detailed Description of Job Lifecycle", on page 39 of the PBS Professional User's Guide](#).

4.9.9.8 Parallel Jobs With Cycle Harvesting

Cycle harvesting is not recommended for hosts that will run multi-host jobs. However, you may find that your partition or site benefits from using cycle harvesting on these machines. We provide advice on how to prevent cycle harvesting on these machines, and advice on how to accomplish it.

4.9.9.8.i General Advice: Parallel Jobs Not Recommended

Cycle harvesting is somewhat incompatible with multi-host jobs. If one of the hosts being used for a parallel job running on several hosts is being used for cycle harvesting, and the user types at the keyboard, job execution will be delayed for the entire job because the tasks running on that host will be suspended.

To prevent a machine which is being used for cycle harvesting from being assigned a multi-host job, set the vnode's `no_multinode_jobs` attribute to `True`. This attribute prevents a host from being used by jobs that span multiple hosts.

4.9.9.8.ii How to Use Cycle Harvesting with Multi-host Jobs

When a single-host job is running on a workstation configured for cycle harvesting, and that host becomes *busy*, the job is suspended. However, suspending a multi-host parallel job may have undesirable side effects because of inter-process communications. For a job which uses multiple hosts when one or more of the hosts becomes *busy*, the default action is to leave the job running.

However, you can specify that the job should be requeued and subsequently re-scheduled to run elsewhere when any of the hosts on which the job is running becomes *busy*. To enable this action, add the following parameter to MoM's configuration file:

```
$action multinodebusy 0 requeue
```

where `multinodebusy` is the action to modify; `"0"` (zero) is the action timeout value (it is ignored for this action); and `requeue` is the new action to perform. The only action that can be performed is requeueing.

Multi-host jobs which are not rerunnable (i.e. those submitted with the `qsub -rn` option) will be killed if the `requeue` argument is configured for the `multinodebusy` action and a vnode becomes busy.

4.9.9.9 Cycle Harvesting Caveats and Restrictions

4.9.9.9.i Cycle Harvesting and Multi-host Jobs

Cycle harvesting is not recommended for hosts that will run multi-host jobs. See [section 4.9.9.8.i, "General Advice: Parallel Jobs Not Recommended", on page 127](#).

4.9.9.9.ii Cycle Harvesting and Reservations

Cycle harvesting is incompatible with jobs in reservations. Reservations should not be made on a machine used for cycle harvesting, because the user may appear during the reservation period and use the machine's keyboard. This will suspend the jobs in the reservation, defeating the purpose of making a reservation.

To prevent a vnode which is being used for cycle harvesting from being used for reservations, set the vnode's `resv_enable` attribute to `False`. This attribute controls whether the vnode can be used for reservations.

4.9.9.9.iii File Transfers with Cycle Harvesting

File transfers behave differently depending on job details. See [section 4.9.9.7, “Cycle Harvesting and File Transfers”, on page 127](#).

4.9.9.9.iv Cycle Harvesting on Windows

- Under Windows, if the `pbs_idled` process is killed, cycle harvesting will not work.
- Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in.

4.9.10 Dedicated Time

PBS provides a feature called *dedicated time* which allows you to define times during which the only jobs that can run are the ones in dedicated queues. You can use dedicated time for things like upgrades.

You can define multiple dedicated times. Any job in a dedicated time queue must have a `walltime` in order to run. Jobs without walltimes will never run. PBS won't let a reservation conflict with dedicated time. Hooks should not access or modify the dedicated time file.

For information on configuring dedicated time queues, see [section 2.3.5.2.i, “Dedicated Time Queues”, on page 26](#).

4.9.10.1 Dedicated Time File

You define dedicated time by adding one or more time slots in the file `<sched_priv directory>/dedicated_time`. A time slot is a start date and start time and an end date and end time. Format:

```
<start date> <start time> <end date> <end time>
```

expressed as

```
MM/DD/YYYY HH:MM MM/DD/YYYY HH:MM
```

Any line whose first non-whitespace character is a pound sign (“#”) is a comment.

Example:

```
#Dedicated time for maintenance
04/15/2007 12:00 04/15/2007 15:30
```

A sample dedicated time file (`PBS_EXEC/etc/pbs_dedicated`) is included in the installation.

The dedicated time file is read on startup and HUP.

4.9.10.2 Steps in Defining Dedicated Time

You define dedicated time by performing the following steps:

1. Edit the file `<sched_priv directory>/dedicated_time` and add one or more time slots.
2. HUP or restart the scheduler:

Linux:

```
kill -HUP <pbs_sched PID>
```

4.9.10.3 Recommendations for Dedicated Time

If you need to set up dedicated time for something like system maintenance, you may want to avoid having the machines become idle for a significant period before dedicated time starts. You can allow jobs to shrink their walltimes to fit into those shorter-than-normal slots before dedicated time. See [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

4.9.11 Dependencies

PBS allows job submitters to specify dependencies between jobs, for example specifying that job J2 can only run if job J1 finishes successfully. In addition, you can add dependencies to existing jobs via a hook, default arguments to `qsub`, or via the `qalter` command.

For a description of how job dependencies work, see ["Using Job Dependencies", on page 109 of the PBS Professional User's Guide](#).

For how to use hooks, see the PBS Professional Hooks Guide.

For how to add default `qsub` arguments, see ["Server Attributes" on page 281 of the PBS Professional Reference Guide](#).

For how to use the `qalter` command, see ["qalter" on page 130 of the PBS Professional Reference Guide](#).

4.9.12 Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for application licenses and scratch space. You can represent elements that are available to the entire PBS partition or complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see [section 5.14.3.1.iii, "Example of Configuring Dynamic Server-level Resource", on page 264](#). For an example of configuring a dynamic host-level resource, see [section 5.14.4.1.i, "Example of Configuring Dynamic Host-level Resource", on page 265](#).

For a complete description of how to create and use dynamic resources, see [section 5.14, "Custom Resources", on page 252](#).

4.9.13 Eligible Wait Time for Jobs

PBS provides a method for tracking how long a job that is eligible to run has been waiting to run. By "eligible to run", we mean that the job could run if the required resources were available. The time that a job waits while it is not running can be classified as "eligible" or "ineligible". Roughly speaking, a job accrues eligible wait time when it is blocked due to a resource shortage, and accrues ineligible wait time when it is blocked due to project, user, or group limits. A job can be accruing any of the following kinds of time. A job can only accrue one kind of wait time at a time, and cannot accrue wait time while it is running.

4.9.13.1 Types of Time Accrued

eligible_time

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by the other kinds of time. For a job currently accruing `eligible_time`, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f` by job owner, Manager and Operator. Settable by Operator or Manager.

ineligible_time

The amount of wall clock time a job has accrued because the job is blocked by limits on the job's project, owner, or group, or because the job is blocked because of its state.

run_time

The amount of wall clock time a job has spent running.

exiting

The amount of wall clock time a job has spent exiting.

initial_time

The amount of wall clock wait time a job has accrued before the type of wait time has been determined.

4.9.13.2 How Eligible Wait Time Works

A job accrues `ineligible_time` while it is blocked by project, user, or group limits, such as:

```
max_run
max_run_soft
max_run_res.<resource name>
max_run_res_soft.<resources>
```

A job also accrues `ineligible_time` while it is blocked due to a user hold or while it is waiting for its start time, such as when submitted via

```
qsub -a <run-after> ...
```

A job accrues `eligible_time` when it is blocked by a lack of resources, or by anything not qualifying as `ineligible_time` or `run_time`. A job's `eligible_time` will only increase during the life of the job, so if the job is requeued, its `eligible_time` is preserved, not set to zero. The job's `eligible_time` is not recalculated when a job is `qmove`d or moved due to peer scheduling.

For information on project, user, and group limits, see [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

The kind of time a job is accruing is sampled periodically, with a granularity of seconds.

A job's `eligible_time` attribute can be viewed via `qstat -f`.

4.9.13.3 Configuring Eligible Wait Time

To enable using eligible time as the job's wait time, set the `eligible_time_enable` server attribute to *True*.

4.9.13.4 How Eligible Wait Time Is Used

- When a job is requeued, for example being checkpointed and aborted or preempted, its accumulated queue waiting time depends on how that time is calculated:
 - If you are using eligible time, the accumulated waiting time is preserved
 - If you are not using eligible time, the accumulated waiting time is lost

See [section 8.3, “Checkpoint and Restart”, on page 387](#) and [section 4.9.33, “Using Preemption”, on page 180](#).

4.9.13.5 Altering Eligible Time

A Manager or Operator can set the value for a job's `eligible_time` attribute using the `qalter` command, for example:

```
qalter -Weligible_time=<time> <job ID>
```


4.9.13.6 Attributes Affecting Eligible Time

eligible_time_enable

Server attribute. Enables accumulation of eligible time for jobs.

On an upgrade from versions of PBS prior to 9.1 or on a fresh install, `eligible_time_enable` is set to *False* by default.

When `eligible_time_enable` is set to *False*, PBS does not track `eligible_time`. Whether `eligible_time` continues to accrue for a job or not is undefined. The output of `qstat -f` does not include `eligible_time` for any job. Accounting logs do not show `eligible_time` for any job submitted before or after turning `eligible_time_enable` off. Log messages do not include accrual messages for any job submitted before or after turning `eligible_time_enable` off. If the scheduling formula includes `eligible_time`, `eligible_time` evaluates to 0 for all jobs.

When `eligible_time_enable` is changed from *False* to *True*, jobs accrue `eligible_time` or `ineligible_time` or `run_time` as appropriate. Changing the value of `eligible_time_enable` does not change the behavior of an active scheduling cycle.

accrue_type

Job attribute. Indicates what kind of time the job is accruing.

Table 4-7: The `accrue_type` Job Attribute

Type	Numeric Representation	Type
JOB_INITIAL	0	initial_time
JOB_INELIGIBLE	1	ineligible_time
JOB_ELIGIBLE	2	eligible_time
JOB_RUNNING	3	run_time
JOB_EXIT	4	exit_time

The job's `accrue_type` attribute is visible via `qstat` only by Manager, and is set only by the server.

eligible_time

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by `ineligible_time`. For a job currently accruing `eligible_time`, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f` by job owner, Manager and Operator. Settable by Operator or Manager.

4.9.13.7 Logging

The server prints a log message every time a job changes its `accrue_type`, with both the new `accrue_type` and the old `accrue_type`. These are logged at the 0x0400 event class.

Server logs for this feature display the following information:

- Time accrued between samples
- The type of time in the previous sample, which is one of initial time, run time, eligible time or ineligible time
- The next type of time to be accrued, which is one of run time, eligible time or ineligible time
- The eligible time accrued by the job, if any, until the current sample

Example:

```
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 0 secs of initial_time, new
accrue_type=eligible_time, eligible_time=00:00:00
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1821 secs of eligible_time, new
accrue_type=ineligible_time, eligible_time=01:20:22
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 2003 secs of ineligible_time, new
accrue_type=eligible_time, eligible_time=01:20:22
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 61 secs of eligible_time, new
accrue_type=run_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 100 secs of run_time, new
accrue_type=ineligible_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 33 secs of ineligible_time, new
accrue_type=eligible_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 122 secs of eligible_time, new
accrue_type=run_time, eligible_time=01:23:25
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1210 secs of run_time, new
accrue_type=exiting, eligible_time=01:23:25
```

The example shows the following changes in time accrual:

- initial to eligible
- eligible to ineligible
- ineligible to eligible
- eligible to running
- running to ineligible
- ineligible to eligible
- eligible to running
- running to exiting

The server also logs the change in accrual when the job's `eligible_time` attribute is altered using `qalter`. For example, if the job's previous eligible time was 123 seconds, and it has been altered to be 1 hour and 1 minute:

```
Accrue type is eligible_time, previous accrue type was eligible_time for 123 secs, due to qalter
total eligible_time=01:01:00
```

4.9.13.8 Accounting

Each job's `eligible_time` attribute is included in the "E" and "R" records in the PBS accounting logs. See [section 12.4, "Types of Accounting Log Records", on page 532](#).

Example:

```
08/07/2007 19:34:06;E;182.Host1;user=user1 group=user1 jobname=STDIN queue=workq ctime=1186494765
qtime=1186494765 etime=1186494765 start=1186494767 exec_host=Host1/0
exec_vnode=(Host1:ncpus=1) Resource_List.ncpus=1 Resource_List.nodect=1
Resource_List.place=pack Resource_List.select=1:ncpus=1 session=4656 end=1186495446
Exit_status=-12 resources_used.cput=0 resources_used.cput=00:00:00
resources_used.mem=3072kb resources_used.ncpus=1 resources_used.vmem=13356kb
resources_used.walltime=00:11:21 eligible_time=00:10:00
```

4.9.13.9 Caveats for Eligible Time

- A job that is dependent on another job can accrue eligible time only after the job on which it depends has finished.
- The action of a hook may affect a job's eligible time. See ["Effect of Hooks on Job Eligible Time" on page 79 in the PBS Professional Hooks Guide](#).
- A subjob that is not running because its array job has hit the `max_run_subjobs` limit accrues eligible time as long as no other limits have been hit.

4.9.14 Sorting Jobs by Entity Shares (Was Strict Priority)

You can sort jobs according to how much of the fairshare tree is allocated to the entity that owns the job. The fairshare percentages in the fairshare tree describe each entity's share. Using entity shares is sorting jobs on a key, using the `fairshare_perc` option to the `job_sort_key` scheduler parameter.

Using entity shares, the jobs from an entity with greater allocation in the fairshare tree run before the jobs with a smaller allocation.

4.9.14.1 Configuring Entity Shares

To configure entity shares, do the following:

- Define fairshare tree entity allocation in `<sched_priv directory>/resource_group`. See [section 4.9.19, "Using Fairshare", on page 139](#). You can use a simple fairshare tree, where every entity's `parent_group` is `root`.
 - Give each entity shares according to desired priority, with higher-priority entities getting larger allocations.
 - Set the `unknown_shares` scheduler parameter to `1`. This causes any entity not in your list of approved entities to have a tiny allocation, and the lowest priority.

For example:

<code>usr1</code>	<code>60</code>	<code>root</code>	<code>5</code>
<code>usr2</code>	<code>61</code>	<code>root</code>	<code>15</code>
<code>usr3</code>	<code>62</code>	<code>root</code>	<code>15</code>
<code>usr4</code>	<code>63</code>	<code>root</code>	<code>10</code>
<code>usr5</code>	<code>64</code>	<code>root</code>	<code>25</code>
<code>usr6</code>	<code>65</code>	<code>root</code>	<code>30</code>

- Set `fairshare_perc` as the option to `job_sort_key`, for example:
`job_sort_key: "fairshare_perc HIGH all"`

4.9.14.2 Viewing Entity Shares

When you are root, you can use the `pbsfs` command to view the fairshare tree allocations.

4.9.15 Estimating Job Start Time

PBS can use a built-in hook called `PBS_est` that runs the job start time estimator, to estimate when jobs will run, and which vnodes each job will use. PBS estimates job start times and vnodes for all jobs using an asynchronous process, not the PBS server, scheduler, or MoM daemons. This estimator process is started by the `PBS_est` hook, whose default interval is 120 seconds. By default, the `PBS_est` hook is disabled.

Jobs have an attribute called `estimated` for reporting estimated start time and estimated vnodes. This attribute reports the values of two read-only built-in resources, `start_time` and `exec_vnode`. Each job's estimated start time is reported in `estimated.start_time`, and its estimated vnodes are reported in `estimated.exec_vnode`.

PBS automatically sets the value of each job's `estimated.start_time` value to the estimated start time for each job.

4.9.15.1 Configuring Start Time Estimation

When a scheduler is backfilling around top jobs, it estimates the start times and `exec_vnode` for those jobs being backfilled around. By default, `PBS_est` is disabled. If you want `PBS_est` to estimate start times and `exec_vnode` for all jobs, enable it:

- Enable the built-in `PBS_est` hook:

```
qmgr -c "set pbshook PBS_est enabled = true"
```

The default frequency for `PBS_est` is 120 seconds. You can set the frequency:

```
qmgr -c "set pbshook PBS_est freq = <interval in seconds>"
```

You set the number of jobs to be backfilled around by setting the server and/or queue `backfill_depth` attribute to the desired number. See [section 4.9.3, “Using Backfilling”, on page 109](#).

Example 4-1: To estimate start times for the top 5 jobs every scheduling cycle, and for all jobs every 3000 seconds:

```
qmgr -c 'set server backfill_depth=5'
qmgr -c 'set pbshook PBS_est enabled = true'
qmgr -c 'set pbshook PBS_est freq = 3000'
```

At each interval, the `PBS_est` hook checks whether the estimator process is running. If the estimator process is running when the `PBS_est` hook hits an interval and performs this check, the `PBS_est` hook does not stop the estimator process or start a new one. It allows the estimator process to finish running. If the estimator process is not running when the `PBS_est` hook hits an interval, the `PBS_est` hook starts a new estimator process.

4.9.15.2 Controlling User Access to Start Times and Vnode List

4.9.15.2.i Making Start Time or Vnodes Invisible

You can make job estimated start times and vnodes invisible to unprivileged users by adding resource permission flags to the `start_time` or `exec_vnode` resources. To do this, use `qmgr` to add the resource, and include the `i` flag, in the same way you would for a custom resource being made invisible.

Example of making `start_time` and `exec_vnode` invisible to users:

```
qmgr -c 'set resource start_time flag=i'
qmgr -c 'set resource exec_vnode flag=i'
```

You can always make the start time and vnodes visible again to unprivileged users by removing the flags via `qmgr`.

See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).

4.9.15.2.ii Allowing Users to See Only Their Own Job Start Times

If you want users to be able to see the start times for their own jobs, but not those of other users, set the server's `query_other_jobs` attribute to `False`, and do not set the `i` or `r` permission flags. Setting the server's `query_other_jobs` attribute to `False` prevents a user from seeing anything about other users' jobs.

4.9.15.3 Attributes and Parameters Affecting Job Start Time Estimation

[backfill](#)

Server attribute

[backfill_depth](#)

Server attribute

[backfill_depth](#)

Queue attribute

[enabled](#)

Hook attribute

[estimated](#)

Job attribute

[freq](#)

Hook attribute.

[strict_ordering](#)

Scheduler parameter

4.9.15.4 Viewing Estimated Start Times

You can view the estimated start times and vnodes of jobs using the `qstat` command. If you use the `-T` option to `qstat` when viewing job information, the *Est Start Time* field is displayed. Running jobs are shown above queued jobs.

See [“qstat” on page 200 of the PBS Professional Reference Guide](#).

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

`qstat -T`

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Est Start Time
5.host1	user1	workq	foojob	12345	1	1	128mb	00:10	R	--
9.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	11:30
10.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	Tu 15
7.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	Jul
8.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	2010
11.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	>5yrs
13.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	--

4.9.15.5 Selecting Jobs By Estimated Start Time

You can use the `qselect` command to select jobs according to their start times by using the `-t` suboption to the `-t` option. This selects jobs according to the value of the `estimated.start_time` attribute. See [“qsig” on page 195 of the PBS Professional Reference Guide](#).

4.9.15.6 Logging

Whenever a scheduler estimates the start time of a job, it logs the start time. A scheduler does not log the estimated `exec_vnode` of a job.

4.9.15.7 Caveats and Advice

- The `estimated.start_time` of a job array is the time calculated for the first queued subjob only.
- Cached estimated start times are only as fresh as the last time PBS calculated them. This should be taken into account when setting the values of the PBS_est hook's `freq` attribute and `backfill_depth`.
- The frequency of calculating start times is a trade-off between having more current start time information and using fewer computing cycles for non-job work. The background task of calculating start times can be computationally intensive. This should be taken into account when setting the value of the PBS_est hook's `freq` attribute. Depending on the size of your partition or site, it is probably a good idea not to set it to less than 10 minutes.
- The best value for the PBS_est hook's `freq` attribute is workload dependent, but we recommend setting it to two hours as a starting point.
- If your partition or site has short scheduling cycles of a few minutes, and can use backfilling and strict ordering, you can have the start times for all jobs calculated at each scheduling cycle. To do this, set `backfill_depth` to a value greater than the number of jobs the partition or site will ever have, and do not set the PBS_est hook's `freq` attribute.
- We recommend setting `backfill_depth` to a value that is less than 100.
- The process of computing the estimated start time for jobs is not instantaneous.
- Note that setting `backfill_depth` changes your scheduling policy. See [section 4.9.3, “Using Backfilling”, on page 109](#).

4.9.16 Calculating Job Execution Priority

When a scheduler examines jobs, either at the whole partition or complex or within a queue, it gives each job an execution priority, and then uses this job execution priority to select which job(s) to run. Job execution priority is mostly independent of job preemption priority. We discuss only job execution priority in this section.

Some of a scheduler's policy for determining job execution priority is built into PBS, but you can specify how execution priority is determined for most of the policy.

First, a scheduler divides queued jobs into classes. Then it sorts the jobs within each class.

4.9.16.1 Dividing Jobs Into Classes

PBS groups all jobs into classes, and handles one class at a time. There are special classes that supersede queue order, meaning that whether or not queues are being examined separately, the jobs in each of those classes are handled before a scheduler takes queues into account. Those jobs are not ordered according to which queue they reside in. For example, all *Express* jobs are handled as a group. PBS has one non-special class called *Normal* for all non-special jobs. This class typically contains most PBS jobs. Queue order is imposed on this class, meaning that queue priority affects job execution order if queues are being handled separately.

Job execution classes have a built-in order of precedence. All jobs in the highest class are considered before any jobs in the next class, and so on. Classes are listed in the following table, highest first:

Table 4-8: Job Execution Classes

Class	Description	Sort Applied Within Class
Reservation	Jobs submitted to an advance, standing, or job-specific reservation	Formula, job sort key, submission time
Express	All jobs with preemption priority higher than normal jobs. Preemption priority is defined in scheduler's <code>preempt_prio</code> attribute. Jobs are sorted into this class only when preemption is enabled. See section 4.9.33, “Using Preemption”, on page 180 .	First by preemption priority, then by preemption time, then by formula, then fairshare, then job sort key, followed by job submission time
Preempted	All jobs that have been preempted. See section 4.9.33, “Using Preemption”, on page 180 .	First by preemption time, then by formula, then fairshare, then job sort key, followed by job submission time
Normal	Jobs that do not belong in any of the special classes	Queue order, if it exists, then formula, then fairshare, then job sort key, followed by job submission time

4.9.16.2 Selecting Job Execution Class

A scheduler places each job in the highest-priority class into which the job can fit. So, for example, if a job is both in a reservation and is preempted, the job is placed in the **Reservation** class.

4.9.16.3 Sorting Jobs Within Classes

Jobs within each class are sorted according to rules specific to each class. The sorting applied to each class is listed in [Table 4-8, “Job Execution Classes,” on page 137](#).

- The **Reservation** class is made up of all jobs in reservations.
 - The Reservation class is sorted within each reservation.
 - The first sort is according to the formula or `job_sort_key`, depending on which is defined.
 - The second sort key is submission time.
- The **Express** class is made up of all the jobs that have a higher priority than "normal_jobs" in the `preempt_prio` scheduler attribute.
 - The Express class is sorted first by applying the rules for preemption priority you set in a scheduler's `preempt_prio` attribute, making preemption priority the first sort key.
 - The second sort key is the time the job was preempted (if that happened), with the earliest-preempted job having the highest priority (in this sort).
 - The third sort key is the formula, fairshare, or `job_sort_key`, depending on which is defined.
 - The fourth sort key is job submission time.

Jobs are sorted into this class only when preemption is enabled. See [section 4.9.33, “Using Preemption”, on page 180](#). Please note that execution priority classes are distinct from preemption levels, and are used for different purposes.

For example, if `preempt_prio` is the following:

```
preempt_prio: "express_queue, normal_jobs"
```

The **Express** class contains all jobs that have preemption priority that is greater than that of normal jobs. In this example, the **Express** class is prioritized with top priority for express queue jobs, followed by normal jobs.

- The **Preempted** class is made up of all preempted jobs.
 - The first sort key is the time the job was preempted, with the earliest-preempted job having the highest priority (in this sort).
 - The second sort key is the formula, fairshare, or `job_sort_key`, depending on which is defined.
 - The third sort key is job submission time.

When you set the `sched_preempt_enforce_resumption` scheduler attribute and the `strict_ordering` scheduler parameter to *True*, a scheduler tries harder to run preempted jobs. By default the attribute is *False*, and in each scheduling cycle, if a top job cannot run now, a scheduler moves on to the next top job and tries to run it. When the attribute and the parameter are *True*, a scheduler treats the job like a top job: it makes sure that no lower-priority job will delay this job, and it backfills around the job.

- The **Normal** class is for any jobs that don't fall into any of the other classes. Most jobs are in this class.
 - If queue ordering exists (there are multiple queues, and queues have different priorities set, and `round_robin` or `by_queue` is *True*), jobs are sorted first by queue order.
 - If defined, the formula, fairshare, or job sort key is the second sort key.
 - The third sort key is job submission time.

4.9.16.3.i Precedence of Sort Method Used Within Class

If the formula is defined, it overrides fairshare and the job sort key. If fairshare is defined, it overrides the job sort key. If none are defined, jobs are ordered by their arrival time in the queue.

For the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

For fairshare, see [section 4.9.19, “Using Fairshare”, on page 139](#).

For sorting jobs on a key, see [section 4.9.45, “Sorting Jobs on a Key”, on page 220](#).

4.9.16.4 Execution Priority Caveats

- Limits are not taken into account when prioritizing jobs for execution. Limits are checked only after setting priority, when selecting a job to run. The only exception is in the **Express** class, where soft limits may be taken into account, because execution priority for **Express** class jobs is calculated using preemption priority. For details, see [section 4.9.33, “Using Preemption”, on page 180](#).
- When you issue `qrun <job ID>`, without the `-H` option, the selected job is the only job considered to run during that scheduling cycle.
- Jobs are sorted into the **Express** class only when preemption is enabled.

4.9.17 Calendaring Jobs

In each scheduling cycle, PBS runs through its list of jobs in the order that you have defined. The backfill depth determines the number of top jobs; these are the highest-priority jobs in its current list. When strict priority and backfilling are in force, and PBS cannot run a top job right now, PBS holds a spot open for that job: PBS finds a future spot in the calendar that fits the job's needs, and doesn't schedule any other jobs that would interfere with the top job.

PBS rebuilds the calendar with each scheduling cycle.

4.9.17.1 Making Jobs Ineligible to be Top Jobs

By default, a job is eligible to be a top job, meaning that PBS holds resources for it if it cannot run right now (the `topjob_ineligible` job attribute defaults to *False*). If you set the value of a job's `topjob_ineligible` attribute to *True*, that job cannot become a top job, and PBS does not hold a spot open for that job if it cannot run the job right now. Having the highest priority is not the same as being a top job.

4.9.17.1.i Caveats for Making Jobs Ineligible to be Top Jobs

When `sched_preempt_enforce_resumption` is set to *True*, all preempted jobs become top jobs, regardless of their setting for `topjob_ineligible`.

4.9.18 Express Queues

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but the cutoff can be set using the `preempt_queue_prio` scheduler attribute. For information on configuring express queues, see [section 2.3.5.3.i, “Express Queues”, on page 27](#).

You can use express queues as tools to manage job execution and preemption priority.

- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in a scheduler, see [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- You can set up preemption levels that include jobs in express queues. For information on preemption, see [section 4.9.33, “Using Preemption”, on page 180](#).

The term "express" is also used in calculating execution priority to mean all jobs that have a preemption level greater than that of the `normal_jobs` level.

4.9.19 Using Fairshare

Fairshare provides a way to enforce a partition's or site's resource usage policy. It is a method for ordering the start times of jobs based on two things: how a site's resources are apportioned, and the resource usage history of partition or site members. Fairshare ensures that jobs are run in the order of how deserving they are. A scheduler performs the fairshare calculations each scheduling cycle. If fairshare is enabled, all jobs have fairshare applied to them and there is no exemption from fairshare.

You can employ basic fairshare behavior, or a policy of the desired complexity.

The `fair_share` parameter is a primetime option, meaning that you can configure it for either primetime or non-primetime, or you can specify it for all of the time. You cannot configure different behaviors for fairshare during primetime and non-primetime.

You can use fairshare information calculated by PBS in the job sorting formula. See [section 4.9.19.6, “Computing Fairshare Values”, on page 145](#) and [section 4.9.21.7, “Using Fairshare in the Formula”, on page 154](#).

4.9.19.1 One Fairshare System Per Scheduler

Each scheduler runs one fairshare system. Each fairshare system is independent of any others. If you are running only the default scheduler (no multischeds), it runs one fairshare system for the entire site. If you are using one or more multischeds, each of the multischeds runs its own fairshare system, and the default scheduler runs one fairshare system.

Each scheduler has its own usage, `resource_group`, etc., fairshare files in its `sched_priv` directory, and its own `sched_config` configuration file.

The `pbsfs` command operates on one scheduler's fairshare database at a time. You specify which scheduler's database to operate on using the `pbsfs -I <scheduler name>` option.

In the following sections on fairshare, we describe the behavior for any single scheduler.

4.9.19.2 Outline of How Fairshare Works

The owner of a PBS job can be defined for fairshare purposes to be a user, a group, the job's accounting string, etc. For example, you can define owners to be groups, and can explicitly set each group's relationship to all the other groups by using the tree structure. If you don't explicitly list an owner, it will fall into the "unknown" catchall. All owners in "unknown" get the same resource allotment. You can define one group to be part of a larger department.

You specify which resources to track and how you want usage to be calculated. So if you defined job owners to be groups, then only the usage of groups is considered. PBS tries to ensure that each owner gets the amount of resources that you have set for it.

4.9.19.3 Enabling Basic Fairshare

If the default fairshare behavior is enabled, PBS enforces basic fairshare rules where all users with queued jobs will get an equal share of CPU time. The root vertex of the tree will have one child, the unknown vertex. All users will be put under the unknown vertex, and appear as children of the unknown vertex.

Enable basic fairshare by doing the following:

- In the scheduler's `sched_config` file, set the scheduler configuration parameter `fair_share` to *True*
- Uncomment the `unknown_shares` setting so that it is set to `unknown_shares: 10`
- Specify how you want fairshare to work with primetime and non-primetime. If you want separate behavior for primetime and non-primetime, list the `fair_share` parameter twice, once for each time slot. The default is both. For example:

```
fair_share True prime
fair_share False non_prime
```

Note that a variant of basic fairshare has all users listed in the tree as children of root. Each user can be assigned a different number of shares.

4.9.19.4 Configuring the Fairshare Tree

Fairshare uses a tree structure, where each vertex in the tree represents some set of job owners and is assigned usage *shares*. Shares are used to apportion the partition's or site's resources. The default tree always has a root vertex and an *unknown* vertex. The default behavior of fairshare is to give all users the same amount of the resource being tracked. In order to apportion a partition's or site's resources according to a policy other than equal shares for each user, you create a fairshare tree to reflect that policy. To do this, you edit the `resource_group` file in the scheduler's `sched_priv` directory, which describes that scheduler's fairshare tree.

To configure non-default fairshare, set up a hierarchical tree structure made up of interior vertices and leaves. Interior vertices are *departments*, which can contain both departments and leaves. Leaves are for *fairshare entities*, defined by setting `fairshare_entity` to one of the following: *euser*, *egroup*, *egroup:euser*, *Account_Name*, or *queue*. Apportioning of resources for the partition or site is among these entities. These entities' usage of the designated resource is used in determining the start times of the jobs associated with them. All fairshare entities must be the same type. If you wish to have a user appear in more than one department, you can use *egroup:euser* to distinguish between that user's different resource allotments. Note that in the `resource_group` file and in the output of `pbsfs`, interior (non-leaf) vertices are referred to as "groups", and exterior (leaf) vertices are referred to as "users".

Table 4-9: Using Fairshare Entities

Keyword	Fairshare Entities	Purpose
<i>euser</i>	Username	Individual users are allotted shares of the resource being tracked. Each username may only appear once, regardless of group.
<i>egroup</i>	OS group name	Groups as a whole are allotted shares of the resource being tracked.
<i>egroup:euser</i>	Combinations of username and group name	Useful when a user is a member of more than one group, and needs to use a different allotment in each group.
<i>Account_Name</i>	Account IDs	Shares are allotted by account string (<i>Account_Name</i> job attribute).
<i>queue</i>	Queues	Shares are allotted between queues.

4.9.19.4.i Allotting Shares in the Tree

You assign shares to each vertex in the tree. The actual number of shares given to a vertex or assigned in the tree is not important. What is important is the ratio of shares among each set of sibling vertices. Competition for resources is between siblings only. The sibling with the most shares gets the most resources.

4.9.19.4.ii Shares Among Unknown Entities

The root vertex always has a child called `unknown`. Any entity not listed in the scheduler's `resource_group` file will be made a child of `unknown`, designating the entity as unknown. The shares used by unknown entities are controlled by two parameters in the scheduler's `sched_config` file: `unknown_shares` and `fairshare_enforce_no_shares`.

The parameter `unknown_shares` controls how many shares are assigned to the unknown vertex. The shipped `sched_config` file contains this line:

```
#unknown_shares 10
```

If you leave `unknown_shares` commented out, the `unknown` vertex will have 0 shares. If you simply remove the "#", the `unknown` vertex's shares default to 10. The children of the `unknown` vertex have equal amounts of the shares assigned to the `unknown` vertex.

The parameter `fairshare_enforce_no_shares` controls whether an entity without any shares can run jobs. If `fairshare_enforce_no_shares` is *True*, entities without shares cannot run jobs. If it is set to *False*, entities without any shares can run jobs, but only when no other entities' jobs are available to run.

4.9.19.4.iii Format for Describing the Tree

The file describing the fairshare tree contains four columns to describe the vertices in the tree. Here is the format for the columns:

```
<Vertex name> <vertex fairshare ID> <parent of vertex> <#shares>
```

The columns are for a vertex's name, its fairshare ID, the name of its parent vertex, and the number of shares assigned to this (not the parent) vertex. Vertex names and IDs must be unique. Vertex IDs are integers. The top row in `resource_group` contains information for the first vertex, rather than column labels.

Neither the root vertex nor the unknown vertex is described in the `resource_group` file. They are always added automatically. Parent vertices must be listed before their children.

For example, we have a tree with two top-level departments, Math and Phys. Under Math are the users Bob and Tom as well as the department Applied. Under Applied are the users Mary and Sally. Under Phys are the users John and Joe. Our `<sched_priv directory>/resource_group` looks like this:

Math	100	root	30
Phys	200	root	20
Applied	110	Math	25
Bob	101	Math	15
Tom	102	Math	10
Mary	111	Applied	1
Sally	112	Applied	2
John	201	Phys	2
Joe	202	Phys	2

If you wish to use `egroup:euser` as your entity, and Bob to be in two groups `pbsgroup1` and `pbsgroup2`, and Tom to be in two groups `pbsgroup2` and `pbsgroup3`:

Math	100	root	30
Phys	200	root	20
Applied	110	Math	20
pbsgroup1:Bob	101	Phys	20
pbsgroup2:Bob	102	Math	20
pbsgroup2:Tom	103	Math	10
pbsgroup3:Tom	104	Applied	10

When a user submits a job using `-Wgroup_list=<group>`, the job's `egroup` will be `<group>`. For example, user Bob is in `pbsgroup1` and `pbsgroup2`. Bob uses `"qsub -Wgroup_list= pbsgroup1"` to submit a job that will be charged to `pbsgroup1`, and `"qsub -Wgroup_list=pbsgroup2"` to submit a job that will be charged to `pbsgroup2`.

The first and third fields are alphanumeric. The second and fourth fields are numeric. Fields can be separated by spaces and tabs.

4.9.19.4.iv Moving Entities within Fairshare Tree

To move an entity within the fairshare tree, change its parent:

1. Edit `<sched_priv directory>/resource_group`. Change the parent (column 3) to the desired parent
2. HUP or restart the scheduler

4.9.19.4.v Removing Entities from Fairshare Tree

You may want to remove an entity from the fairshare tree, either because they no longer run jobs, or because you don't want them to have their own place in the tree. When an entity that is not in the fairshare tree runs a job, their past and future usage, including that for jobs running while you remove the entity, shows up in the Unknown group. To remove an entity from the fairshare tree:

1. Edit the `resource_group` file to remove the entity line
2. HUP or restart the scheduler

If you do not want an entity's usage to show up in the Unknown group, use `pbsfs -e [-I <multisched name>]` to remove the usage. If you are working on a partition managed by a multisched, you must specify the name of the multisched:

1. Prevent jobs from being scheduled
2. Run `pbsfs -e [-I <multisched name>]`
3. Resume scheduling jobs

If you have removed a user from the PBS partition or complex and don't want their usage to show up any more:

1. Prevent jobs from being scheduled
2. Edit the `resource_group` file
3. Run `pbsfs -e [-I <multisched name>]`
4. Resume scheduling jobs

4.9.19.5 Resource Usage for Fairshare

4.9.19.5.i Tracking Resource Usage

You choose which resources to track and how to compute the usage by setting the `fairshare_usage_res` scheduler configuration parameter in the `sched_config` file to the fairshare resource formula you want. This parameter can contain the following:

- Built-in and custom job resources

When you use a resource in the fairshare resource formula, if a value exists for `resources_used.<resource name>`, this value is used in the fairshare resource formula. Otherwise, the value is taken from `Resource_List.<resource name>`.

- Mathematical operators

You can use standard Python operators and the operators in the Python math module.

The default for the tracked resource is `cput` (CPU time).

4.9.19.5.ii Adding Usage

An entity's usage always starts at 1. Resource usage tracking begins when a scheduler is started. Each scheduler cycle, the scheduler adds the usage increment between this cycle and the previous cycle to its sum for the entity.

A static resource does not change its usage from one cycle to the next. If you use a static resource such as `ncpus`, the amount being tracked will not change during the lifetime of the job; it will only be added once when the job starts.

Note that if a job ends between two scheduling cycles, its resource usage for the time between the previous scheduling cycle and the end of the job will not be recorded. A scheduler's default cycle interval is 10 minutes. The scheduling cycle can be adjusted via the `qmgr` command. Use

```
Qmgr: set sched [sched name] scheduler_iteration=<new value>
```

If the fairshare resource formula in `fairshare_usage_res` evaluates to a negative number, PBS uses zero instead. So there is no way to accumulate negative usage.

4.9.19.5.iii Decaying Usage

Each entity's usage is *decayed*, or reduced periodically, at the interval set in the `fairshare_decay_time` parameter in the `sched_config` file. This interval defaults to 24 hours.

The amount by which usage is decayed is set in the `fairshare_decay_factor` scheduler parameter.

An entity with a lot of current or recent usage will have low priority for starting jobs, but if the entity cuts resource usage, its priority will go back up after a few decay cycles.

4.9.19.5.iv Setting Decay Interval and Factor

You set the interval at which usage is decayed by setting the `fairshare_decay_time` scheduler parameter to the desired time interval. The default value for this interval is 24 hours. For example, to set this interval to 14 hours and 23 minutes, put this line in the `sched_config` file:

```
fairshare_decay_time: 14:23:00
```

You set the decay factor by setting the `fairshare_decay_factor` scheduler parameter to the desired multiplier for usage. At each decay interval, the usage is multiplied by the decay factor. This attribute is a float whose value must be between 0 and 1. The value must be greater than 0 and less than 1. The default value for this multiplier is 0.5. For example, to set this multiplier to 70 percent, put this line in `sched_config`:

```
fairshare_decay_factor: .7
```

4.9.19.5.v Examples of Setting Fairshare Usage

To use CPU time as the resource to be tracked, put this line in `sched_config`:

```
fairshare_usage_res: cput
```

To use `ncpus` multiplied by `walltime` as the resource to be tracked, put this line in `sched_config`:

```
fairshare_usage_res: ncpus*walltime
```

An example of a more complex formula:

```
fairshare_usage_res: "ncpus*pow(walltime,.25)*fs_factor"
```

4.9.19.5.vi Fairshare Resource Advice

We recommend including a time-based resource in the fairshare formula so that usage will grow over time.

4.9.19.5.vii Viewing and Managing Fairshare Usage Data

The `pbsfs` command provides a command-line tool for viewing and managing some fairshare data. You can display the data as a tree, a table, or by entity. You can print all information about an entity, or set an entity's usage to a new value. You can force an immediate decay of all the usage values in the tree. You can compare two fairshare entities. You can also remove all entities from the unknown department. This makes the tree easier to read. The tree can become unwieldy because entities not listed in the `resource_group` file all land in the unknown group. See [“pbsfs” on page 32 of the PBS Professional Reference Guide](#).

To change fairshare resource usage data, do the following:

1. Stop scheduling:
`Qmgr: set sched [<sched name>] scheduling = false`
2. Wait until the current scheduling cycle finishes. Check the scheduler's log.
3. Trim the fairshare tree:
`pbsfs -e [-I <multisched name>]`
4. Set each entity's usage to one (cannot be zero). For each leaf entity:
`pbsfs -s <entity name> 1 [-I <multisched name>]`
5. Start scheduling:
`Qmgr: set sched [<sched name>] scheduling = true`

The fairshare usage data is written to the file `usage` file at each scheduling cycle. The usage data is always up to date.

For more information on using the `pbsfs` command, see [“pbsfs” on page 32 of the PBS Professional Reference Guide](#)

4.9.19.6 Computing Fairshare Values

PBS provides `fairshare_perc`, `fairshare_tree_usage`, and `fairshare_factor` as terms to use in the job sorting formula. You can also use `fairshare_perc` as an argument to the `job_sort_key` scheduler parameter.

4.9.19.6.i Computing Target Usage for Each Vertex (`fairshare_perc`)

How much resource each entity should use is its target usage, computed in `fairshare_perc`. Target usage is the percentage of the shares in the tree allotted to the entity. Target usage does not take history into account.

A vertex's portion of all the shares in the tree is its `fairshare_perc`. This is computed for all of the vertices in the tree. Since the leaves of the tree represent the entities among which resources are to be shared, their `fairshare_perc` sums to 100 percent. Only the leaf nodes sum to 100%; if all of the nodes were summed, the result would be greater than 100%. Only the leaf nodes of the tree are fairshare entities.

A scheduler computes the `fairshare_perc` for the vertices this way:

First, it gives the root of the tree a `fairshare_perc` of 100 percent. It proceeds down the tree, finding the `fairshare_perc` first for immediate children of root, then their children, ending with leaves.

1. For each internal vertex A:
 - sum the shares of its children;
2. For each child J of vertex A:
 - divide J's shares by the sum to normalize the shares;
 - multiply J's normalized shares by vertex A's `fairshare_perc` to find J's `fairshare_perc`.

The `fairshare_perc` value can be used in the job sorting formula and as an argument to the `job_sort_key` scheduler parameter.

4.9.19.6.ii Computing Effective Usage (`fairshare_tree_usage`)

An entity's effective usage is `fairshare_tree_usage`, and is a value between 0 and 1.

PBS calculates the value for `fairshare_tree_usage` this way:

For root's children:

$$\text{fairshare_tree_usage} = \text{percent total usage}$$

For entities below root's children:

$$\text{fairshare_tree_usage} = \text{entity's percent total usage} + ((\text{parent's effective usage} - \text{entity's percent total usage}) * \text{entity's relative percent of shares within sibling group})$$

where

$$\text{entity's percent total usage} = \text{entity's usage} / \text{all usage in partition or complex}$$

Summing effective usage for all leaves in the tree does not yield a useful number (such as 1).

4.9.19.6.iii Computing Relative Usage (`fairshare_factor`)

An entity's relative usage allows direct comparison between entities. Relative usage is `fairshare_factor`, and is a value between 0 and 1. A value of 0.5 means that an entity is using exactly its target usage. A higher value indicates less resource usage by the entity, meaning that the entity is more deserving. Calculated this way:

$$2^{-(\text{fairshare_tree_usage} / \text{entity's fairshare_perc})}$$

4.9.19.6.iv Example of Computing Fairshare Values

Example 4-2: The following fairshare tree shows shares and usage for two groups, each with two people:

Table 4-10: Example Fairshare Tree

Vertex		Shares	Actual Usage	% Total Usage	% Group Shares	Target Usage: fairshare_perc	Effective Usage: fairshare_tree_usage	Relative Usage: fairshare_factor
root			1200	100	1.0	1.0	1.0	1.0
	group1	40	200	0.1667	0.4	0.4	0.167	0.75
	Bob	50	100	0.0833	0.5	0.2	0.125	0.65
	Cathy	50	100	0.0833	0.5	0.2	0.125	0.65
	group2	60	1000	0.833	0.6	0.6	0.833	0.38
	Suzy	60	0	0	0	0.36	0.5	0.38
	Scott	40	1000	0.833	1	0.24	0.833	0.09

Comparing Suzy and Bob:

Bob:

Percent total usage: $100/1200 = 0.083$

Parent's effective usage: 0.1667

Bob's percentage of shares in group: Bob's 50 shares / (Bob's 50 shares+ Cathy's 50 shares) = 0.5

Bob's effective usage: Bob's percent total usage: 0.0833 + (group1's effective usage: 0.1667 - Bob's percent total usage: 0.083) * 0.5 = 0.125

Relative usage formula: $2^{-(0.125/0.2)} = 0.648$

Suzy:

Percent total usage: $0/1200 = 0$

Parent's effective usage: $1000/1200 = 0.833$

Suzy's percentage of shares in group: Suzy's 60 shares / (Suzy's 60 shares + Scott's 40 shares) = 0.6

Suzy's effective usage: Suzy's percent total usage: 0 + (group2's usage: 0.833 - Suzy's usage: 0) * 0.6 = 0.5

Relative usage formula: $2^{-(0.5/0.36)}: 0.382$

Even though Suzy had a higher fairshare_perc than Bob and less usage than Bob, her relative usage (fairshare_factor) is quite a bit lower than his, because of the huge amount Scott used.

Output of `pbsfs -g`:

Our `pbsfs` output uses the same fairshare data as the previous example.

```
# ./pbsfs -g scott
fairshare entity: scott
Resgroup           : 11
cresgroup          : 15
Shares             : 40
Percentage         : 24.000000%
fairshare_tree_usage : 0.832973
usage              : 1000 (cput)
usage/perc         : 4167
Path from root:
TREEROOT   :    0      1201 / 1.000 = 1201
group2     :   11      1001 / 0.600 = 1668
scott      :   15      1000 / 0.240 = 4167
```

4.9.19.7 Choosing Which Job to Run

4.9.19.7.i Finding the Most Deserving Entity

The most deserving entity is found by starting at the root of the tree, comparing its immediate children, finding the most deserving, then looking among that vertex's children for the most deserving child. This continues until a leaf is found. In a set of siblings, the most deserving vertex will be the vertex with the lowest ratio of resource usage divided by `fairshare_perc`.

4.9.19.7.ii Sorting and Selecting Jobs to Run

The job to be run next is selected from the set of jobs belonging to the most deserving entity. The jobs belonging to the most deserving entity are sorted according to the methods a scheduler normally uses. This means that fairshare effectively becomes the primary sort key. If the most deserving job cannot run, then the next most is selected to run, and so forth. All of the most deserving entity's jobs are examined first, then those of the next most deserving entity, etcetera.

At each scheduling cycle, a scheduler attempts to run as many jobs as possible. It selects the most deserving job, runs it if it can, then recalculates to find the next most deserving job, runs it if it can, and so on.

When a scheduler starts a job, all of the job's requested usage is added to the sum for the owner of the job for one scheduling cycle. The following cycle, the job's usage is set to the actual usage that occurred between the first and second cycles. This prevents one entity from having all its jobs started and using up all of the resource in one scheduling cycle.

4.9.19.8 Files and Parameters Used in Fairshare

`sched_config`

File in the directory specified in the scheduler's `sched_priv` attribute

PBS uses the following parameters from `sched_config` to compute fairshare values:

Table 4-11: `sched_config` Parameters used in Fairshare

Parameter	Use
<code>fair_share</code>	[<i>True/False</i>] Enable or disable fairshare
<code>fairshare_usage_res</code>	Resource whose usage is to be tracked; default is <code>cput</code>
<code>fairshare_decay_factor</code>	Amount to decay usage at each decay interval
<code>fairshare_decay_time</code>	Decay interval; default is 24 hours
<code>unknown_shares</code>	Number of shares for unknown vertex; default 10, 0 if commented out
<code>fairshare_entity</code>	The kind of entity which is having fairshare applied to it. Leaves in the tree are this kind of entity. Default: <i>euser</i>
<code>fairshare_enforce_no_shares</code>	If an entity has no shares, this controls whether it can run jobs. T: an entity with no shares cannot run jobs. F: an entity with no shares can only run jobs when no other jobs are available to run. Default: <i>True</i>
<code>by_queue</code>	If on, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.

`resource_group`

File in the directory specified in the scheduler's `sched_priv` attribute

Contains the description of the fairshare tree.

`usage`

File in the directory specified in the scheduler's `sched_priv` attribute

Contains the usage database. Written by PBS. Do not edit. Written each scheduling cycle.

`scheduler_iteration`

Scheduler attribute. Specifies scheduler cycle frequency; default is 10 minutes.

Qmgr: `set sched <scheduler name> scheduler_iteration=<new value>`

`resources_used.<resource name>`

Job attribute. Contains resources used for tracking usage. Default is `cput`.

4.9.19.9 Ways to Use Fairshare

4.9.19.9.i Fairshare for Partition Or Complex or Within Queues

You can use fairshare to compare all jobs in the partition managed by a scheduler, or within each queue. Fairshare within a queue means that a scheduler examines the jobs in a queue, and compares them to each other, to determine which job to start next.

To use fairshare for the entire partition or complex, set the `by_queue` and `round_robin` scheduler configuration parameters to *False*.

To use fairshare within queues, set the `by_queue` scheduler parameter to *True*, and `round_robin` to *False*. If you want to examine queues in a particular order, prioritize the queues by setting each queue's `priority` attribute.

The scheduler configuration parameter `by_queue` in the `sched_config` file is set to *True* by default.

If `by_queue` is *True*, queues cannot be designated as fairshare entities.

4.9.19.9.ii Altering Fairshare According to Queue

You can introduce a fairshare factor that is different at each queue. To do this, create a custom floating point resource, and set each queue's `resources_default.<resource name>` to the desired value. Use this resource in the `fairshare_usage_res` computation. If you do not set this value at a queue, PBS uses zero for the value. To avoid having to set a value at multiple queues, you can set the server's `resources_default.<resource name>` to the default value for all queues where the value is unset. The server value is used only where the queue value is unset; where the queue value is set, the queue value takes precedence.

For example, to reduce the priority for jobs in the "expensive" queue by assigning them twice the usage of the jobs in `workq`:

- Define the resource:
`Qmgr: create resource fs_factor type = float, flag = i`
- Set the resource values:
`Qmgr: set server resources_default.fs_factor = 1`
`Qmgr: set queue workq resources_default.fs_factor = 0.3`
`Qmgr: set queue expensive resources_default.fs_factor = 0.6`
- Edit `sched_config`:
`fairshare_usage_res: "fs_factor*ncpus*walltime"`

4.9.19.9.iii Using Fairshare in Job Execution Priority

Jobs are sorted as specified by the formula in `job_sort_formula`, if it exists, then by fairshare, if it is enabled, or if neither of those is used, by `job_sort_key`. The job sorting formula can use the following calculated values: `fairshare_perc`, the percentage of the fairshare tree for this job's entity, `fairshare_tree_usage`, an entity's effective usage, and `fairshare_factor`, an entity's comparative usage. See [section 4.9.16, "Calculating Job Execution Priority", on page 136](#).

4.9.19.9.iv Using Fairshare in Job Preemption Priority

You can use the *fairshare* preemption level in determining job preemption priority. This level applies to jobs whose owners are over their fairshare allotment. See [section 4.9.33, "Using Preemption", on page 180](#).

4.9.19.10 Fairshare Restrictions

- Entity shares (strict priority):
If you enable entity shares (strict priority), you use the same fairshare tree that you would use for fairshare. Fairshare and entity shares (strict priority) are incompatible, so in order to use entity shares, you disable fairshare by setting `fair_share` to *False*. For how to configure entity shares, see [section 4.9.14, "Sorting Jobs by Entity Shares \(Was Strict Priority\)", on page 133](#).
- Requested jobs:
When a job is requested, it normally retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requested job started, this job's owner has gone over their fairshare limit, etc.
- With `strict_ordering` or backfilling:
We do not recommend using fairshare with `strict_ordering`, or with `strict_ordering` and backfilling. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group as the top job will be chosen as the filler job. The usage from the filler job will lower the priority of the most deserving, i.e. top, job. This could delay the execution of the top job.

However, if all of your leaf entities are children of root (the tree has only two levels), and all users tend to submit the same size jobs, results may be useful.

- With `fairshare_perc` option to `job_sort_key`:

Do not use fairshare when using the `fairshare_perc` option to `job_sort_key`. You can still use the value of `fairshare_perc` in the job sorting formula.

- Static resources:

Do not use static resources such as `ncpus` as the resource to track. A scheduler adds the incremental change in the tracked resource at each scheduling cycle, and a static resource will not change.

4.9.19.11 Fairshare Caveats and Advice

- The most deserving entity can change with every scheduling cycle, if each time a job is run, it changes usage sufficiently.
- Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot of how the jobs would run for the next n days. Hence fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.
- The `half_life` parameter is **deprecated** and has been replaced by the `fairshare_decay_time` parameter.
- Beware of overflow: PBS stores fairshare allocations in a signed integer (32-bit on Linux x86_64 platforms), and fairshare usage in a long (64-bit on Linux x86_64 platforms)

4.9.20 FIFO Scheduling

With FIFO scheduling, PBS runs jobs in the order in which they are submitted. You can use FIFO order for all of the jobs in your partition or complex, or you can go queue by queue, so that the jobs within each queue are considered in FIFO order.

4.9.20.1 Configuring Basic FIFO Scheduling

To configure basic FIFO scheduling, whether across all a scheduler's partition or queue by queue, set the following scheduler parameters and queue/server attribute to these values:

```
round_robin:           False  ALL
job_sort_key:          (commented out)
fair_share             False  ALL
backfill_depth:        0
job_sort_formula:      (unset)
```

4.9.20.2 FIFO for Entire Partition Or Complex

To configure FIFO across your entire partition or complex, follow the steps above and do one of the following:

- Use only one execution queue
- Set the `by_queue` scheduler parameter to *False*

4.9.20.3 Queue by Queue FIFO

To configure FIFO for each queue separately, first decide how you want queues to be selected. You can set the order in which PBS chooses queues from which to run jobs, or you can allow the queues to be selected in an undefined way. First configure this scheduler as in [Section 4.9.20.1, "Configuring Basic FIFO Scheduling"](#).

- To allow queues to be selected in an undefined way, set the `by_queue` scheduler parameter to *True*.
- To set the order in which queues are selected, do the following:
 - Specify a priority for each queue
 - Set the `by_queue` scheduler parameter to *True*

4.9.20.4 FIFO with Strict Ordering

If your jobs must run exactly in submission order, you can use strict ordering with FIFO scheduling. If you use strict ordering with FIFO scheduling, this means that when the job that is supposed to run next cannot run, no jobs can run. This can result in less throughput than you could otherwise achieve. To avoid that problem, you can use backfilling. See the following section.

To use strict ordering with FIFO scheduling, use the following scheduler parameter settings in `<sched_priv directory>/sched_config` and queue/server attribute settings:

```
strict_ordering:      True    ALL
round_robin:          False   ALL
job_sort_key:         (commented out)
fair_share            False   ALL
backfill_depth:       0
job_sort_formula:     (unset)
```

4.9.20.5 FIFO with Strict Ordering and Backfilling

If you want to run your jobs in submission order, except for backfilling around top jobs that are stuck, use the following:

```
strict_ordering:      True    ALL
round_robin:          False   ALL
job_sort_key:         (commented out)
fair_share            False   ALL
backfill_depth:       <depth>
job_sort_formula:     (unset)
```

4.9.21 Using a Formula for Computing Job Execution Priority

When PBS sorts jobs to determine execution priority, it starts with a coarse-granularity sort according to job class. You can choose to use a formula by which to sort jobs in a class relative to the other jobs in that class. The formula can only direct how jobs are sorted at the finest level of granularity. However, that is where most of the sorting work is done. See [section 4.9.16.3, "Sorting Jobs Within Classes", on page 137](#) for how the formula is used in setting job execution priority.

When a scheduler sorts jobs according to the formula, it computes an in-class priority for each job. This priority is the value produced by the formula. Jobs with a higher value get higher priority.

The priority computed by the formula can be used as part of the job execution priority. The priority computed by the formula and the job execution priority are both distinct from the **Priority** job attribute, which can be set by the job submitter. The administrator can choose to use the **Priority** job attribute as an input to the formula, calculation of execution and/or preemption priority, or job sorting, where the attribute is represented by the `job_priority` keyword. The **Priority** job attribute is not automatically changed when PBS computes the formula or job execution priority, although the administrator can use a hook to change the attribute. The

Only one formula is used to prioritize jobs. At each scheduling cycle, the formula is applied to all jobs, regardless of when they were submitted. If you change the formula, the new formula is applied to all jobs.

For example, if you submit some jobs, change the formula, then submit more jobs, the new formula is used for all of the jobs, during the next scheduling cycle.

You can set a job priority threshold so that jobs with priority at or below the specified value do not run. See [section 4.9.21.10, “Setting Minimum Job Priority Value for Job Execution”, on page 155](#).

You may find that the formula is most useful when you use it with custom resources inherited by or allocated to jobs. For example, you may want to route all jobs from a particular project to a queue where they inherit a specific value for a custom resource. Other jobs may end up at a different queue, where they inherit a different value, or they may inherit no value. You can then use this custom resource in the formula as a way to manage job priority. See [section 10.3, “Allocating Resources to Jobs”, on page 455](#), and [section 4.9.8, “Using Custom and Default Resources”, on page 116](#).

It may be helpful if these custom resources are invisible and unrequestable by users. See [section 4.9.21.12, “Examples of Using Resource Permissions in Job Sorting Formula”, on page 156](#).

4.9.21.1 When the Formula is Applied

Once you set `job_sort_formula` via `qmgr`, it takes effect with the following scheduling cycle.

Variables are evaluated at the start of the scheduling cycle.

4.9.21.2 Configuring the Job Sorting Formula

- Define the formula:

You specify the formula in the server's `job_sort_formula` attribute. To set the `job_sort_formula` attribute, use the `qmgr` command. When specifying the formula, be sure to follow the requirements for entering an attribute value via `qmgr`: strings containing whitespace, commas, or other special characters must be enclosed in single or double quotes. See [“Caveats and Restrictions for Setting Attribute and Resource Values” on page 162 of the PBS Professional Reference Guide](#). Format:

```
Qmgr: s s job_sort_formula = "<formula>"
```

- Optional: set a priority threshold. See [section 4.9.21.10, “Setting Minimum Job Priority Value for Job Execution”, on page 155](#)

4.9.21.3 Requirements for Creating Formula

The job sorting formula must be created at the server host.

Under Linux, root privilege is required in order to operate on the `job_sort_formula` server attribute.

4.9.21.4 Format of Formula

The formula must be valid Python, and must use Python syntax. The formula can be made up of any number of *expressions*, where expressions contain *terms* which are added, subtracted, multiplied, or divided. You can use parentheses, exponents, unary + and - operators, and the ternary operator (which must be Python). All operators use standard mathematical precedence. The formula can use standard Python mathematical operators and those in the Python math module.

The formula can be any length.

The range for the formula is defined by the IEEE floating point standard for a double.

4.9.21.5 Units in Formula

The variables you can use in the formula have different units. Make sure that some terms do not overpower others, by normalizing them where necessary. Resources like `ncpus` are integers, size resources like `mem` are in kb, so 1gb is 1048576kb, and time-based resources are in seconds (e.g. `walltime`). Therefore, if you want a formula that combines memory and `ncpus`, you'll have to account for the factor of 1024 difference in the units.

The following are the units for the supported built-in resources:

Table 4-12: Job Sorting Formula Units

Resource	Units	Example
Time resources	<i>Integer number of seconds</i>	300
Memory	<i>kb</i>	1gb => 1048576kb
<code>ncpus</code>	<i>Integer</i>	8

Example 4-3: If you use `'1 * ncpus + 1 * mem'`, where `mem=2mb`, `ncpus` will have almost no effect on the formula result. However, if you use `'1024 * ncpus + 1 * mem'`, the scaled `mem` won't overpower `ncpus`.

Example 4-4: You are using gb of `mem`:

```
Qmgr: s s job_sort_formula='1048576 * ncpus + 2 * mem'
```

Example 4-5: If you want to add days of `walltime` to queue priority, you might want to multiply the time by 0.0000115, equivalent to dividing by the number of seconds in a day:

```
Qmgr: s s job_sort_formula = '.0000115*walltime + queue_priority'
```

Note that a Python bug may make it necessary to multiply by 1.0 in order to prevent rounding to the nearest integer.

4.9.21.6 Resources in Formula

The formula can use resources in the job's `Resource_List` attribute, but no other resources. The resources in the job's `Resource_List` attribute are the numeric job-level resources, and may have been explicitly requested, inherited, or summed from consumable host-level resources. See [section 5.9.2, “Resources Requested by Job”, on page 241](#).

This means that all variables and coefficients in the formula must be resources that were either requested by the job or were inherited from defaults at the server or queue. These variables and coefficients can be custom numeric resources inherited by the job from the server or queue, or they are long integers or floats.

You may need to create custom resources at the server or queue level to be used for formula coefficients. See [section 4.9.8, “Using Custom and Default Resources”, on page 116](#).

4.9.21.7 Using Fairshare in the Formula

PBS provides the following fairshare values for use as keywords in the job sorting formula:

Table 4-13: Fairshare Terms in Formula

Keyword	Description
fairshare_perc (was fair_share_perc)	Percentage of fairshare tree allotted to this job's entity. See section 4.9.19.6.i, “Computing Target Usage for Each Vertex (fairshare_perc)” , on page 145.
fairshare_tree_usage	Value between 0 and 1, reflecting an entity's effective usage. See section 4.9.19.6.ii, “Computing Effective Usage (fairshare_tree_usage)” , on page 145.
fairshare_factor	Value between 0 and 1, which allows direct comparison between entities. A value of 0.5 means that an entity is using exactly its allotted usage. A higher value indicates less resource usage by the entity, meaning that the entity is more deserving. See section 4.9.19.6.iii, “Computing Relative Usage (fairshare_factor)” , on page 145.

See [section 4.9.19, “Using Fairshare”](#), on page 139.

4.9.21.8 Terms in Formula

Table 4-14: Terms in Job Sorting Formula

Terms		Allowable Value
Constants		<number> or <number>.<number>
Attributes, keywords, parameters, etc.	queue_priority	Value of priority attribute for queue in which job resides
	job_priority	Value of the job's Priority attribute
	fairshare_perc	Percentage of fairshare tree allotted to this job's entity
	fairshare_tree_usage	The effective usage by the entity
	fairshare_factor	Value allowing direct comparison between entities
	eligible_time	Amount of wait time job has accrued while waiting for resources
	accrue_type	Kind of time job is accruing. See section 4.9.13, “Eligible Wait Time for Jobs” , on page 129.
Resources		ncpus
		mem
		walltime
		cput
Custom numeric job-wide resources		Uses the amount requested, not the amount used. Must be of type long, float, or size. See section 5.14.2.2, “Custom Resource Values” , on page 255.

4.9.21.9 Modifying Coefficients For a Specific Job

Formula coefficients can be altered for each job by using the `qalter` command to change the value of that resource for that job. If a formula coefficient is a constant, it cannot be altered per-job.

4.9.21.10 Setting Minimum Job Priority Value for Job Execution

You can specify a minimum job priority value for jobs to run by setting the `job_sort_formula_threshold` scheduler attribute. If the value calculated for a job by the job sorting formula is at or below this value, the job cannot run during this scheduling cycle.

4.9.21.11 Examples of Using the Job Sorting Formula

Examples of formulas:

Example 4-6: $10 * \text{ncpus} + 0.01 * \text{walltime} + A * \text{mem}$

Here, "A" is a custom resource.

Example 4-7: $\text{ncpus} + 0.0001 * \text{mem}$

Example 4-8: To change the formula on a job-by-job basis, alter the value of a resource in the job's

`Resource_List.<resource name>`. So if the formula is $A * \text{queue_priority} + B * \text{job_priority} + C * \text{ncpus} + D * \text{walltime}$, where A-D are custom numeric resources. These resources can have a default value via `resources_default.A ... resources_default.D`. You can change the value of a job's resource through `qalter`.

Example 4-9: $\text{ncpus} * \text{mem}$

Example 4-10: Set via `qmgr`:

```
qmgr -c 'set server job_sort_formula= 5*ncpus+0.05*walltime'
```

Following this, the output from `qmgr -c 'print server'` will look like

```
set server job_sort_formula="5*ncpus+0.05*walltime"
```

Example 4-11:

```
Qmgr: s s job_sort_formula=ncpus
```

Example 4-12:

```
Qmgr: s s job_sort_formula='queue_priority + ncpus'
```

Example 4-13:

```
Qmgr: s s job_sort_formula='5*job_priority + 10*queue_priority'
```

Example 4-14: Sort jobs using the value of `ncpus` x `walltime`:

Formula expression: `"ncpus * walltime"`

Submit these jobs:

Job 1: `ncpus=2 walltime=01:00:00 -> 2*60s = 120`

Job 2: `ncpus=1 walltime=03:00:00 -> 1*180s = 180`

Job 3: `ncpus=5 walltime=01:00:00 -> 5*60s = 300`

The scheduler logs the following:

```
Job ;1.host1;Formula Evaluation = 120
```

```
Job ;2.host1;Formula Evaluation = 180
```

```
Job; 3.host1;Formula Evaluation = 300
```

The jobs are sorted in the following order:

Job 3

Job 2

Job 1

4.9.21.12 Examples of Using Resource Permissions in Job Sorting Formula

See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#) for information on using resource permissions.

Example 4-15: You may want to create per-job coefficients in your job sorting formula which are set by system defaults and which cannot be viewed, requested or modified by the user. To do this, you create custom resources for the formula coefficients, and make them invisible to users. In this example, A, B, C and D are the coefficients. You then use them in your formula:

$$A * (\text{Queue Priority}) + B * (\text{Job Class Priority}) + C * (\text{CPUs}) + D * (\text{Queue Wait Time})$$

Example 4-16: You may need to change the priority of a specific job, for example, have one job or a set of jobs run next. In this case, you can define a custom resource for a special job priority. If you do not want users to be able to change this priority, set the resource permission flag for the resource to *r*. If you do not want users to be able to see the priority, set its resource permission flag to *i*. For the job or jobs that you wish to give top priority, use `qalter` to set the special resource to a value much larger than any formula outcome.

Example 4-17: To use a special priority:

```
sched_priority = W_prio * wait_secs + P_prio * priority + ... + special_priority
```

Here, `special_priority` is very large.

4.9.21.13 Supporting Starving via the Formula

The formula can support the use of starving information by including how long a job has waited to run in its calculations.

Crafting a formula will be different for every site, but the thing to know about adding `eligible_time` to the formula is to correctly normalize it to the other factors. The `eligible_time` factor will grow to be large; for example, if a job waits for one day, it is already *86400*.

4.9.21.13.i Prerequisites for Starving Support via Formula

- Eligible time must be enabled:
`qmgr -c 'set server eligible_time_enable = True'`
- Strict ordering must be on to enable top job support. In `sched_config`:
`strict_ordering: True ALL`

4.9.21.13.ii Examples of Starving Support in Formula

Example 4-18: If there is no formula already, it is simple:

```
qmgr -c 'set sched job_sort_formula = eligible_time'
```

Example 4-19: Give shorter jobs priority over longer jobs:

```
qmgr -c 'set sched job_sort_formula = eligible_time / walltime'
```

Example 4-20: Give larger jobs priority. The `eligible_time` factor won't overtake the `ncpus` factor for 11.5 days. If this is too short, increase the scaling factor on `ncpus`:

```
qmgr -c 'set sched job_sort_formula = 1000000*ncpus + eligible_time'
```

Example 4-21: Recreate `help_starving_jobs` exactly using the `job_sort_formula`:

```
qmgr -c 'set sched job_sort_formula = 10000 if eligible_time > 86400 else 0'
```

4.9.21.14 Caveats and Error Messages

- If the formula overflows or underflows the sorting behavior is undefined.
- If you set the formula to an invalid formula, `qmgr` will reject it, with one of the following error messages:
`"Invalid Formula Format"`
`"Formula contains invalid keyword"`
`"Formula contains a resource of an invalid type"`
- If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:
`"1234.mars;Formula evaluation for job had an error. Zero value will be used"`
- The job sorting formula must be set via `qmgr` at the server host.
- When a job is moved to a new server or queue, it inherits new default resources from that server or queue. If it is moved to a new server, it is prioritized according to the formula on that server, if one exists.
- If the job is moved to another server through peer scheduling and the pulling server uses queue priority in its job sorting formula, the queue priority used in the formula will be that of the queue to which the job is moved.
- If you are using FIFO scheduling, the `job_sort_formula` server attribute must be unset.
- If you are using eligible time in the formula, and `eligible_time_enable` is *False*, each job's eligible time evaluates to zero in the formula.
- If a job is requeued, and you are using the formula, the job may lose its place, because various factors may affect the job's priority. For example, a higher-priority job may be submitted between the time the job is requeued and the time it would have run, or another job's priority may be increased due to changes in which jobs are running or waiting.
- If the formula is configured, it is in force during both primetime and non-primetime.
- If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:
`"1234.mars;Formula evaluation for job had an error. Zero value will be used"`
- You may have to work around a Python bug by multiplying by 1.0, in order to prevent rounding to the nearest integer.

4.9.21.15 Logging

For each job, the evaluated formula answer is logged at the highest log level (0x0400):

```
"Formula Evaluation = <answer>"
```

4.9.22 Gating Jobs at Server or Queue

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than a scheduler's partition or a queue can supply.

You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using the resources in the job sorting formula or to route jobs to particular vnodes.

You can either force users to submit their jobs to specific queues, or you can have users submit jobs to routing queues, and then route the jobs to the desired queues.

For information on using resources for gating, see [section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251](#).

For a description of which resources can be used for gating, see [section 2.3.6.4.iii, “Resources Used for Routing and Admittance”, on page 29](#).

For how queue resource limits are applied to jobs, see [section 2.3.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 29](#).

For how routing queues work, see [section 2.3.6, “Routing Queues”, on page 27](#).

For how to route jobs to particular vnodes, see [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#).

For how to use resources in the job sorting formula, see [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

4.9.22.1 Gating Caveats

- For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. See [section 5.9.3.6, “Using Gating Values As Defaults”, on page 243](#).
- For shrink-to-fit jobs, if a `walltime` limit is specified:
 - Both `min_walltime` and `max_walltime` must be greater than or equal to `resources_min.walltime`.
 - Both `min_walltime` and `max_walltime` must be less than or equal to `resources_max.walltime`.

4.9.23 Managing Application Licenses

PBS does not check application licenses out from the license server. PBS has no direct control over application licenses. However, you can have a scheduler use a dynamic resource to track application license use. This way, a scheduler knows how many application licenses are available, and how many have been checked out. For how to configure dynamic resources to represent application licenses, see [section 5.14.6, “Supplying Application Licenses”, on page 270](#).

Unfortunately, some jobs or applications don't check out all of the application licenses they use until they have been running for some time. For example, job J1, which requests licenses, starts running, but doesn't check them out for a few minutes. Next, the scheduler considers job J2, which also requests licenses. The scheduler runs its query for the number of available licenses, and the query returns with a sufficient number of licenses to run J2, so the scheduler starts J2. Shortly afterward, J1 checks out licenses, leaving too few to run J2.

It might appear that you could track the number of application licenses being used with a static integer PBS resource, and force jobs requesting application licenses to request this resource as well, but there is a drawback: if a job that has requested this resource is suspended, its static resources are released, but its application licenses are not. In this case you could end up with a deceptively high number for available licenses.

You can limit the number of jobs that request application licenses, if you know how many jobs can run at one time:

- Create a custom server-level consumable integer resource to represent these jobs. See [section 5.14.3, “Creating Server-level Custom Resources”, on page 263](#).
- Use `qmgr` to set `resources_available.<job limit>` at the server to the number of jobs that can run at one time.
- Force all jobs requesting the application to request one of these. See [section 10.3, “Allocating Resources to Jobs”, on page 455](#).

4.9.24 Limits on Per-job Resource Usage

You can specify how much of each resource any job is allowed to request, at the server and queue level. The server and queues each have per-job limit attributes. The `resources_min.<resource name>` and `resources_max.<resource name>` server and queue attributes are limits on what each individual job may request.

You cannot set `resources_min` or `resources_max` limits on `min_walltime` or `max_walltime`.

See [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#), and [section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251](#).

4.9.25 Limits on Project, User, and Group Jobs

You can manage the number of jobs being run by users or groups, and the number of jobs being run in projects, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.

See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”](#), on page 283.

4.9.26 Limits on Project, User, and Group Resource Usage

You can manage the total amount of each resource that is used by projects, users, or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.

See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”](#), on page 283.

4.9.27 Using Load Balancing

As of version 2022.1, the `load_balancing` scheduler parameter is **removed**. We recommend sorting vnodes according to load average, described in [section 4.9.49.3, “Sorting Vnodes According to Load Average”](#), on page 225.

When managing load levels on vnodes, a scheduler only pays attention to the state of the vnode, and does not calculate whether a job would put the vnode over its load limit. See [section 8.6.6, “Managing Load Levels on Vnodes”](#), on page 415.

4.9.28 Matching Jobs to Resources

A scheduler places each job where the resources requested by the job are available. A scheduler handles built-in and custom resources the same way. For a complete description of PBS resources, see [Chapter 5, “Using PBS Resources”](#), on page 227.

4.9.28.1 Scheduling on Consumable Resources

A scheduler constrains the use of a resource to the value that is set for that resource in `resources_available.<resource name>`. For a consumable resource, a scheduler won't place more demand on the resource than is available. For example, if a vnode has `resources_available.ncpus` set to 4, a scheduler will place jobs requesting up to a total of 4 CPUs on that vnode, but no more.

A scheduler computes how much of a resource is available by subtracting the total of `resources_assigned.<resource name>` for all running jobs and started reservations from `resources_available.<resource name>`.

4.9.28.2 Scheduling on Non-Consumable Resources

For non-consumable resources such as `arch` or `host`, a scheduler matches the value requested by a job with the value at one or more vnodes. Matching a job this way does not change whether or not other jobs can be matched as well; non-consumable resources are not used up by jobs, and therefore have no limits.

4.9.28.3 Scheduling on Dynamic Resources

At each scheduling cycle, a scheduler queries each dynamic resource. If a dynamic resource is not under the control of PBS, jobs requesting it may run in an unpredictable fashion.

4.9.28.4 Scheduling on the walltime Resource

A scheduler looks at each job in priority order, and tries to run the job. A scheduler checks whether there is an open time slot on the requested resources that is at least as long as the job's **walltime**. If there is, the scheduler runs the job.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's **min_walltime** and **max_walltime**. If the job can fit somewhere, PBS sets the job's **walltime** to a duration that fits the time slot, and runs the job. For more information about shrink-to-fit jobs, see [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

4.9.28.4.i Caveats for Scheduling on walltime

Do not set values for resources such as **walltime** at the server or a queue, because a scheduler will not allocate more than the specified value. This means that if you set **resources_available.walltime** at the server to **10:00:00**, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

4.9.28.5 Unrequestable or Invisible Resources

You can define custom resources that are invisible to and unrequestable by users, or simply unrequestable by users. A scheduler treats these resources the same as visible, requestable resources. See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).

4.9.28.6 Enforcing Scheduling on Resources

A scheduler chooses which resources to schedule on according to the following rules:

- A scheduler always schedules jobs based on the availability of the following vnode-level resources:
 - vnode
 - host
 - Any Boolean resource
- A scheduler will schedule jobs based on the availability of other resources only if those resources are listed in the **"resources:"** line in `<sched_priv directory>/sched_config`. Some resources are automatically added to this line. You can add resources to this line. The following resources are automatically added to this line:
 - aoe
 - arch
 - eo
 - host
 - mem
 - ncpus
 - vnode

4.9.28.7 Matching Unset Resources

When job resource requests are being matched with available resources, unset resources are treated as follows:

- A numerical resource that is unset on a host is treated as if it were **zero**
- An unset resource on the server or queue is treated as if it were infinite
- An unset string cannot be matched
- An unset Boolean resource is treated as if it were set to **False**.
- The resources **ompthreads**, **mpiprocs**, and **nodes** are ignored for unset resource matching.

The following table shows how a resource request will or won't match an unset resource at the host level.

Table 4-15: Matching Requests to Unset Host-level Resources

Resource Type	Unset Resource	Matching Request Value
Boolean	<i>False</i>	<i>False</i>
float	<i>0.0</i>	<i>0.0</i>
long	<i>0</i>	<i>0</i>
size	<i>0</i>	<i>0</i>
string	<i>""</i>	Never matches
string array	<i>""</i>	Never matches
time	<i>0, 0:0, 0:0.0, 0:0:0</i>	<i>0, 0:0, 0:0.0, 0:0:0</i>

4.9.28.7.i When Dynamic Resource Script Fails

If a server dynamic resource script fails, a scheduler uses the value of `resources_available.<resource name>`. If this was never set, it is treated as an unset resource, described above.

If a host-level dynamic resource script fails, a scheduler treats the resource as if its value is zero.

4.9.28.7.ii Backward Compatibility of Unset Resources

To preserve backward compatibility, you can set the server's `resource_unset_infinite` attribute with a list of host-level resources that will behave as if they are infinite when they are unset. See [“resource_unset_infinite” on page 257 of the PBS Professional Reference Guide](#) for information on `resource_unset_infinite`.

4.9.28.8 Resource Scheduling Caveats

- Do not set values for resources such as `walltime` at the server or a queue, because a scheduler will not allocate more than the specified value. This means that if you set `resources_available.walltime` at the server to `10:00:00`, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
- Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job's resource request will no longer match the same resources on the server, queues and vnodes.
- Beware of application license race conditions. If two jobs require the same application license, the first job may be started, but may not get around to using the license before the second job is started and uses the license. The first job must then wait until the license is available, taking up resources. A scheduler cannot avoid this problem.

4.9.29 Node Grouping

The term "*node grouping*" has been superseded by the term "*placement sets*". Vnodes were originally grouped according to the value of one resource, so for example all vnodes with a value of `linux` for `arch` were grouped together, and all vnodes with a value of `arch1` for `arch` were in a separate group. We use placement sets now because this means grouping vnodes according to the value of one *or more* resources. See [section 4.9.32, “Placement Sets”, on page 168](#).

4.9.29.1 Configuring Old-style Node Grouping

Configuring old-style node grouping means that you configure the simplest possible placement sets. In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will be in only one node group.

You enable node grouping by setting the server's `node_group_enable` attribute to *True*.

You can configure one set of vnode groups for the entire complex by setting the server's `node_group_key` attribute to a resource name.

You can configure node grouping separately for each queue by setting that queue's `node_group_key` attribute to a resource name.

4.9.30 Overrides

You can use various overrides to change how one or more jobs run.

4.9.30.1 Run a Job Manually

You can tell PBS to run a job now, and you can optionally specify where to run it. You run a job manually using the `qrun` command.

The `-H` option to the `qrun` command makes an important difference:

`qrun`

When preemption is enabled, a scheduler preempts other jobs in order to run this job. Running a job via `qrun` gives the job higher preemption priority than any other class of job, except for reservation jobs.

When preemption is not enabled, a scheduler runs the job only if enough resources are available.

`qrun -H`

PBS runs the job regardless of scheduling policy and available resources.

The `qrun` command alone overrides the following:

- Limits on resource usage by users, groups, and projects
- Limits on the number of jobs that can be run at a vnode
- Boundaries between primetime and non-primetime, specified in `backfill_prime`
- Whether the job is in a primetime queue: you can run a job in a primetime queue even when it's not primetime, or vice versa. Primetime boundaries are not honored.
- Dedicated time: you can run a job in a dedicated time queue, even if it's not in a dedicated time queue, and vice versa. However, dedicated time boundaries are still honored.
- Top jobs
- The threshold set in the `job_sort_formula_threshold` scheduler attribute
- The limit on the number of simultaneously running subjobs for an array job set in the `max_run_subjobs` job attribute

The `qrun` command alone does not override the following:

- Server and queue resource usage limits

4.9.30.1.i Using `qrun` Without `-H` Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via `qrun`, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job's `max_walltime` but not its `min_walltime`, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.
- If there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.9.30.1.ii Using `qrun` With `-H` Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via `qrun -H`, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via `qrun -H`, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for `walltime`, that value is used, instead of one set by PBS when the job runs. If no `walltime` is specified, the job runs without a `walltime`.

See [“qrun” on page 185 of the PBS Professional Reference Guide](#), and [section 4.9.33, “Using Preemption”, on page 180](#).

4.9.30.1.iii `qrun` Caveats

- A job that has just been run via `qrun` has top priority only during the scheduling cycle where it was `qrun`. At the next scheduling cycle, that job is available for preemption just like any other job.
- Be careful when using `qrun -H` on jobs or vnodes involved in reservations.

4.9.30.2 Hold a Job Manually

You can use the `qhold` command to place a hold on a job. The effect of placing a hold depends on whether the job is running and whether you have checkpointing configured:

- If the job is queued, the job will not run.
- If the job is running and checkpoint-abort is configured, the job is checkpointed, requeued, and held.
- If the job is running and checkpoint-abort is not configured, the only change is that the job's `Hold_Types` attribute is set to User Hold. If the job is subsequently requeued, it will not run until the hold is released.

You can release the hold using the `qrls` command.

For information on checkpointing jobs, see [section 8.3, “Checkpoint and Restart”, on page 387](#).

See [“qhold” on page 150 of the PBS Professional Reference Guide](#) and [“qrls” on page 183 of the PBS Professional Reference Guide](#).

4.9.30.3 Suspend a Job Manually

You can use the `qsig -s suspend` command to suspend a job so that it won't run. If you suspend a job, and then release it using the `qsig -s resume` command, the job remains in the suspended state until the required resources are available.

You can resume the job immediately by doing the following:

1. Resume the job:
`qsig -s resume <job ID>`
2. Run the job manually:
`qrun <job ID>`

See [“qsig” on page 195 of the PBS Professional Reference Guide](#).

4.9.30.4 Set Special Resource Value Used in Formula

You can change the value of a resource used in the job sorting formula. For example, to give a particular job a higher priority by changing the value of a custom resource called "higher":

- Create a custom resource that is invisible to job submitters:
`Qmgr: create resource higher type=float, flag=i`
- The formula expression includes "higher":
`Qmgr: s s job_sort_formula = "higher"`

- Set the default for this resource at the server:

```
Qmgr: set server resources_default.higher = 1
```

- These jobs are submitted:

Job 1

Job 2

Job 3

- Change Job 2 so that its value for "higher" is 5:

```
qalter -l higher = 5 job2
```

- The scheduler logs the following:

```
Job;1.host1;Formula Evaluation = 1
```

```
Job;2.host1;Formula Evaluation = 5
```

```
Job;3.host1;Formula Evaluation = 1
```

- Jobs are sorted in this order:

Job 2

Job 1

Job 3

4.9.30.5 Change Formula On the Fly

You can change the job sorting formula on the fly, so that the next scheduler iteration uses your new formula. This will change how job priorities are computed, and can rearrange the order in which jobs are run. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

4.9.30.6 Using Dedicated Time

You can set up blocks of dedicated time, where the only jobs eligible to be started or running are the ones in dedicated time queues. You can use dedicated time for upgrades. See [section 4.9.10, “Dedicated Time”, on page 128](#), and [section 2.3.5.2.i, “Dedicated Time Queues”, on page 26](#).

4.9.30.7 Using cron Jobs

You can use cron jobs to change PBS settings according to the needs of your time slots. See [section 4.9.7, “cron Jobs”, on page 115](#).

4.9.30.8 Using Hooks

You can use hooks to examine jobs and alter their characteristics. See the *PBS Professional Hooks Guide*.

4.9.30.9 Preventing Jobs from Being Calendared

You can prevent a scheduler from calendaring a job by setting its `topjob_ineligible` attribute to *True*. See [section 4.9.17, “Calendaring Jobs”, on page 138](#).

4.9.31 Peer Scheduling

Peer scheduling allows separate PBS partitions or complexes to automatically run jobs from each other's queues. This means that you can dynamically balance the workload across multiple, separate PBS partitions or complexes. These cooperating PBS partitions or complexes are referred to as "*Peers*".

4.9.31.1 How Peer Scheduling Works

In peer scheduling, a PBS server pulls jobs from one or more peer servers and runs them locally. When Partition or Complex A pulls a job from Partition or Complex B, Partition or Complex A is the "pulling" complex and Partition or Complex B is the "furnishing" partition or complex. When the pulling scheduler determines that another partition's or complex's job can immediately run locally, it moves the job to the specified queue on the pulling server and immediately run the job. The job is run as if it had been submitted to the pulling partition or complex.

You can set up peer scheduling so that A pulls from B and C, and so that B also pulls from A and C.

A job is pulled **only** when it can run immediately.

The pulling partition or complex must have all of the resources required by the job, including custom resources.

When a job is pulled from one partition or complex to another, the pulling partition or complex applies its policy to the job. The job's execution priority is determined by the policy of the pulling partition or complex. You can set special priority for pulled jobs; see [section 4.9.31.4.ii, "Setting Priority for Pulled Jobs", on page 167](#).

4.9.31.2 Prerequisites for Peer Scheduling

- You must create the pulling and furnishing queues before peer scheduling can be configured. See [section 2.3.3, "Creating Queues", on page 25](#) on how to create queues.
- When configuring peer scheduling, it is *strongly* recommended to use the same version of PBS Professional at all peer locations.
- Make sure that custom resources are consistent across peer locations. Jobs requesting custom resources at one location will not be able to run at another unless the same resources are available.
- If you are using MUNGE authentication, set it up for both PBS servers, put them in the same MUNGE domain, and use the same key for both servers. See [section 11.4, "Authentication for Daemons & Users", on page 508](#).

4.9.31.3 Configuring Peer Scheduling

The following sections give details on how to configure peer scheduling. Here is a brief outline:

- Define a flat user namespace on all complexes
- Map pulling queues to furnishing queues
- If necessary, specify port
- Grant manager access to each pulling server
- If possible, make user-to-group mappings be consistent across complexes
- If any of the peering sites is using failover, configure peering to work with failover

4.9.31.3.i Defining a Flat User Namespace

Peer scheduling requires a flat user namespace in all complexes involved. This means that user "joe" on the remote peer system(s) must be the same as user "joe" on the local system. Your site must have the same mapping of user to UID across all hosts, and a one-to-one mapping of UIDs to usernames. It means that PBS does not need to check whether X@hostA is the same as X@hostB; it can just assume that this is true. Set `flatuid` to `True`:

```
Qmgr: set server flatuid = True
```

For more on `flatuid`, see [section 11.3.12, "Flatuid and Access", on page 506](#).

4.9.31.3.ii Mapping Pulling Queues to Furnishing Queues

You configure peer scheduling by mapping a furnishing peer's queue to a pulling peer's queue. You can map each pulling queue to more than one furnishing queue, or more than one pulling queue to each furnishing queue.

The pulling and furnishing queues must be *execution* queues, not route queues. However, the queues can be either ordinary queues used for normal work, or special queues set up just for peer scheduling.

You map pulling queues to furnishing queues by setting the `peer_queue` scheduler configuration option in `<sched_priv directory>/sched_config`. The format is:

```
peer_queue: "<pulling queue> <furnishing queue>@<furnishing server>.domain"
```

For example, Complex A's queue "workq" is to pull from two queues: Complex B's queue "workq" and Complex C's queue "slowq". Complex B's server is ServerB and Complex C's server is ServerC. You would add this to Complex A's `<sched_priv directory>/sched_config`:

```
peer_queue: "workq workq@ServerB.domain.com"
```

```
peer_queue: "workq slowq@ServerC.domain.com"
```

Or if you wish to direct Complex B's jobs to queue Q1 on Complex A, and Complex C's jobs to Q2 on Complex A:

```
peer_queue: "Q1 workq@ServerB.domain.com"
```

```
peer_queue: "Q2 fastq@ServerC.domain.com"
```

In one partition or complex, you can create up to 50 mappings between queues. This means that you can have up to 50 lines in `<sched_priv directory>/sched_config` beginning with "peer_queue".

4.9.31.3.iii Specifying Ports

The default port for the server to listen on is 15001, and a scheduler uses any privileged port (1023 and lower). If the furnishing server is not using the default port, you must specify the port when you specify the queue. For example, if ServerB is using port 16001, and you wish to pull jobs from workq at ServerB to workq at ServerA, add this to `<sched_priv directory>/sched_config` at ServerA:

```
peer_queue: "workq workq@ServerB.domain.com:16001"
```

A scheduler and server communicate via TCP.

4.9.31.3.iv Granting Manager Access to Pulling Servers

Each furnishing server must grant manager access to each pulling server. If you wish jobs to move in both directions, where Complex A will both pull from and furnish jobs to Complex B, ServerA and ServerB must grant manager access to each other.

On the furnishing complex:

```
Qmgr: set server managers += root@pullingServer.domain.com
```

4.9.31.3.v Making User-to-group Mappings Consistent Across Complexes

If possible, ensure that for each user in a peer complex, that user is in the same group in all participating complexes. So if user "joe" is in groupX on Complex A, user "joe" should be in groupX on Complex B. This means that a job's `egroup` attribute will be the same on both complexes, and any group limit enforcement can be properly applied.

There is a condition when using peer scheduling in which group hard limits may not be applied correctly. This can occur when a job's effective group, which is its `egroup` attribute, i.e. the job's owner's group, is different on the furnishing and pulling systems. When the job is moved over to the pulling complex, it can evade group limit enforcement if the group under which it will run on the pulling system has not reached its hard limit. The reverse is also true; if the group under which it will run on the pulling system has already reached its hard limit, the job won't be pulled to run, although it should.

This situation can also occur if the user explicitly specifies a group via `qsub -W group_list`.

It is recommended to advise users to *not* use the `qsub` options "`-u user_list`" or "`-W group_list=groups`" in conjunction with peer scheduling.

4.9.31.3.vi Configuring Peer Scheduling with Failover

If you are configuring peer scheduling so that Complex A will pull from Complex B where Complex B is configured for failover, you must configure Complex A to pull from both of Complex B's servers. For these instructions, see [section 8.2.6.2, “Configuring Failover to Work With Peer Scheduling”](#), on page 384.

4.9.31.4 Peer Scheduling Advice

4.9.31.4.i Selective Peer Scheduling

You can choose the kinds of jobs that can be selected for peer scheduling to a different partition or complex. You can do the following:

- Set resource limits at the furnishing queue via the `resources_min` and `resources_max` queue attributes. See [section 2.3.6.4, “Using Resources to Route Jobs Between Queues”](#), on page 28.
- Route jobs into the furnishing queue via a hook. See [“Routing Jobs” on page 7 in the PBS Professional Hooks Guide](#).
- Route jobs into the furnishing queue via a routing queue. See [section 2.3.6, “Routing Queues”](#), on page 27.

4.9.31.4.ii Setting Priority for Pulled Jobs

You can set a special priority for pulled jobs by creating a queue that is used only as a pulling queue, and setting the pulling queue's priority to the desired level. You can then use the queue's priority when setting job execution priority. See [section 4.3.5.4.iv, “Using Queue Priority when Computing Job Priority”](#), on page 69.

For example, if you give the pulling queue the lowest priority, the pulling partition or complex will pull a job only when there are no higher-priority jobs that can run.

You can also have pulled jobs land in a special queue where they inherit a custom resource that is used in the job sorting formula.

4.9.31.5 How Peer Scheduling Affects Jobs

4.9.31.5.i How Peer Scheduling Affects Inherited Resources

If the job is moved partition or complex to another via peer scheduling, any default resources in the job's resource list inherited from the furnishing queue or server are removed. This includes any select specification and place directive that may have been generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either `select` or `place` is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing partition or complex. If the job is requeued and restarted at the pulling partition or complex, the job picks up new default resources from the pulling partition or complex, and is scheduled according to the newly-inherited resources from the pulling partition or complex.

4.9.31.5.ii How Peer Scheduling Affects Policy Applied to Job

After a job is pulled from one partition or complex to another, the scheduling policy of the pulling partition or complex is applied to the job.

For example, if you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the queue to which the job is moved.

When a job is pulled from one partition or complex to another, hooks are applied at the new partition or complex as if the job had been submitted locally. For example, if the pulling partition or complex has a `queuejob` hook, that hook runs when a job is pulled.

4.9.31.5.iii How Peer Scheduling Affects Job Eligible Time

The job's `eligible_time` is preserved when a job is moved due to peer scheduling.

4.9.31.5.iv Viewing Jobs That Have Been Moved to Another Server

If you are connected to ServerA and a job submitted to ServerA has been moved from ServerA to ServerB through peer scheduling, in order to display it via `qstat`, give the job ID as an argument to `qstat`. If you only give the `qstat` command, the job will not appear to exist. For example, the job `123.ServerA` is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

4.9.31.5.v Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any `movejob` hooks at the furnishing server are run

4.9.31.6 Peer Scheduling Caveats

- Each partition or complex can peer with at most 50 other partitions or complexes.
- When using peer scheduling, group hard limits may not be applied correctly. This can occur when the job owner's group is different on the furnishing and pulling systems. For help in avoiding this problem, see [section 4.9.31.3.v, “Making User-to-group Mappings Consistent Across Complexes”, on page 166](#).
- When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing partition or complex. If the job is requeued and restarted at the pulling partition or complex, the job picks up new default resources from the pulling partition or complex, and is scheduled according to the newly-inherited resources from the pulling partition or complex.
- Peer scheduling is not supported for job arrays.

4.9.32 Placement Sets

Placement sets are the sets of vnodes within which PBS will try to place a job. PBS tries to group vnodes into the most useful sets, according to how well connected the vnodes are, or the values of resources available at the vnodes. Placement sets are used to improve task placement (optimizing to provide a "good fit") by exposing information on system configuration and topology. A scheduler tries to put a job in the smallest appropriate placement set.

4.9.32.1 Definitions

Task placement

The process of choosing a set of vnodes to allocate to a job that will satisfy both the job's resource request (select and place specifications) and the configured scheduling policy.

Placement Set

A set of vnodes. Placement sets are defined by the values of vnode-level string array resources. A placement set is all of the vnodes that have the same value for a specified defining resource substring. For example, if the defining resource is a vnode-level string array named *"switch"*, which can have values *"S1"*, *"S2"*, or *"S3"*: the set of vnodes which have a substring matching *"switch=S2"* is a placement set.

Placement sets can be specified at the server or queue level.

Placement Set Series

A set of placement sets; a set of sets of vnodes.

A placement set series is all of the placement sets that are defined by specifying one string array resource. Each placement set in the series is the set of vnodes that share one value for the resource. There is one placement set for each value of the resource. If the resource takes on *N* values at the vnodes, then there are *N* sets in the series. For example, if the defining resource is a string array named *"switch"*, which can have values *"S1"*, *"S2"*, or *"S3"*, there are three sets in the series. The first is defined by the value *"S1"*, where all the vnodes in that set have the value *"S1"* for the resource *switch*. The second set is defined by *"S2"*, and the third by *"S3"*.

Each of the resources named in *node_group_key* specifies a placement series. For example, if the server's *node_group_key* attribute contains *"router,switch"*, then the server has two placement set series.

Placement Pool

All of the placement sets that are defined; the server can have a placement pool, and each queue can have its own placement pool. If a queue has no placement pool, a scheduler uses the server's placement pool.

A placement pool is the set of placement set series that are defined by one or more string array resources named in *node_group_key*.

For example, if the server's *node_group_key* attribute contains *"router,switch"*, and *router* can take the values *"R1"* and *"R2"* and *switch* can take the values *"S1"*, *"S2"*, and *"S3"*, then there are five placement sets, in two placement series, in the server's placement pool.

Static Fit

A job statically fits into a placement set if the job could fit into the placement set if the set were empty. It might not fit right now with the currently available resources.

Dynamic Fit

A job dynamically fits into a placement set if it will fit with the currently available resources (i.e. the job can fit right now).

4.9.32.2 Requirements for Placement Sets

- Placement sets are enabled by setting the server's *node_group_enable* attribute to *True*
- Server-level placement sets are defined by setting the server's *node_group_key* attribute to a list of vnode-level string array resources.
- Queue-level placement sets are defined by setting a queue's *node_group_key* attribute to a list of vnode-level string array resources.
- At least one vnode-level string array resource must exist on vnodes and be set to values that can be used to assign the vnodes to placement sets.

4.9.32.3 Description of Placement Sets

4.9.32.3.i What Defines a Placement Set, Series, or Pool

Placement sets are defined by the values of vnode-level string array resources. You define placement sets by specifying the names of these resources in the `node_group_key` attribute for the server and/or queues. Each value of each resource defines a different placement set. A placement set is all of the vnodes that have the same value for a specified defining resource. For example, if the defining resource is a vnode-level string array named `"switch"`, which has the values `"S1"`, `"S2"`, and `"S3"`, the set of vnodes where `switch` has the value `"S2"` is a placement set. If some vnodes have more than one substring, and one of those substrings is the same in each vnode, those vnodes make up a placement set. For example, if the resource is `"router"`, and vnode V0 has `resources_available.router` set to `"r1i0,r1"`, and vnode V1 has `resources_available.router` set to `"r1i1,r1"`, V0 and V1 are in the placement set defined by `resources_available.router = "r1"`. If the resource has N distinct values across the vnodes, including the value zero and being unset, there can be $N-1$ or N placement sets defined by that resource. If the `only_explicit_psets` scheduler attribute is `False`, there are N placement sets. If the `only_explicit_psets` scheduler attribute is `True`, there are $N-1$ placement sets; see [section 4.9.32.3.v, "Placement Sets Defined by Unset Resources"](#), on page 171.

Each placement set can have a different number of vnodes; the number of vnodes is determined only by how many vnodes share that resource value.

Each placement set series is defined by the values of a single resource across all the vnodes. For example, if there are three switches, S1, S2 and S3, and there are vnodes with `resources_available.switch` that take on one or more of these three values, then there will be three placement sets in the series.

Whenever you define any placement sets, you are defining a placement pool. Placement pools can be defined for the server and for each queue. You define a server-level placement pool by setting the server's `node_group_key` to a list of one or more vnode-level string array resources. You define a queue-level placement pool by similarly setting the queue's `node_group_key`.

4.9.32.3.ii Vnode Participation in Placement Sets, Series, and Pools

Each vnode can be in multiple placement sets, placement set series, and placement pools.

A vnode can be in multiple placement sets in the same placement set series. For example, if the resource is called `"router"`, and a vnode's `router` resource is set to `"R1, R2"`, then the vnode will be in the placement set defined by `router = R1` and the set defined by `router = R2`.

A vnode is in a placement series whenever the resource that defines the series is defined on the vnode. For example, if placement sets are defined by the values of the `router` and the `switch` resources, and a vnode has value `R1` for `router`, and `S1` for `switch`, then the vnode is in both placement series, because it is in the set that shares the `R1` value for `router`, and the set that shares the `S1` value for `switch`. Each of those sets is one of a different series.

The server has its own placement pool if the server's `node_group_key` attribute is set to at least one vnode-level string array resource. Similarly, each queue can have its own placement pool. A vnode can be in any placement pool that specifies a resource that is defined on the vnode.

4.9.32.3.iii Multihost Placement Sets

Placement sets, series, and pools can span hosts. Placement sets can be made up of vnodes from anywhere, regardless of whether the vnode is from a multi-vnode host.

To set up a multihost placement set, choose a string array resource for the purpose, and list it in the desired `node_group_key` attribute. For example, create a string_array resource called `"span"`:

```
Qmgr: create resource span type=string_array, flag=h
```

Add the resource `"span"` to `node_group_key` on the server or queue. Use `qmgr` to give it the same value on all the desired vnodes. You can write a script that sets the same value on each vnode that you want in your placement set.

4.9.32.3.iv Machines with Multiple Vnodes

Machines with multiple vnodes are represented as a generic set of vnodes. Placement sets are used to allocate resources on a single machine to improve performance and satisfy scheduling policy and other constraints. Jobs are placed on vnodes using placement set information.

4.9.32.3.v Placement Sets Defined by Unset Resources

The `only_explicit_psets` scheduler attribute controls whether unset resources define placement sets.

- If the `only_explicit_psets` scheduler attribute is *False*, vnodes where a defining resource is unset are grouped into their own placement set, for each defining resource. For example, if you have ten vnodes, on which there is a string resource `COLOR`, where two have `COLOR` set to "red", two are set to "blue", two are set to "green" and the rest are unset, there will be four placement sets defined by the resource `COLOR`. This is because the fourth placement set consists of the four vnodes where `COLOR` is unset. This placement set will also be the largest. Every resource listed in `node_group_key` can potentially define such a placement set.
- If the `only_explicit_psets` scheduler attribute is *True*, vnodes where a resource is unset are not grouped into placement sets.

4.9.32.3.vi Placement Sets and Node Grouping

Node grouping is the same as one placement set series, where the placement sets are defined by a single resource. Node grouping has been superseded by placement sets.

In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will only be in one node group.

4.9.32.4 How Placement Sets Are Used

You use placement sets to group vnodes according to the value of one or more resources. Placement sets allow you to group vnodes into useful sets.

You can run multi-vnode jobs in one placement set. For example, it makes the most sense to run a multi-vnode job on vnodes that are all connected to the same high-speed switch.

PBS will attempt to place each job in the smallest possible set that is appropriate for the job.

4.9.32.4.i Order of Placement Pool Selection

A scheduler chooses one placement pool from which to select a placement set.

Queue placement pools override the server's placement pool. If a queue has a placement pool, jobs from that queue are placed using the queue's placement pool. If a queue has no placement pool (the queue's `node_group_key` is not defined), jobs are placed using the server's placement pool, if it exists.

A per-job placement set is defined by the `-l place` statement in the job's resource request. Since the job can only request one value for the resource, it can only request one specific placement set. A job's `place=group` resource request overrides the sets defined by the queue's or server's `node_group_key`.

A scheduler chooses the most specific placement pool available, following this order of precedence:

1. A per-job placement set (job's `place=group=` request)
2. A placement set from the placement pool for the job's queue
3. A placement set from the placement pool in a scheduler's partition

4.9.32.4.ii Order of Placement Set Consideration Within Pool

A scheduler looks in the selected placement pool and chooses the smallest possible placement set that is appropriate for the job. A scheduler examines the placement sets in the pool and orders them, from smallest to largest, according to the following rules:

1. Static total `ncpus` of all vnodes in set
2. Static total `mem` of all vnodes in set
3. Dynamic free `ncpus` of all vnodes in set
4. Dynamic free `mem` of all vnodes in set

If a job can fit statically within any of the placement sets in the placement pool, then a scheduler places a job in the first placement set in which it fits dynamically. This ordering ensures a scheduler will use the smallest possible placement set in which the job will dynamically fit. If there are multiple placement sets where the job fits statically, but some are being used, a scheduler uses the first placement set where the job can run now. If the job fits statically into at least one placement set, but these placement sets are all busy, a scheduler waits until a placement set can fit the job dynamically.

For example, we have the following placement sets, and a job that requests 8 CPUs:

Set1 `ncpus` = 4

Set2 `ncpus` = 12; this placement set is full

Set3 `ncpus` = 16; this placement set is not being used

The scheduler looks at Set1; Set1 is statically too small, and the scheduler moves to the next placement set. Set2 is statically large enough, but the job does not fit dynamically. The scheduler looks at Set3; Set3 is large enough, and the job fits dynamically. The scheduler runs the job in Set3.

If the job requests 24 CPUs, the scheduler attempts to run the job in the set consisting of all vnodes that are associated with a specific queue, if `do_not_span_psets` is *False*.

4.9.32.4.iii Determining Whether Job Can Run

Whether the job can run in the selected placement pool is determined by the value of the `do_not_span_psets` attribute.

- If this attribute is *False*, and a job cannot statically fit into any placement set in the selected placement pool, a scheduler ignores defined placement sets and uses all vnodes that satisfy job restrictions as its placement set, and runs the job without regard to placement sets. For example, if the job's queue has access to a restricted set of vnodes, the job runs within that set of vnodes.
- If the attribute is *True*, a scheduler does not run the job.

4.9.32.4.iv Order of Vnode Selection Within Set

A scheduler orders the vnodes within the selected placement set using the following rules:

- If `node_sort_key` is set, vnodes are sorted by `node_sort_key`. See [section 4.9.49, “Sorting Vnodes on a Key”, on page 224](#).
- If `node_sort_key` is not set, the order in which the vnodes are returned by `pbs_statnode()`. This is the default order the vnodes appear in the output of the `pbsnodes -a` command.

A scheduler places the job on the vnodes according to their ordering above.

4.9.32.5 Summary of Placement Set Requirements

The steps to configure placement sets are given in the next section. The requirements are summarized here for convenience:

- Definitions of the resources of interest
- Vnodes defining a value for each resource to be used for placement sets (e.g., rack)
 - If defined via vnode definition, you must HUP the MoMs involved
- The server's or queue's `node_group_key` attribute must be set to the resources to be used for placement sets. For example, if we have custom resources named "rack", "socket", "board", and "boardpair", which are to be used for placement sets:

```
Qmgr: set server node_group_key = "rack,socket,board,boardpair"
```

- No signals needed, takes effect immediately
- Placement sets must be enabled at the server by setting the server's `node_group_enable` attribute to *True*. For example:

```
Qmgr: set server node_group_enable=True
```

- No signals needed, takes effect immediately

Adding a resource to a scheduler's `resources:` line is required only if the resource is to be specifically requested by jobs. It is not required for `-lplace=group=<resource name>`.

4.9.32.6 How to Configure Placement Sets

The following steps show how to satisfy the requirements for placement sets:

1. If the vnodes that you will use in placement sets are not defined, define them. See [section 3.3, “Creating Vnodes”, on page 42](#).
2. If the vnode-level string array resources that you will use to define placement sets do not exist, create them. See [section 5.14.4, “Configuring Host-level Custom Resources”, on page 265](#).
3. If values for the vnode-level string array resources that you will use to define placement sets are not set at the vnodes you wish to use, set the values. See [section 3.4, “Configuring Vnodes”, on page 45](#).
4. If you use vnode definition files to set values for vnode-level string array resources, HUP the MoMs involved.
5. To create queue placement pools, set the `node_group_key` attribute to the name(s) of one or more vnode-level string array resources. Do this for each queue for which you want a separate pool. For example:

```
Qmgr: set queue workq node_group_key = <router,switch>
```

6. To create a server placement pool, set the `node_group_key` server attribute to the name(s) of one or more vnode-level string array resources. For example:

```
Qmgr: set server node_group_key = <router,switch>
```

For example, to create a server-level placement pool for the resources `host`, `L2` and `L3`:

```
Qmgr: set server node_group_key = "host,L2,L3"
```

7. Set the server's `node_group_enable` attribute to *True*

```
Qmgr: set server node_group_enable = True
```

8. Set the `do_not_span_psets` scheduler attribute to *True* if you don't want jobs to span placement sets.

```
Qmgr: set sched do_not_span_psets = True
```

9. Set the `only_explicit_psets` attribute to *True* if you don't want a scheduler to create placement sets from unset resources.

```
Qmgr: set sched only_explicit_psets = True
```

10. For ease of reviewing placement set information, you can add the name of each resource used in each vnode's `pnames` attribute:

```
Qmgr: active node <vnode name>,<vnode name>,...
```

```
Qmgr: set node pnames += <resource name>
```

or

```
Qmgr: set node pnames = <resource list>
```

For example:

```
Qmgr: set node pnames = "board,boardpair,iruquadrant,iruhalf,iru,rack"
```

We recommend using the parent vnode for any placement set information that is invariant for a given host.

Resources used only for defining placement sets, and not for allocation to jobs, do not need to be listed in the `resources:` line in `<sched_priv directory>/sched_config`. So for example if you create a resource just for defining placement sets, and jobs will not be requesting this resource, you do not need to list it in the `resources:` line.

4.9.32.7 Examples of Creating Placement Sets

4.9.32.7.i Cluster with Four Switches

This cluster is arranged as shown with vnodes 1-4 on Switch1, vnodes 5-10 on Switch2, and vnodes 11-24 on Switch3. Switch1 and Switch2 are on Switch4.

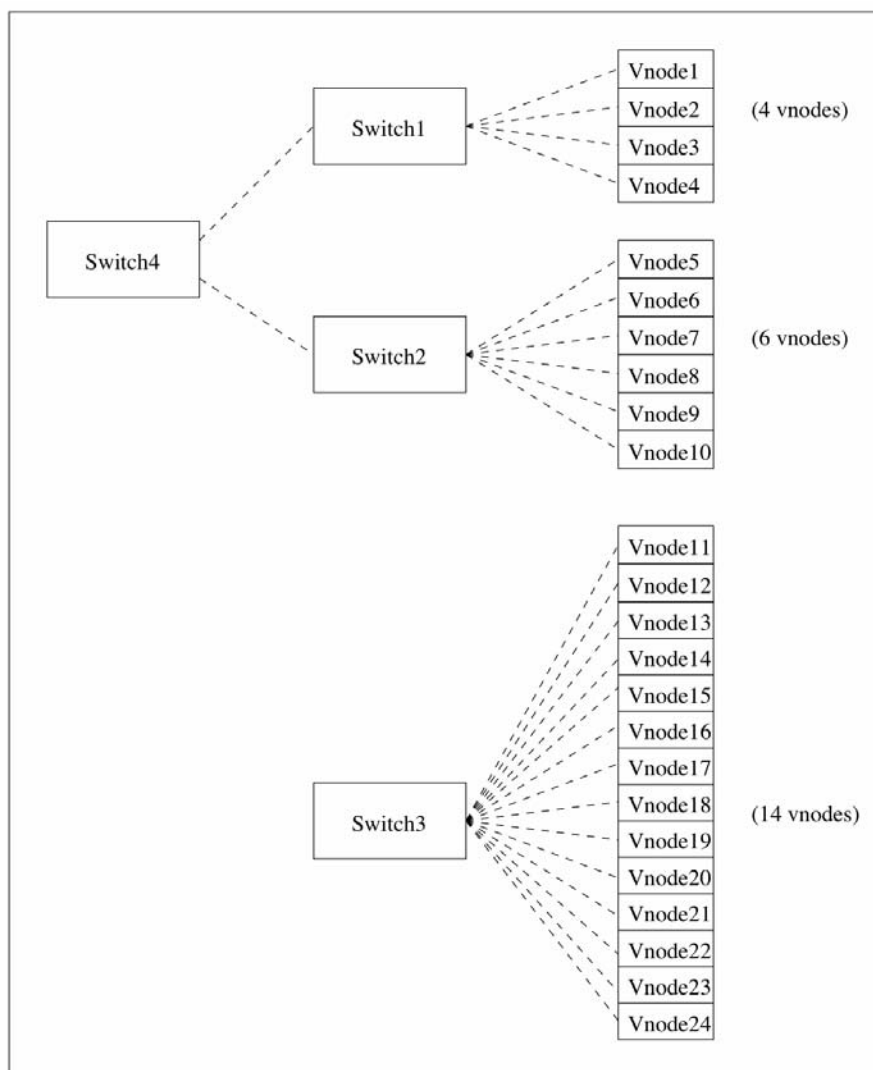


Figure 4-1: Cluster with Four Switches

To make the placement sets group the vnodes as they are grouped on the switches:

Create a custom resource called *switch*. The *-h* flag makes the resource requestable:

```
Qmgr: create resource switch type=string_array, flag=h
```

On vnodes[1-4] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch1,switch4"
```

On vnodes[5-10] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch2,switch4"
```

On vnodes[11-24] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch3"
```

On the server set:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

So you have 4 placement sets:

The placement set "switch1" has 4 vnodes

The placement set "switch2" has 6 vnodes

The placement set "switch3" has 14 vnodes

The placement set "switch4" has 10 vnodes

PBS will try to place a job in the smallest available placement set. Does the job fit into the smallest set (switch1)? If not, does it fit into the next smallest set (switch2)? This continues until it finds one where the job will fit.

4.9.32.7.ii Example of Configuring Placement Sets on a Multi-vnode Machine

For information on how to configure vnodes via Version 2 configuration files, see [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).

In this example, we define a new placement set using the new resource "NewRes". We create a file called `SetDefs` that contains the changes we want.

1. Create the new resource:

```
Qmgr: create resource NewRes type=string_array, flag=h
```

2. Add NewRes to the server's node_group_key

```
Qmgr: set server node_group_key+="NewRes"
```

3. Add NewRes to the value of the pnames attribute for the parent vnode. This makes the name of the resource you used easily available. Add a line like this to `SetDefs`:

```
host3: resources_available.pnames =...,NewRes
```

4. For each vnode, V, that's a member of a new placement set you're defining, add a line of the form:

```
V: resources_available.NewRes = <value1[,...]>
```

All the vnodes in the new set should have lines of that form, with the same resource value, in the new configuration file.

Here the value of the resource is "P" and/or "Q".

We'll put vnodes A, B and C into one placement set, and vnodes B, C and D into another.

```
A: resources_available.NewRes2 = P
```

```
B: resources_available.NewRes2 = P,Q
```

```
C: resources_available.NewRes2 = P,Q
```

```
D: resources_available.NewRes2 = Q
```

For each new placement set you define, use a different value for the resource.

5. Add `SetDefs` and tell MoM to read it, to make a Version 2 vnode configuration file `NewConfig`:

```
pbs_mom -s insert NewConfig SetDefs
```

6. Stop and restart the MoM. For Linux, see [“Restarting and Reinitializing MoM” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“Restarting MoMs” on page 159 in the PBS Professional Installation & Upgrade Guide](#).

4.9.32.7.iii Example of Placement Sets Using Colors

A placement pool is defined by two resources: `colorset1` and `colorset2`, by using `"node_group_key=colorset1,colorset2"`.

If a vnode has the following values set:

```
resources_available.colorset1=blue, red
resources_available.colorset2=green
```

The placement pool contains at least three placement sets. These are:

```
{resources_available.colorset1=blue}
{resources_available.colorset1=red}
{resources_available.colorset2=green}
```

This means the vnode is in all three placement sets. The same result would be given by using one resource and setting it to all three values, e.g. `colorset=blue,red,green`.

Example: We have five vnodes `v1 - v5`:

```
v1 color=red host=mars
v2 color=red host=mars
v3 color=red host=venus
v4 color=blue host=mars
v5 color=blue host=mars
```

The placement sets are defined by

```
node_group_key=color
```

The resulting node groups would be: `{v1, v2, v3}`, `{v4, v5}`

4.9.32.7.iv Simple Switch Placement Set Example

Say you have a cluster with two high-performance switches each with half the vnodes connected to it. Now you want to set up placement sets so that jobs will be scheduled only onto the same switch.

First, create a new resource called `"switch"`. See [section 5.14.2, “Defining New Custom Resources”, on page 254](#).

Next, we need to enable placement sets and specify the resource to use:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

Now, set the value for `switch` on each vnode:

```
Qmgr: active node vnode1,vnode2,vnode3
Qmgr: set node resources_available.switch=A
Qmgr: active node vnode4,vnode5,vnode6
Qmgr: set node resources_available.switch=B
```

Now there are two placement sets:

```
switch=A: {vnode1, vnode2, vnode3}
switch=B: {vnode4, vnode5, vnode6}
```

4.9.32.8 Placement Sets and Reservations

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See [section 4.9.32.4.ii, “Order of Placement Set Consideration Within Pool”, on page 172](#).

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

4.9.32.9 Placement Sets and Load Balancing

If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

4.9.32.10 Viewing Placement Set Information

You can find information about placement sets in the following places:

- The server's `node_group_enable` attribute shows whether placement sets are enabled
- The server's `node_group_key` attribute contains the names of resources used for that queue's placement pool
- Each queue's `node_group_key` attribute contains the names of resources used for that queue's placement pool
- Each vnode's `pnames` attribute can contain the names of resources used for placement sets, if properly set
- A scheduler's `do_not_span_psets` attribute shows whether jobs are allowed to span placement sets
- A scheduler's `only_explicit_psets` attribute shows placement sets are created using unset resources
- PBS-generated MoM configuration files contain names and values of resources

4.9.32.11 Placement Set Caveats and Advice

- If there is a vnode-level platform-specific resource set on the vnodes on a multi-vnode machine, then `node_group_key` should probably include this resource, because this will enable PBS to run jobs in more logical sets of vnodes.
- If the user specifies a job-specific placement set, for example `-lplace=group=switch`, but the job cannot statically fit into any switch placement set, then the job will still run, but not in a switch placement set.
- The `pnames` vnode attribute is for displaying to the administrator the resources used for placement sets. This attribute is not used by PBS.

4.9.32.11.i Non-backward-compatible Change in Node Grouping

Given the following example configuration:

```

vnode1: switch=A
vnode2: switch=A
vnode3: switch=B
vnode4: switch=B
vnode5: switch unset
Qmgr: s s node_group_key=switch

```

There is no change in the behavior of jobs submitted with `qsub -l ncpus=1`

version 7.1: The job can run on any node: node1, ..., node5

version 8.0: The job can run on any node: node1, ..., node5

Example of 8.0 and later behavior: jobs submitted with `qsub -lnodes=1`

version 7.1: The job can only run on nodes: node1, node2, node3, node4. It will never use node5

version 8.0: The job can run on any node: node1, ..., node5

Overall, the change for version 8.0 was to include every vnode in placement sets (when enabled). In particular, if a resource is used in `node_group_key`, PBS will treat every vnode as having a value for that resource, hence every vnode will appear in at least one placement set for every resource. For vnodes where a string resource is "unset", PBS will behave as if the value is "".

4.9.32.12 Attributes and Parameters Affecting Placement Sets

do_not_span_psets

Scheduler attribute. Specifies whether or not this scheduler requires the job to fit within one of the existing placement sets. When `do_not_span_psets` is set to *True*, a scheduler will require the job to fit within a single existing placement set. A scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.

When this attribute is set to *False*, a scheduler first attempts to place the job in a single placement set. All existing placement sets are checked. If the job fits in an occupied placement set, the job waits for the placement set to be available. If there is no existing placement set, occupied or empty, into which the job could fit, the job runs regardless of placement sets, running on whichever vnodes can satisfy the job's resource request.

Format: *Boolean*

Default value: *False* (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

```
Qmgr: set sched do_not_span_psets=True
```

node_group_enable

Server attribute. Specifies whether placement sets are enabled.

Format: *Boolean*

Default: *False*

node_group_key

Server and queues have this attribute. Specifies resources to use for placement set definition. Queue's attribute overrides server's attribute.

Format: *string_array*

Default: Unset

only_explicit_psets

Scheduler attribute. Specifies whether placement sets are created using unset resources. If *False*, for each defining resource, if there are vnodes where the value of the resource is unset, PBS creates a placement set for the series defined by that resource. If *True*, PBS does not create placement sets for resources that are unset.

Format: *Boolean*

Default: *False*

4.9.32.13 Errors and Logging

If `do_not_span_psets` is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
"Not Running: can't fit in the largest placement set, and can't span psets"
- The following message is printed to the scheduler's log:
"Can't fit in the largest placement set, and can't span placement sets"

4.9.33 Using Preemption

PBS provides the ability to preempt currently running jobs in order to run higher-priority work. This is called *preemption* or *preemptive scheduling*. PBS has two different approaches to specifying preemption:

- You can define a set of preemption priorities for all jobs. Jobs that have high preemption priority preempt those with low preemption priority. Preemption priority is mostly independent of execution priority. See [section 4.9.33.7, “Preemption Levels”, on page 184](#).
- You can specify a set of preemption targets for each job. You can also set defaults for these targets at the server and queues. Preemption targets are jobs in specific queues or that have requested specific resources. See [section 4.9.33.4, “Using Preemption Targets”, on page 182](#).

Preemption is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.9.33.1 Glossary

Preempt

Stop one or more running jobs in order to start a higher-priority job

Preemption level

Job characteristic that determines preemption priority. Levels can be things like being in an express queue, having an owner who is over a soft limit, being a normal job, or having an owner who is over a fairshare allotment

Preemption method

The method by which a job is preempted. This can be checkpointing, suspension, requeueing, or deletion

Preemption priority

How important this job is compared to other jobs, when considering whether to preempt

Preemption Target

A preemption target is a job in a specified queue or a job that has requested a specified resource. The queue and/or resource is specified in another job's `Resource_List.preempt_targets`.

4.9.33.2 Preemption Parameters and Attributes

The scheduler parameters that control preemption are defined in `<sched_priv directory>/sched_config`. A scheduler also has attributes that control preemption; they can be set via `qmgr`. Parameters and attributes that control preemption are listed here:

preemptive_sched

Scheduler parameter defined in `<sched_priv directory>/sched_config`. Enables job preemption.

Format: *String*

Default: *True all*

preempt_order

Scheduler attribute. Defines the order of preemption methods which this scheduler will use on jobs. Can contain any of *S*, *C*, *R*, and *D*, in any order.

Format: *String*, as quoted list

Default: *"SCR"*

preempt_prio

Scheduler attribute. Specifies the ordering of priority of different preemption levels.

Format: *String*, as quoted list

Default: *"express_queue, normal_jobs"*

preempt_queue_prio

Scheduler attribute. Specifies the minimum queue priority required for a queue to be classified as an express queue.

Format: *Integer*

Default: *150*

preempt_sort

Scheduler attribute. Whether jobs most eligible for preemption will be sorted according to their start times.

Allowable values: *"min_time_since_start"*. The first job preempted will be that with most recent start time.

Format: *String*

Default: *min_time_since_start*

preempt_targets

Resource that a job can request or inherit from the server or a queue. The **preempt_targets** resource lists one or more queues and/or one or more resources. Jobs in those queues, and jobs that request those resources, are the jobs that can be preempted.

restrict_res_to_release_on_suspend

Server attribute. Comma-separated list of consumable resources to be released when jobs are suspended. If unset, all consumable resources are released on suspension. See [section 5.9.6.2, “Job Suspension and Resource Usage”, on page 247](#) and [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

Format: *string_array*

Default: *unset*

Python type: *list*

resources_released

Job attribute. Listed by vnode, consumable resources that were released when the job was suspended. Populated only when **restrict_res_to_release_on_suspend** server attribute is set. See [section 5.9.6.2, “Job Suspension and Resource Usage”, on page 247](#) and [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).

Format: String of the form: (*<vnode>:<resource name>=<value>:<resource name>=<value>:...*)+(*<vnode>:<resource name>=<value>:...*)

Python type: *str*

resource_released_list

Job attribute. Sum of each consumable resource requested by the job that was released when the job was suspended. Populated only when **restrict_res_to_release_on_suspend** server attribute is set. See [section 5.9.6.2, “Job Suspension and Resource Usage”, on page 247](#) and [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).

Format: String of the form: *resource_released_list.<resource name>=<value>,resource_released_list.<resource name>=<value>, ...*

sched_preempt_enforce_resumption

Scheduler attribute. Specifies whether this scheduler creates a special execution priority class for preempted jobs. If so, this scheduler runs these jobs just after any higher-priority jobs. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

Format: *Boolean*

Default: *False*

Python type: *pbs.pbs_resource*

4.9.33.3 How Preemption Works

If preemption is enabled, a scheduler uses preemption as part of its normal pattern of examining each job and figuring out whether or not it can run now. If a job with high preemption priority cannot run immediately, a scheduler looks for jobs with lower preemption priority. A scheduler finds jobs in the lowest preemption level that have been started the most recently. A scheduler preempts these jobs and uses their resources for the higher-priority job. A scheduler tracks resources used by lower-priority jobs, looking for enough resources to run the higher-priority job. If a scheduler cannot find enough work to preempt in order to run a given job, it will not preempt any work.

A job running in a reservation cannot be preempted.

A job's preemption priority is determined by its preemption level.

4.9.33.4 Using Preemption Targets

You can restrict the set of jobs that can be preempted by an entity, by setting that entity's **preempt_targets** resource to a list of jobs and/or queues that can be preempted. This resource is a string array which can contain a list of queues and/or job resources. You specify job resources as *Resource_List.<resource>=<value>*.

Syntax:

```
preempt_targets="Queue=<queue name>[,Queue=<queue name>],Resource_List.<resource
name>=<value>[,Resource_List.<resource name>=<value>]"
```

or

```
preempt_targets=None
```

The **preempt_targets** resource has the following keywords:

Queue=<queue name>

Jobs in the specified queue are eligible to be preempted. "Queue" is case-insensitive.

None

The job, or the jobs at the queue or server whose **preempt_targets** resource is set to *NONE* cannot preempt other jobs. "None" is case-insensitive.

In order for a job to preempt another job, the job to be preempted must have lower preemption priority than the preempting job.

4.9.33.4.i Setting Job Preemption Targets

Preemption targets work as a restriction on which jobs can be preempted by a particular job. If a job has requested **preempt_targets**, a scheduler searches for lower-priority jobs among only the jobs specified in that job's **preempt_targets**. If a job has not requested **preempt_targets**, the scheduler searches among all jobs. For example, if a scheduler is trying to run JobA, and JobA requests **preempt_targets="queue=Queue1,Resource_List.arch=linux"**, JobA is eligible to preempt only those jobs in Queue1 and/or that request **arch=linux**. In addition, JobA can only preempt jobs with lower preemption priority than JobA.

You can prevent a job from preempting any other job in the complex by setting its **preemption_targets** to the keyword "None" (case-insensitive).

You can set `preempt_targets` for a job during submission:

```
-l preempt_targets=...
```

You can set `preempt_targets` via `qalter`:

```
qalter -l preempt_targets=...
```

4.9.33.4.ii Setting Queue Preemption Targets

You can set the default preemption target for jobs in a queue. For example, you can specify that the jobs in a particular queue can preempt the jobs in one or more listed queues:

```
qmgr -c 'set queue <queue name> resources_default.preempt_targets="QUEUE=<queue name>,QUEUE=<queue name>"
```

For example:

```
qmgr -c 'set queue high_prio_queue resources_default.preempt_targets="QUEUE=queueA,QUEUE=queueB"'
```

You can prevent the jobs in a queue which don't explicitly request `preempt_targets` from preempting other jobs by setting the queue's default `preempt_targets` to "NONE":

```
qmgr -c "set queue <queue name> resources_default.preempt_targets=NONE"
```

For example:

```
qmgr -c "set queue lowest_prio_queue resources_default.preempt_targets=NONE"
```

4.9.33.4.iii Setting Default Server Preemption Targets

You can set the default preemption target for jobs at a server. For example, you can specify that the jobs at a server can preempt the jobs in one or more listed queues:

```
qmgr -c 'set server resources_default.preempt_targets="QUEUE=<queue name>,QUEUE=<queue name>"
```

For example:

```
qmgr -c 'set server resources_default.preempt_targets="QUEUE=queueA,QUEUE=queueB"'
```

You can prevent the jobs which don't explicitly request `preempt_targets` from preempting other jobs by setting the server's default `preempt_targets` to "NONE":

```
qmgr -c "set server resources_default.preempt_targets=NONE"
```

For example:

```
qmgr -c "set server resources_default.preempt_targets=NONE"
```

4.9.33.5 Preemption and Job Execution Priority

PBS has an execution class we call *Preempted* for jobs that have been preempted. A scheduler restarts preempted jobs as soon as the preemptor finishes and any other higher-priority jobs finish. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

4.9.33.6 Triggers for Preemption

If preemption is enabled, preemption is used during the following:

- The normal scheduling cycle
- When you run a job via `qrun`

4.9.33.7 Preemption Levels

A preemption level is a class of jobs, where all the jobs in the class share a characteristic. PBS provides built-in preemption levels, and you can combine them or ignore them as you need, except for the *normal_jobs* class, which is required. The built-in preemption levels are listed in the table below.

Table 4-16: Built-in Preemption Levels

Preemption Level	Description
express_queue	Jobs in express queues. See section 4.9.33.7.ii, “The Express Queues Preemption Level”, on page 186
normal_jobs	The preemption level into which a job falls if it does not fit into any other specified level. See section 4.9.33.7.iv, “The Normal Jobs Preemption Level”, on page 186
fairshare	When the entity owning a job exceeds its fairshare limit. See section 4.9.33.7.iii, “The Fair-share Preemption Level”, on page 186
queue_softlimits	Jobs which are over their queue soft limits. See section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 184
server_softlimits	Jobs which are over their server soft limits. See section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 184

You can specify the relative priority of each preemption level, by listing the levels in the desired order in the `preempt_prio` scheduler attribute. Placing a level earlier in the list, meaning to the left, gives it higher priority. For example, if your list is "express_queue", "normal_jobs", "server_softlimits", you are giving the highest priority to jobs in express queues, and the lowest priority to jobs that are over their server soft limits. You can list levels in any order, but be careful not to work at cross-purposes with your execution priority. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

The default value for `preempt_prio` is the following:

```
preempt_prio: "express_queue, normal_jobs"
```

If you do not list a preemption level in the `preempt_prio` scheduler attribute, the jobs in that level are treated like normal jobs. For example, if you do not list `server_softlimits`, then jobs that are over their server soft limits are treated like jobs in the `normal_jobs` level.

You can create new levels that use combinations of the built-in tests. For example, you can define a level which is "*express_queue + server_softlimits*". This level contains jobs that are in express queues and are over their server soft limits. You would probably want to place this level just to the right of the `express_queue` level, meaning that these jobs could be preempted by jobs that are in express queues but are not over their server soft limits.

You can be specific about dividing up jobs: if you want jobs in the express queue to preempt jobs that are also in the express queue but are over their server soft limits, list each level in order:

```
preempt_prio: "express_queue, express_queue+server_softlimits, normal_jobs"
```

However, be careful not to create a runaway effect by placing levels that are over limits before those that are not, for example, `express_queue+server_softlimits` to the left of `express_queue`.

You must list `normal_jobs` in the `preempt_prio` scheduler attribute.

4.9.33.7.i The Soft Limits Preemption Level

You can set a limit, called a *hard limit*, on the number of jobs that can be run or the amount of a resource that can be consumed by a person, a group, or by everyone, and this limit can be applied at the server and at each queue. If you set such a limit, that is the greatest number of jobs that will be run, or the largest amount of the resource that will be consumed.

You can also set a *soft limit* on the number of jobs that can be run or the amount of a resource that can be consumed. This soft limit should be lower than the hard limit, and should mark the point where usage changes from being normal to being "extra, but acceptable". Usage in this "extra, but acceptable" range can be treated by PBS as being lower priority than the normal usage. PBS can preempt jobs that are over their soft limits. The difference between the soft limit and the hard limit provides a way for users or groups to use resources as long as no higher-priority work is waiting.

Example 4-22: Using group soft limits

One group of users, group A, has submitted enough jobs that the group is over their soft limit. A second group, group B, submits a job and are under their soft limit. If preemption is enabled, jobs from group A are preempted until the job from group B can run.

Example 4-23: Using soft limits on number of running jobs

Given the following:

- You have three users, UserA, UserB, and UserC
- Each has a soft limit of 3 running jobs
- UserA runs 3 jobs
- UserB runs 4 jobs
- UserC submits a job to an express queue

This means:

- User C has an express level job, UserA has jobs at the normal level, and UserB has 1 job over the soft limit, so UserB's jobs are over their soft limit and most eligible for preemption by UserC's job

Example 4-24: Using soft limits on amount of resource being used

Given the following:

- Queue soft limit for `ncpus` is 8
- UserA's jobs use 6 CPUs
- UserB's jobs use 10 CPUs

This means:

- UserB is over their soft limit for CPU usage
- UserB's jobs are eligible for preemption

To use soft limits in preemption levels, you must define soft limits. Soft limits are specified by setting server and queue limit attributes. The attributes that control soft limits are:

max_run_soft

Sets the soft limit on the number of jobs that can be running

max_run_res_soft.<resource name>

Sets the soft limit on the amount of a resource that can be consumed by running jobs

Soft limits are enforced only when they are used as a preemption level.

To use soft limits as preemption levels, add their keywords to the `preempt_prio` attribute:

- To create a preemption level for those over their soft limits at the server level, add `"server_softlimits"` to the `preempt_prio` attribute.
- To create a preemption level for those over their soft limits at the queue level, add `"queue_softlimits"` to the `preempt_prio` attribute.
- To create a preemption level for those over their soft limits at both the queue and server, add `"server_softlimits+queue_softlimits"` to the `preempt_prio` attribute.

The jobs of a user or group are over their soft limit only as long as the number of running jobs or the amount of resources used by running jobs is over the soft limit. If some of these jobs are preempted or finish running, and the soft limit is no longer exceeded, the jobs of that user or group are no longer over their soft limit, and no longer in that preemption level. For example, if the soft limit is 3 running jobs, and UserA runs 4 jobs, as soon as one job is preempted and only 3 of UserA's jobs are running, UserA's jobs are no longer over their soft limit.

For a complete description of the use of these attributes, see [section 5.15.1.4, “Hard and Soft Limits”, on page 286](#).

4.9.33.7.ii The Express Queues Preemption Level

The `express_queue` preemption level applies to jobs residing in express queues. An express queue is an execution queue with priority at or above the value set in the `preempt_queue_prio` scheduler attribute. The default value for this parameter is `150`.

Express queues do not require the `by_queue` scheduler parameter to be *True*.

If you will use the `express_queue` preemption level, you probably want to configure at least one express queue, along with some method of moving jobs into it. See [section 2.3, “Queues”, on page 23](#).

If you have more than one express queue, and they have different priorities, you are effectively creating separate sub-levels for express queues. Jobs in a higher-priority express queue have greater preemption priority than jobs in lower-priority express queues.

See [“preempt_queue_prio” on page 255 of the PBS Professional Reference Guide](#).

4.9.33.7.iii The Fairshare Preemption Level

The `fairshare` preemption level applies to jobs owned by entities who are over their fairshare allotment. For example, if each of five users has 20 percent of the fairshare tree, and UserA is using 25 percent of the resources being tracked for fairshare, UserA's jobs become eligible for preemption at the `fairshare` preemption level.

To use the `fairshare` preemption level, you must enable fairshare. See [section 4.9.19, “Using Fairshare”, on page 139](#).

4.9.33.7.iv The Normal Jobs Preemption Level

One special class, `normal_jobs`, is the default class for any job not otherwise specified. If a job does not fall into any of the specified levels, it is placed in `normal_jobs`.

Example 4-25: Normal jobs have the highest priority, then jobs whose entities are over their fairshare limit:

```
preempt_prio: "normal_jobs, fairshare"
```

Example 4-26: `queue_softlimits` jobs whose entities are also over their fairshare limit are lower priority than normal jobs:

```
preempt_prio: "normal_jobs, queue_softlimit+fairshare"
```

4.9.33.8 Selecting Preemption Level

PBS places each job in the most exact preemption level, or the highest preemption level that fits the job.

Example 4-27: We have a job that is express and over its server soft limits. The job is placed in the `"express_queue"` level:

```
preempt_prio: "express_queue, normal_jobs, server_softlimits"
```

Example 4-28: We have a job that is express and over its server soft limits. The job is placed in the `"express_queue+server_softlimits"` level:

```
preempt_prio: "express_queue, express_queue+server_softlimits, normal_jobs, server_softlimits"
```


4.9.33.9 Sorting Within Preemption Level

If there is more than one job within the preemption level chosen for preemption, PBS chooses jobs within that level according to their start time. By default, PBS preempts the job which started running most recently.

For example, if we have two jobs where job A started running at 10:00 a.m. and job B started running at 10:30 a.m:

- The default behavior preempts job B

The allowable value for the `preempt_sort` attribute is `"min_time_since_start"`.

The default value for the `preempt_sort` attribute is `"min_time_since_start"`.

4.9.33.10 Preemption Methods

A scheduler can preempt a job in one of the following ways:

- Suspend the job
- Checkpoint the job
- Requeue the job
- Delete the job

A scheduler tries to preempt a job using the methods listed in the order you specify. This means that if you specify that the order is "checkpoint, suspend, requeue, delete", the scheduler first tries to checkpoint the job, and if it cannot, it tries to suspend the job, and if it cannot do that, it tries to requeue the job, and if it cannot requeue the job, it tries to delete it.

You can specify the order of these attempts in the `preempt_order` scheduler attribute.

The `preempt_order` attribute defines the order of preemption methods which a scheduler uses on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of *S*, *C*, *R*, and *D* (for suspend, checkpoint, requeue, and delete).

The contents is an ordering, for example "SCRD" optionally followed by a percentage of time remaining and another ordering.

The format is a quoted list("").

Example 4-29: PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job:

```
preempt_order: "SR"
```

Example 4-30: If the job has between 100-81% of requested time remaining, first try to suspend the job, then try checkpoint, then requeue. If the job has between 80-51% of requested time remaining, then attempt suspend then checkpoint; and between 50% and 0% time remaining just attempt to suspend the job:

```
preempt_order: "SCR 80 SC 50 S"
```

The default value for `preempt_order` is `"SCR"`.

You cannot repeat a method within a percentage specification. Note that in the example above, the *S* method appears only once per percentage.

4.9.33.10.i Preemption Via Checkpoint

When a job is preempted via checkpointing, MoM runs the `checkpoint_abort` script, and PBS kills and requeues the job. When a scheduler elects to run the job again, it runs the job on the same vnodes as it was originally run on, and the MoM runs the restart script to restart the job from where it was checkpointed.

To preempt via checkpointing, you must define both of the following:

- The checkpointing action in the MoM's `checkpoint_abort` \$action parameter that is to take place when the job is preempted
- The restarting action in the MoM's `restart` \$action parameter that is to take place when the job is restarted

To do this, you must supply checkpointing and restarting scripts or equivalents, and then configure the MoM's `checkpoint_abort` and `restart $action` parameters. Do not use the `$action checkpoint` MoM parameter; it is used when the job should keep running.

See [section 8.3, “Checkpoint and Restart”, on page 387](#).

4.9.33.10.ii Preemption Via Suspension

Jobs are normally suspended via the SIGSTOP signal and resumed via the SIGCONT signal. An alternate suspend or resume signal can be configured in MoM's `$suspendsig` configuration parameter. See [“pbs mom” on page 71 of the PBS Professional Reference Guide](#).

4.9.33.10.iii Suspended Jobs and Resources

Suspended jobs will hold onto some memory and disk space. Suspended jobs may hold application licenses if the application releases them only when it exits. See [Chapter 5, “Job Suspension and Resource Usage”, on page 247](#) and [section 5.9.6.2.iii, “Suspension/resumption Resource Caveats”, on page 248](#).

4.9.33.10.iv Preemption Via Requeue

When a job is preempted and requeued, the job stops execution and is requeued. A requeued job's eligible time is preserved. The amount of time allowed to requeue a job is controlled by the `job_requeue_timeout` server attribute. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

A job that is not eligible to be requeued, meaning a job that was submitted with `-r n`, will not be selected to be preempted via requeue.

4.9.33.10.v Preemption via Deletion

When a job is preempted via deletion, the job is deleted. It is not requeued. Deletion is not in the default preemption order.

4.9.33.11 Enabling Preemption

Preemptive scheduling is enabled by setting a scheduler's attributes and a parameter in that scheduler's configuration file `<sched_priv_directory>/sched_config`.

To enable preemption, you must do the following:

1. Specify the preemption levels to be used by setting `preempt_prio` to desired preemption levels (the default is `"express_queue, normal_jobs"`)

The `preempt_prio` attribute must contain an entry for `normal_jobs`.

2. Optional: specify preemption order by setting `preempt_order`
3. If you will use the `fairshare` preemption level, configure `fairshare`. See [section 4.9.19, “Using Fairshare”, on page 139](#).
4. If you will use the `server_softlimits` and/or `queue_softlimits` preemption levels, configure server and/or queue soft limits. See [section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 184](#).
5. Enable preemption by setting `preemptive_sched` to `True`. It is `True` by default.
6. Choose whether to use preemption during primetime, non-primetime, or all of the time. The default is ALL. If you want separate behavior for primetime and non-primetime, specify each separately. For example:

```
preemptive_sched: True prime
preemptive_sched: False non_prime
```

4.9.33.12 Preemption Example

Example 4-31: To configure a scheduler for the following preemption priority:

- a. Express queue jobs at the highest preemption priority
- b. followed by jobs that are express but whose user/group is over a soft limit,
- c. then normal jobs,
- d. and last, jobs belonging to users/groups over their server soft limit (not in express queues)

We turn on preemptive scheduling in the scheduler's configuration file:

```
preemptive_sched:      TRUE ALL
```

We set scheduler attributes:

- To set the queue priority level for express queues:

```
qmgr -c 'set sched <scheduler name> preempt_queue_prio=150'
```

- To set the preemption priority order:

```
qmgr -c 'set sched <scheduler name> preempt_prio="express_queue,
express_queue+server_softlimits, normal_jobs, server_softlimits"'
```

We specify when to use each preemption method.

If the first method fails, try the next method.

If a job has between 100-81% time remaining, try to suspend, then checkpoint then requeue.

From 80-51% suspend and then checkpoint, but don't requeue.

If between 50-0% time remaining, then just suspend it.

```
qmgr -c 'set sched <scheduler name> preempt_order="SCR 80 SC 50 S"'
```

4.9.33.13 Preemption Caveats and Recommendations

- It is not advisable to use preemption via deletion with a `runjob` hook. Jobs are preempted before the high-priority job is run, whether or not that job actually runs. Job deletion happens before a `runjob` hook would execute. Even if the `runjob` hook rejects the high-priority job, the preempted jobs are still deleted.
- When using any of the fairshare, soft limits, or express queue preemption levels, be sure to enable the corresponding PBS feature. For example, when using preemption with the fairshare preemption level, be sure to turn fairshare on. Otherwise, you will be using stale fairshare data to preempt jobs.
- It's important to be careful about the order of the preemption levels and the sizes of the limits at queue and server. For example, if you make users who are over their server soft limits have higher priority than users who are over their queue soft limits, and you set the soft limit higher at the server than at the queue, you can end up with users who have more jobs running preempting users who have fewer jobs running.

In this example, a user with more jobs preempts a user with fewer jobs.

- If a subjob is not running because its array job has hit the limit in `max_run_subjobs`, the subjob is not eligible to start and PBS does not try to use preemption to start the subjob.
- Beware of setting up situations where a user running more jobs than they should can preempt users who are running fewer jobs. Given the following:
 - `preempt_prio` attribute contains "`server_softlimits, queue_softlimits`"
 - Server soft limit is 5
 - Queue soft limit is 3
 - User1 has 6 jobs running
 - User2 has 4 jobs running

This means:

- Both users are over the queue soft limit, and User1 is over the server soft limit
- User1 has higher priority, so User1's jobs can preempt User2's jobs

To avoid this scenario, you could set the `preempt_prio` attribute to contain "`server_softlimits, queue_softlimits, server_softlimits+queue_softlimits`". In this case User1 would have lower priority, because User1 is over both soft limits.

- Preemption priority is mostly independent of execution priority. You can list preemption levels in any order in `preempt_prio`, but be careful not to work at cross-purposes with your execution priority. Be sure that you are not preempting jobs that have higher execution priority. See [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- If a high-priority job has been selected to preempt lower-priority jobs, but is rejected by a runjob hook, a scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.
- A job that has requested an AOE will not preempt another job, regardless of whether the job's requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOE's.
- When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. See [section 8.6.3, “Setting Job Requeue Timeout”, on page 414](#).
- When you issue "`qrun <job ID>`", without the `-H` option, the selected job has the highest preemption priority, for that scheduling cycle. However, at the following scheduling cycle, the preemption priority of the selected job returns to whatever it would be without `qrun`. If you give the `qrun` command multiple job IDs, each job is run in its own scheduling cycle.
- When `sched_preempt_enforce_resumption` is set to *True*, all preempted jobs become top jobs, regardless of their setting for `topjob_ineligible`.
- PBS will not use suspension or checkpointing to preempt a job that requests a value for `ee`.

4.9.34 Using Primetime and Holidays

Often it is useful to run different scheduling policies for specific intervals during the day or work week. PBS provides a way to specify two types of interval, called *primetime* and *non-primetime*.

Between them, primetime and non-primetime cover all time. There is no time slot that is neither primetime nor non-primetime. This includes dedicated time. Primetime and/or non-primetime overlap dedicated time.

You can use non-primetime for such tasks as running jobs on desktop clusters at night.

4.9.34.1 How Primetime and Holidays Work

By default, primetime is 24/7. A scheduler looks in the `<sched_priv directory>/holidays` file for definitions of primetime, non-primetime, and holidays. You can edit this file to define your holidays and primetime.

Many PBS scheduling parameters can be specified separately for primetime, non-primetime, or all of the time. This means that you can use, for example, fairshare during primetime and no fairshare during non-primetime. These parameters have a time slot default of all, meaning that if enabled, they are in force all of the time.

A scheduler applies the parameters defined for primetime during the primetime time slots, and applies parameters defined for non-primetime during the non-primetime time slots. Any scheduler parameters defined for all time are run whether it is primetime or not.

Any holidays listed in the holidays file are treated as non-primetime. To have a holiday treated like a normal workday or weekend, do not list it in the holidays file.

There are default behaviors for primetime and non-primetime, but you can set up the behavior you want for each type. The names "primetime" and "non-primetime" are meant to be informative, but they are arbitrary. The default for primetime is 24/7, meaning that primetime is all of the time by default. Example holidays are provided, but commented out, in the `holidays` file.

You can define primetime and non-primetime queues. Jobs in these queues can run only during the designated time. Queues that are not defined specifically as primetime or non-primetime queues are called "anytime queues".

4.9.34.2 Configuring Primetime and Non-primetime

In order to use primetime and non-primetime, you must have a `holidays` file with the current year in it.

You can specify primetime and non-primetime time slots by specifying them in the `<sched_priv_directory>/holidays` file.

The format of the primetime and non-primetime section of the `holidays` file is the following:

```
YEAR YYYY
<day> <prime> <nonprime>
<day> <prime> <nonprime>
```

In `YEAR YYYY`, `YYYY` is the current year.

`Day` can be *weekday*, *monday*, *tuesday*, *wednesday*, *thursday*, *friday*, *saturday*, or *sunday*.

Each day line must have all three fields.

Any line that begins with a `"*"` or a `"#"` is a comment.

Weekday names must be lowercase.

The ordering of elements in this file is important. The ordering of `<day>` lines in the `holidays` file controls how primetime is determined. A later line takes precedence over an earlier line.

For example:

```
weekday      0630    1730
friday       0715    1600
```

means the same as

```
monday       0630    1730
tuesday      0630    1730
wednesday    0630    1730
thursday     0630    1730
friday       0715    1600
```

However, if a specific day is followed by "weekday",

```
friday       0700    1600
weekday      0630    1730
```

the "weekday" line takes precedence, so Friday will have the same primetime as the other weekdays.

Times can be expressed as one of the following:

- `HHMM` with no colons(:)
- The word "all"
- The word "none"

4.9.34.3 Configuring Holidays

You can specify primetime and non-primetime time slots by specifying them in the `<sched_priv directory>/holidays` file.

You must specify the year, otherwise primetime is in force at all times, and PBS will not recognize any holidays. Specify the year here, where YYYY is the current year:

YEAR YYYY

Holidays are specified in lines of this form:

<day of year> <month day-of-month> <holiday name>

PBS uses the *<day of year>* field and ignores the *<date>* string.

Day of year is the julian day of the year between 1 and 365 (e.g. "1").

Month day-of-month is the calendar date, for example "Jan 1".

Holiday name is the name of the holiday, for example "New Year's Day".

4.9.34.4 Example of holidays File

```

YEAR      2020
*
* Prime    Non-Prime
* Day      Start    Start
*
weekday    0600    1730
saturday   none    all
sunday     none    all
*
* Day of    Calendar    Company Holiday
* Year      Date        Holiday
  1         Jan 1       New Year's Day
  20        Jan 20      Martin Luther King Day
  48        Feb 17      Presidents Day
  146       May 25      Memorial Day
  186       Jul 4       Independence Day
  251       Sep 7       Labor Day
  286       Oct 12      Columbus Day
  316       Nov 11      Veterans Day
  331       Nov 26      Thanksgiving
  360       Dec 25      Christmas Day

```

4.9.34.5 Reference Copy of holidays File

A reference copy of the holidays file contains example holidays that are commented out. It is provided in `PBS_EXEC/etc/pbs_holidays`. The file looks like this:

```
* UNCOMMENT AND CHANGE THIS TO THE CURRENT YEAR
*YEAR 1970
*
* Prime/Nonprime Table
*
* Prime Non-Prime
* Day Start Start
*
* UNCOMMENT AND SET THE REQUIRED PRIME/NON-PRIME START TIMES
* weekday 0600 1730
* saturday none all
* sunday none all
*
* Day of Calendar Company
* Year Date Holiday
*

* UNCOMMENT AND ADD CALENDAR HOLIDAYS TO BE CONSIDERED AS NON-PRIME DAYS
* 1 Jan 1 New Year's Day
* 359 Dec 25 Christmas Day
```

4.9.34.6 Defining Primetime and Non-primetime Queues

Jobs in a primetime queue can start only during primetime. Jobs in a non-primetime queue can start only during non-primetime. Jobs in an anytime queue can start at any time.

You define a primetime queue by naming it using the primetime prefix. The prefix is defined in the `primetime_prefix` scheduler parameter. The default is `"p_"`. For example, you could name a primetime queue `"p_queueA"`, using the default.

Similarly, you define a non-primetime queue by prefixing the name. The prefix is defined in the `nonprimetime_prefix` scheduler parameter, and defaults to `"np_"`.

4.9.34.7 Controlling Whether Jobs Cross Primetime Boundaries

You can control whether jobs are allowed to start running in one time slot and finish in another, for example when job A starts during primetime and finishes a few minutes into non-primetime. When a job runs past the boundary, it delays the start of a job that is constrained to run only in the later time slot. For example, if job B can run only during non-primetime, it may have to wait while job A uses up non-primetime before it can start. You can control this behavior for all queues, or you can exempt anytime queues, controlling only primetime and non-primetime queues. You can also specify how much time past the boundary a job is allowed to run.

To prevent a scheduler from starting any jobs which would run past a primetime/non-primetime boundary, set the `backfill_prime` scheduler parameter to `True`. You can specify this separately for primetime and non-primetime. If you specify it for one type of time slot, it prevents those jobs from crossing the next boundary. For example, if you set the following:

```
backfill_prime True prime
```


jobs in primetime slots are not allowed to cross into non-primetime slots.

If you set the following:

```
backfill_prime True non_prime
```

jobs in non-primetime slots are not allowed to cross into primetime slots.

To exempt jobs in anytime queues from the control of `backfill_prime`, set the `prime_exempt_anytime_queues` scheduler parameter to *True*. This means that jobs in an anytime queue are not prevented from running across a prime-time/nonprimetime or non-primetime/primetime boundary.

To allow jobs to spill over a certain amount of time past primetime/non-primetime boundaries, but no more, specify this amount of time in the `prime_spill` scheduler parameter. You can specify separate behavior for primetime and non-primetime jobs. For example, to allow primetime jobs to spill by 20 minutes, but only allow non-primetime jobs to spill by 1 minute:

```
prime_spill 00:20:00 prime
prime_spill 00:01:00 non_prime
```

The `prime_spill` scheduler parameter applies only when `backfill_prime` is *True*.

4.9.34.8 Logging

A scheduler logs a message at the beginning of each scheduling cycle indicating whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log level 0x0100. The message is of this form:

```
"It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```

or

```
"It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```

4.9.34.9 Scheduling Parameters Affecting Primetime

`backfill_prime`

This scheduler will not run jobs which would overlap the boundary between primetime and non-primetime.

Format: *Boolean*

Default: *False all*

`nonprimetime_prefix`

Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime.

Format: *String*

Default: *np_*

`primetime_prefix`

Queue names starting with this prefix are treated as primetime queues. Jobs will only run in these queues during primetime.

Format: *String*

Default: *p_*

prime_exempt_anytime_queues

Determines whether anytime queues are controlled by `backfill_prime`.

If set to *True*, jobs in an anytime queue will not be prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an anytime queue may not cross this boundary, except for the amount specified by their `prime_spill` setting.

Format: *Boolean*

Default: *False*

prime_spill

Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for prime- and non-primetime. This option is only meaningful if `backfill_prime` is *True*.

Format: *Duration*

Default: *00:00:00*

4.9.34.10 Caveats for Primetime and Holidays

- In order to use primetime and non-primetime, you must have a `holidays` file with the current year in it. If there is no `holidays` file with a year in it, primetime is in force all of the time.
- You cannot combine `holidays` files.
- If you use the formula, it is in force all of the time.
- If there is no *YEAR* line in the holidays file, primetime is in force at all times. If there is more than one *YEAR* line, the last one is used.
- If the information for any day is missing or incorrect, primetime is in force for all of that day.

4.9.35 Provisioning

PBS provides automatic provisioning of an OS or application, on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

For a complete description of how provisioning works and how to configure it, see [Chapter 16, "Provisioning", on page 591](#).

4.9.36 Queue Priority

Queues and queue priority play several different roles in scheduling, so this section contains pointers to other sections.

Each queue can have a different priority. A higher value for priority means the queue has greater priority. By default, queues are sorted from highest to lowest priority. Jobs in the highest priority queue will be considered for execution before jobs from the next highest priority queue. If queues don't have different priority, queue order is undefined.

Each queue's priority is specified in its `priority` attribute. By default, the queue `priority` attribute is unset. There is no limit to the priority that you can assign to a queue, however it must fit within integer size. See [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

4.9.36.1 Configuring Queue Priority

You can specify the priority of each queue by setting a value for its `priority` attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

4.9.36.2 Using Queue Priority

You can configure a scheduler so that job execution or preemption priority is partly or entirely determined by the priority of the queue in which the job resides. Queue priority can be used for the following purposes:

- Queue priority can be used as a term in the job sorting formula. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#)
- Queue priority can be used to specify the order in which queues are examined when scheduling jobs. If you want jobs to be examined queue by queue, in order of queue priority, you must specify a different priority for each queue. A queue with a higher value is examined before a queue with a lower value. See [section 4.3.5.4.i, “Using Queue Order to Affect Order of Consideration”, on page 68](#)
- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in a scheduler, see [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).
- You can set up preemption levels that include jobs in express queues. For information on preemption, see [section 4.9.33, “Using Preemption”, on page 180](#).

A queue is an express queue if its priority is greater than or equal to the value that defines an express queue. For more about using express queues, see [section 4.9.18, “Express Queues”, on page 139](#).

4.9.36.3 Queue Priority Caveats

- If you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the new queue to which the job is moved.

4.9.37 Reservations

PBS provides a way to reserve specific resources for a defined time period. If you want reservations in which to run jobs, you can make one-time reservations (called *advance reservations*), or you can make a series of reservations (called *standing reservations*), where each one is for the same resources, but for a different time period. Or, if you want to secure resources for a specific (perhaps troublesome) job, you can create a *job-specific reservation* for that job at submission time, while the job is queued, or later while the job is running.

If you want to sequester hosts for maintenance, you can create a *maintenance reservation*. Maintenance reservations block out time on specified machines, preventing jobs from being started where you need to perform maintenance tasks.

Advance, standing, and job-specific reservations are “job reservations”, to distinguish them from maintenance reservations.

Reservations are useful for accomplishing the following job-related tasks:

- To get a time slot on a specific host
- To run a job in a specific time slot, meaning at or by a specific time
- To be sure a job will run
- To have a high-priority job run soon
- To make sure that a job doesn't lose access to resources when needs to be re-run

4.9.37.1 Definitions

Advance reservation

A reservation for a set of resources for a specified time. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

Degraded reservation

A job-specific or advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with the soonest occurrence are unavailable.

Job-specific reservation

A reservation created for a specific job, for the same resources that the job requested.

Job-specific ASAP reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS schedules the reservation to run as soon as possible, and PBS moves the job into the reservation. Created when you use `pbs_rsub -Wqmove=<job ID>` on a queued job.

Job-specific now reservation

Reservation created for a specific running job. PBS immediately creates a job-specific now reservation on the same resources as the job is using, and moves the job into the reservation. The reservation is created and starts running immediately when you use `pbs_rsub --job <job ID>` on a running job.

Job-specific start reservation

Reservation created for a specific job, for the same resources the job requests. PBS starts the job according to scheduling policy. When the job starts, PBS creates and starts the reservation, and PBS moves the job into the reservation. Created when you use `qsub -Wcreate_resv_from_job=true` to submit a job or when you `qalter` a job to set the job's `create_resv_from_job` attribute to *True*.

Maintenance reservation

A reservation designed for performing maintenance on the specified hosts for the specified time. Created using `pbs_rsub --hosts <host list>`.

Occurrence of a standing reservation

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See [“qsub” on page 216 of the PBS Professional Reference Guide](#).
- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the `resv_nodes` attribute reported by `pbs_rstat`.

Also called an *instance* of a standing reservation.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Standing reservation

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

4.9.37.2 Job Reservations

4.9.37.2.i Creating Advance and Standing Reservations

Any PBS user can create both advance and standing reservations for jobs using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<sequence number>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<sequence number>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job.

See ["Reserving Resources", on page 137 of the PBS Professional User's Guide](#), for detailed information on creation and use of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

4.9.37.2.ii Job-Specific Reservations

A job-specific reservation is for the same resources that the job requested. Job-specific reservations are intended to pre-serve access to the job's resources in the case where a job may need to be modified and then re-run, so that the job does not need to wait to be re-scheduled.

Any PBS user can create a job-specific reservation.

A job-specific reservation ID has the format:

R<sequence number>.<server name>

Job-specific reservations cannot be used with job arrays.

4.9.37.2.iii Creating Job-specific Start Reservations

Job submitters can create a job-specific start reservation at submission time. The job is scheduled normally, and when it starts, PBS creates and starts a reservation on the same resources, and puts the job into the reservation. To create a job-specific reservation at submission time, set the job's `create_resv_from_job` attribute to *True*:

qsub ... -Wcreate_resv_from_job=1

To create a job-specific start reservation from a queued job, use `qalter` to set the `create_resv_from_job` attribute to *True*.

4.9.37.2.iv Creating Job-specific ASAP Reservations

Job submitters can create a job-specific ASAP from a queued job. PBS creates the reservation for the same resources the job requests, moves the job into the reservation, and schedules the reservation to start as soon as possible.

To create a job-specific ASAP reservation:

pbs_rsub -Wqmove=<job ID>

Note that job-specific ASAP reservations, once created, do not adjust themselves to a change in resource availability. An ASAP reservation can cause resources to go idle while waiting for the reservation to start. For example, if a job scheduled to finish before an ASAP reservation finishes early, and no jobs can be backfilled into the new open slot, resources will sit idle until the reservation runs. In addition, if a high-priority job comes in after an ASAP reservation has been created for a lower-priority job, the high-priority job must wait until after the reservation finishes.

To get the equivalent of flexible ASAP reservations that don't cause idle resources, use a job sort formula with a custom priority term, for example "cust_high_pri", and set this term to a high value, for example 10, for the desired job. Then you can alter the job: `qalter -l cust_high_pri=10 -wcreate_resv_from_job=true`. Make sure that `cust_high_pri` has a large enough coefficient in the formula to change the job priority.

4.9.37.2.v Creating Job-specific Now Reservations

Job submitters can create a job-specific now from a running job. PBS creates the reservation for the same resources the job is using, starts the reservation, and moves the job into the reservation.

To create a job-specific now reservation:

```
pbs_rsub --job <job ID>
```

4.9.37.2.vi Job Reservations and Placement Sets

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See [section 4.9.32.4.ii, "Order of Placement Set Consideration Within Pool", on page 172](#).

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored, if the scheduler's `do_not_span_psets` attribute is *False*. If no placement set will satisfy the reservation, and the scheduler's `do_not_span_psets` attribute is *True*, the reservation is not created.

The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

4.9.37.2.vii Requesting Resources for Job Reservations

Reservations request resources using the same mechanism that jobs use. If a resource is unrequestable, users cannot request it for a reservation. If a resource is invisible, users cannot view it or request it for a reservation.

4.9.37.2.viii Job Reservations and Provisioning

Users can create reservations that request AOE. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE. See [section 16.4.3, "Provisioning And Reservations", on page 595](#).

The vnodes allocated to a reservation that requests an AOE are put in the *resv-exclusive* state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

For information on restrictions applying to reservations used with provisioning, see [section 16.7.2.3, "Vnode Reservation Restrictions", on page 608](#).

For how to avoid problems with provisioning and reservations, see [section 16.10.1, "Using Provisioning Wisely", on page 617](#).

4.9.37.2.ix Job Reservation Priority

A job running in a reservation cannot be preempted.

A job running in a reservation has the highest execution priority.

4.9.37.2.x Querying Reservations

To query a reservation, use the `pbs_rstat` command. See ["Viewing the Status of a Reservation", on page 146 of the PBS Professional User's Guide](#). To delete a reservation, use the `pbs_rdel` command, not the `qmgr` command.

4.9.37.2.xi Controlling Access to Job Reservations

You can specify which projects, users, and groups can and cannot submit jobs to reservations. Use the `pbs_rsub -U/-G` command to set the reservation's `acl_users` and/or `acl_groups` attributes, and `pbs_ralter -U/-G` to change them. See [section 11.3.8, "Reservation Access", on page 501](#).

4.9.37.2.xii Job Reservation Fault Tolerance

PBS automatically keeps track of the vnodes assigned to reservations, and tries to find replacement vnodes for those that become unavailable. See [section 8.4, "Reservation Fault Tolerance", on page 401](#).

4.9.37.2.xiii Logging Standing Reservation Information

The start and end of each occurrence of a standing reservation is logged as if each occurrence were a single advance reservation.

Reservation-related messages are logged at level `PBSEVENT_RESV`, which is `0x0200` (512).

4.9.37.2.xiv Accounting

Resources requested for a reservation are recorded in the reservation's `Resource_List` attribute, and reported in the accounting log `B` and `Y` records for the reservation. See [section 12.5.3, "Timeline for Reservation Accounting Messages", on page 549](#).

4.9.37.3 Maintenance Reservations

You can create maintenance reservations using `pbs_rsub --hosts <host list>`. Maintenance reservations are designed to make the specified hosts available for the specified amount of time, regardless of what else is happening:

- You can create a maintenance reservation that includes or is made up of vnodes that are down or offline.
- Maintenance reservations ignore the value of a vnode's `resv_enable` attribute.
- PBS immediately confirms any maintenance reservation.
- Maintenance reservations take precedence over other reservations; if you create a maintenance reservation that overlaps an advance or standing job reservation, the overlapping vnodes become unavailable to the job reservation, and the job reservation is in conflict with the maintenance reservation. PBS looks for replacement vnodes; see ["Reservation Fault Tolerance" on page 401 in the PBS Professional Administrator's Guide](#).

PBS will not start any new jobs on vnodes overlapping or in a maintenance reservation. However, jobs that were already running on overlapping vnodes continue to run; you can let them run or requeue them.

You cannot specify place or select for a maintenance reservation; these are created by PBS:

- PBS creates the reservation's placement specification so that hosts are assigned exclusively to the reservation. The placement specification is always the following:

`-lplace=exclhost`

- PBS sets the reservation's `resv_nodes` attribute value so that all CPUs on the reserved hosts are assigned to the maintenance reservation. The select specification is always the following:

`-lselect=host=<host1>:ncpus=<number of CPUs at host1>+host=<host2>:ncpus=<number of CPUs at host2>+...`

Maintenance reservations are prefixed with *M*. A maintenance reservation ID has the format:

M<sequence number>.<server name>

You cannot create a recurring maintenance reservation.

Creating a maintenance reservation does not trigger a scheduling cycle.

You must have manager or operator privilege to create a maintenance reservation.

4.9.37.4 Modifying Reservations

You can use the `pbs_ralter` command to alter an existing reservation, whether it is an individual job-specific or advance reservation, or the next or current instance of a standing reservation. Syntax:

```
pbs_ralter [-D <duration>] [-E <end time>] [-G <auth group list>] [-I <block time>] [-l select=<select spec>] [-m
  <mail points>] [-M <mail list>] [-N <reservation name>] [-R <start time>] [-U <auth user list>] <reservation
  ID>
```

To force a change to the start time, end time, or duration of a reservation, you can use the `-Wforce` option:

```
pbs_ralter -Wforce [-D <duration>] [-E <end time>] [-R <start time>] <reservation ID>
```

Note that with the `-Wforce` option you can force PBS to oversubscribe resources, in which case you (the administrator) may need to manage them yourself.

You can modify an advance or standing reservation so that if the reservation sits idle, it is automatically deleted after the amount of time you specify. For a standing reservation, this applies to each occurrence separately. If one occurrence of a standing reservation is deleted, the next occurrence still starts at its designated time. To have a reservation be deleted automatically, use `pbs_ralter -Wdelete_idle_time=<allowed idle time>` and specify the number of seconds as an integer, or the duration as `HH:MM:SS`. Note that you cannot change any other reservation attributes when you change this one.

You cannot change the start time of a reservation in which jobs are running.

When changing the select specification, the behavior depends on whether there are jobs running.

- If jobs are running in the reservation:
 - You cannot release chunks where reservation jobs are running
 - Vnodes where jobs are running cannot change, but all other vnodes can change
- If no jobs are running, the select specification can be changed completely

When requesting chunks, make sure each chunk request specifies chunks of a single type.

To find unused chunks in a running reservation, you can compare the reservation's `resv_nodes` attribute to the `exec_vnode` attribute of the jobs running in the reservation.

If the reservation has started and is degraded, you must release all unavailable chunks in order to alter the reservation select specification.

If the reservation has not started, modifying the select specification may result in moving the reservation to different vnodes.

After the change is requested, the change is either confirmed or denied. On denial of the change, the reservation is not deleted and is left as is, and the following message appears in the server's log:

```
Unable to alter reservation <reservation ID>
```

When a reservation is confirmed, the following message appears in the server's log:

```
Reservation alter successful for <reservation ID>
```

To find out whether or not the change was allowed:

- Use the `pbs_rstat` command: see whether you altered reservation attribute(s)
- Use the interactive option: check for confirmation after the blocking time has run out

If the reservation has not started and it cannot be confirmed on the same vnodes, PBS searches for another set of vnodes. See [section 8.4, “Reservation Fault Tolerance”, on page 401](#).

You must be the reservation owner or the PBS Administrator to run this command.

For details, see [“pbs_ralter” on page 85 of the PBS Professional Reference Guide](#).

4.9.37.4.i Examples of Modifying Reservations

Example 4-32: Grow a reservation:

```
Existing:
select=100:ncpus=20:mem=512gb
pbs_ralter -l select=150:ncpus=20:mem=512gb
```

Example 4-33: Grow and shrink a reservation:

```
Existing:
select=100:ncpus=20+10:ncpus=10:mem=512gb
pbs_ralter -l select=150:ncpus=20+5:ncpus=10:mem=512gb
```

Example 4-34: Grow a reservation, and get rid of a type of chunk:

```
Existing:
select=100:ncpus=20+10:ncpus=10:mem=512MB+15:ncpus=40
pbs_ralter -l select=150:ncpus=20+30:ncpus=40
```

Example 4-35: No running jobs; change select completely:

```
Existing:
select=100:ncpus=20+10:ncpus=10:mem=512GB
pbs_ralter -l select=150:ncpus=20:mem=1024GB+5:ncpus=15:mem=512GB
```

Example 4-36: Job is running on 50 vnodes of the first type of chunk; grow and shrink reservation:

```
Existing:
select=100:ncpus=20+50:ncpus=40
pbs_ralter -l select=50:ncpus=20+100:ncpus=40
```

Example 4-37: Negative example. With job running on 50 vnodes on the first type of chunk, we try to do an invalid alteration by trying to remove chunks from running jobs:

```
Existing:
select =100:ncpus=20+50:ncpus=40
pbs_ralter -l select=25:ncpus=20+100:ncpus=40
ALTER DENIED
```

4.9.37.5 Attributes Affecting Reservations

We list the server, vnode, and job attributes affecting reservations here. See the full list of reservation attributes in ["Reservation Attributes" on page 304 in the PBS Professional Administrator's Guide](#). See ["Server Attributes" on page 281 of the PBS Professional Reference Guide](#) and ["Vnode Attributes" on page 321 of the PBS Professional Reference Guide](#).

Table 4-17: Attributes Affecting Reservations

Entity	Attribute	Effect
Server	acl_resv_host_enable	Controls whether or not the server uses the <code>acl_resv_hosts</code> access control lists.
Server	acl_resv_hosts	List of hosts from which reservations may and may not be created at this server.
Server	acl_resv_group_enable	Controls whether or not the server uses the <code>acl_resv_groups</code> access control lists.

Table 4-17: Attributes Affecting Reservations

Entity	Attribute	Effect
Server	acl_resv_groups	List of groups who may and may not create reservations at this server.
Server	acl_resv_user_enable	Controls whether or not the server uses the <code>acl_resv_users</code> access control lists.
Server	acl_resv_users	List of users who may and may not create reservations at this server.
Server	resv_enable	Controls whether or not reservations can be created at this server.
Server	reserve_retry_time	Length of time to wait between when a reservation becomes degraded and when PBS tries to reconfirm the reservation, as well as interval between attempts to reconfirm a degraded reservation. Default: 600 (10 minutes)
Vnode	queue deprecated	Associates the vnode with an execution queue. If this attribute is set, this vnode cannot be used for reservations.
Vnode	resv_enable	Controls whether the vnode can be used for reservations. Default is <i>True</i> , but set to <i>False</i> for a vnode used for cycle harvesting.
Job	create_resv_from_job	Controls whether PBS creates a job-specific reservation for this job.

4.9.37.6 Reservation Advice and Caveats

- Do not attempt to alter a maintenance reservation.
- Do not delete a reservation's queue.
- Do not start a reservation's queue (do not set the reservation's **started** attribute to *True*). Jobs will run prematurely.
- Do not try to set attribute values for a reservation queue directly; instead, operate on the reservation.
- Reservations are incompatible with cycle harvesting. Do not allow reservations on machines used for cycle harvesting. The user may begin using the machine, which will suspend any PBS jobs, possibly preventing them from finishing before the reservation runs out. Set each cycle harvesting vnode's **resv_enable** attribute to *False*, to prevent the vnode from being used for reservations.
- You can write hooks that execute, modifying a reservation's attributes, when a reservation is created. See the PBS Professional Hooks Guide.
- Allow enough time in reservations. If a job is submitted to a reservation with a duration close to the **walltime** of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.
- Hosts or vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance reservations. Hosts or vnodes that are being used for cycle harvesting should not be used for reservations.
- Hosts with **\$max_load** and **\$ideal_load** configured should not be used for reservations. Set the **resv_enable** vnode attribute on these hosts to *False*.
- For troubleshooting problems with reservations, see ["Reservation Caveats and Errors", on page 150 of the PBS Professional User's Guide](#).
- Be careful when using `qrun -H` on jobs or vnodes involved in reservations. Make sure that you don't oversubscribe reserved resources.
- In order to create reservations, the submission host must have its timezone set to a value that is understood by the PBS server. See [section 20.9.5, "Unrecognized Timezone Variable", on page 647](#).
- Avoid making reservations for resources that are out of the control of PBS. Resources that are managed through a `server_dyn_res` script may not be available when jobs need them.
- If you create a maintenance reservation that overlaps an advance or standing job reservation, the maintenance reservation takes precedence, the overlapping vnodes become unavailable to the job reservation, and the job reservation becomes degraded. PBS looks for replacement vnodes; see [section 8.4, "Reservation Fault Tolerance", on page 401](#). Any job reservation overlapping a maintenance reservation goes into the **RESV_IN_CONFLICT** substate (12).
- Note that job-specific ASAP reservations, once created, do not adjust themselves to a change in resource availability. An ASAP reservation can cause resources to go idle while waiting for the reservation to start. For example, if a job scheduled to finish before an ASAP reservation finishes early, and no jobs can be backfilled into the new open slot, resources will sit idle until the reservation runs.
- To get the equivalent of flexible ASAP reservations that don't cause idle resources, use a job sort formula with a custom priority term, for example `"cust_high_pri"`, and set this term to a high value, for example 10, for the desired job. Then you can alter the job: `qalter -l cust_high_pri=10 -Wcreate_resv_from_job=true`.
- Beware of oversubscribing resources when using the `-Wforce` option to `pbs_ralter`.

4.9.38 Round Robin Queue Selection

PBS can select jobs from execution queues by examining the queues in round-robin fashion. The behavior is round-robin only when you have groups of queues where all queues in each group have the same priority.

The order in which queues are selected is determined by each queue's priority. You can set each queue's priority; see [section 2.3.5.3, “Prioritizing Execution Queues”, on page 27](#). If queue priorities are not set, they are undefined. If you do not prioritize the queues, their order is undefined.

When you have multiple queues with the same priority, a scheduler round-robins through all of the queues with the same priority as a group. So if you have Q1, Q2, and Q3 at a priority of 100, Q4 and Q5 at a priority of 50, and Q6 at a priority of 10, a scheduler will round-robin through Q1, Q2, and Q3 until all of those jobs are out of the way, then the scheduler will round-robin through Q4 and Q5 until there are no more jobs in them, and finally the scheduler will go through Q6.

When using the round-robin method with queues that have unique priorities, a scheduler runs all jobs from the first queue, then runs all the jobs in the next queue, and so on.

To specify that PBS should use the round-robin method to select jobs, set the value of the `round_robin` scheduler parameter to *True*.

The `round_robin` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

You can use the round-robin method as a resource allocation tool. For example, if you need to run the same number of jobs from each group, you can put each group's jobs in a different queue, and then use round-robin to run jobs, one from each queue.

The round-robin method is also used in PBS for a feature that is not controlled by the `round_robin` scheduler attribute. The SMP cluster distribution parameter, `smp_cluster_dist`, can use a round-robin method to place jobs. See [section 4.9.43, “SMP Cluster Distribution”, on page 217](#).

See [“round_robin” on page 257 of the PBS Professional Reference Guide](#).

4.9.38.1 Round-robin Caveats

- Each scheduling cycle starts with the highest-priority queue. Therefore, when using round-robin, this queue gets preferential treatment.
- When set to *True*, the `round_robin` parameter overrides the `by_queue` parameter.
- If round robin and strict ordering are *True*, and backfilling is not being used, and the top job cannot run, whether because of resources or rejection by MoM, no job runs. However, if round robin is *True* and strict ordering is *False*, and the top job in the current queue cannot run, the next top job is considered instead. For example, we have 3 queues, each with 3 jobs, and with the same priority:

Q1: J1 J2 J3

Q2: J4 J5 J6

Q3: J7 J8 J9

If `round_robin` and `strict_ordering` are *True*, and J1 cannot run, no job runs.

If `round_robin` is *True* and `strict_ordering` is *False*, and J1 cannot run, job order is J4, J7, J2, J5, J8, J3, etc.

- With `round_robin` and `strict_ordering` set to *True*, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job with a start time at some later time to allow other jobs in the same queue to be run.

4.9.39 Routing Jobs

Before reading this section, please read about the mechanics of configuring and using routing queues, in [section 2.3.6, “Routing Queues”, on page 27](#).

In this section, we use the term "routing" to mean the general process of moving a job somewhere, whether it is from one queue to another, from one partition or complex to another, or from a queue to particular vnodes.

Routing jobs can involve collecting jobs so they don't stray into the wrong queues, moving those jobs to the correct queues, and filtering which jobs are allowed into queues.

You may need to collect jobs into a routing queue, before moving them to the correct destination queue. If you use a routing queue, you can force users to submit jobs to the routing queue only, you can grab jobs as they are submitted and put them in the routing queue, and you can set a routing queue as the default. The mechanisms to collect jobs are described below, and listed here:

- Setting default queue; see [section 4.9.39.1.i, “Default Queue as Mechanism to Collect Jobs”, on page 206](#)
- Grabbing jobs upon submission; see [section 4.9.39.1.ii, “Grabbing Jobs Upon Submission”, on page 207](#)
- Disallowing direct submission to execution queues; see [section 4.9.39.1.iii, “Disallowing Direct Submission as Mechanism to Collect Jobs”, on page 207](#)
- Disallowing submission using access controls; see [section 4.9.39.3.ii, “Access Controls as Filtering Mechanism”, on page 208](#)

There is also a one-step process, but depending on the number of jobs being submitted, it may be too slow. You can also simply examine them upon submission and send them where you want. The method is listed here:

- Examining jobs upon submission and routing them using a hook; see [section 4.9.39.1.iv, “Examining Jobs Upon Submission”, on page 207](#).

You can use any of several mechanisms for moving jobs. Each is described in subsections below. The mechanisms for moving jobs are the following:

- Routing queues; see [section 4.9.39.2.i, “Routing Queues as Mechanism to Move Jobs”, on page 207](#)
- Hooks; see [section 4.9.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 207](#)
- Peer scheduling; see [section 4.9.39.2.iii, “Peer Scheduling as Mechanism to Move Jobs”, on page 207](#)
- The `qmove` command; see [section 4.9.39.2.iv, “The `qmove` Command as Mechanism to Move Jobs”, on page 208](#)

You can use filtering methods to control which jobs are allowed into destination queues. We describe filtering methods in subsections below. The filtering mechanisms are the following:

- Resource limits; jobs are filtered by resource request. See [section 4.9.39.3.i, “Resource Limits as Filtering Mechanism”, on page 208](#)
- Access control limits; jobs are filtered by owner. See [section 4.9.39.3.ii, “Access Controls as Filtering Mechanism”, on page 208](#)

You can use a combination of moving a job and "tagging" it, that is, including a special custom resource in the job's resource request, to route the job. If you set the resource using a hook, you can route the job either to a queue or to vnodes. If you make the job inherit the resource from a queue, you can route it only to vnodes. You can set resource limits for the special custom resource at the receiving queue, allowing in only jobs with the special resource. You can set the special custom resource at vnodes, so that the job must run there. Mechanisms for tagging jobs are listed here:

- Using a hook to assign a resource; see [section 4.9.39.4.i, “Using Hooks to Tag Jobs”, on page 208](#)
- Associating vnodes with queues; see [section 4.9.2, “Associating Vnodes with Queues”, on page 107](#)
- Changing the job's resource request using the `qalter` command; see [section 4.9.39.4.ii, “Using the `qalter` Command to Tag Jobs”, on page 209](#)

4.9.39.1 Mechanisms for Collecting Jobs

4.9.39.1.i Default Queue as Mechanism to Collect Jobs

To make it easy on your users, have their jobs land in your routing queue by default. You probably don't want frustrated users trying to submit jobs without specifying a queue, only to have the jobs be rejected if you have set access controls on, or only allowed routing to, the default queue. The server's `default_queue` attribute specifies the name of the default queue. To make things easy, make the default queue be the routing queue:

```
Qmgr: set server default_queue = <queue name>
```

4.9.39.1.ii Grabbing Jobs Upon Submission

You can allow users to submit jobs to any queue, and then scoop up the newly-submitted jobs and put them in the desired queue. To do this, you write a hook. See the *PBS Professional Hooks Guide*.

4.9.39.1.iii Disallowing Direct Submission as Mechanism to Collect Jobs

If you are using a routing queue, you can disallow job submission to all other queues. This forces users to submit jobs to the routing queue. You should probably make the routing queue be the default queue in this case, to avoid irritating users. Whether or not a queue allows direct job submission is controlled by its `from_route_only` attribute. To disallow job submission to a queue:

```
Qmgr: set queue <queue name> from_route_only = True
```

4.9.39.1.iv Examining Jobs Upon Submission

You can use a job submission hook to examine each job as it is submitted, and then route it to the desired queue. For example, you can route jobs directly according to resource request, project, owner, etc. See the *PBS Professional Hooks Guide*.

4.9.39.2 Mechanisms for Moving Jobs

4.9.39.2.i Routing Queues as Mechanism to Move Jobs

Routing queues are a mechanism supplied by PBS that automatically move jobs from a routing queue to another queue. You can direct which destination queues accept a job using these filters at each destination queue:

- Resource limits: you can set up execution queues designed for specific kinds of jobs, and then route each kind of job separately. For example, you can create two execution queues, and one routing queue, and route all jobs requesting large amounts of memory to one of the execution queues, and the rest of the jobs to the other queue. See [section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 28](#).
- Access control limits: you can set up destination queues that are designed for specific groups of users. Each queue accepts jobs only from a designated set of users or groups. For example, if you have three departments, Math, Physics, and Chemistry, the queue belonging to Math accepts only users from the Math department. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).

When routing a job between complexes, the job's owner must be able to submit a job to the destination complex.

For how to configure and use routing queues, see [section 2.3.6, “Routing Queues”, on page 27](#).

4.9.39.2.ii Hooks as Mechanism to Move Jobs

You can use a submission hook to move jobs into queues such as dedicated time queues, queues with special priority, or reservation queues. You write the hook so that it identifies the jobs that should go into a particular queue, and then moves them there. For example, your hook can move all jobs from ProjectA to a specific queue. This is a snippet, where you would replace `<destination queue>` with the queue name.

```
import pbs
e = pbs.event()
e.job.queue = pbs.server().queue("<destination queue>")
```

For complete information on hooks, see the *PBS Professional Hooks Guide*.

4.9.39.2.iii Peer Scheduling as Mechanism to Move Jobs

To send jobs from one partition or complex to another, you use peer scheduling. In peer scheduling, the partition or complex that supplies the jobs (the "furnishing" partition or complex) contains at least one special queue (the "furnishing queue"), whose jobs can be pulled over to another partition or complex, to be run at the other partition or complex. The partition or complex that pulls jobs contains a special queue (the "pulling queue"), where those pulled jobs land.

You can use any of the job routing methods, such as routing queues, tagging, or hooks, to control which jobs land in the furnishing queue.

You can use any of the job filtering methods, such as resource limits or access controls, to control which jobs land in the furnishing queue.

You can use job submission hooks on the jobs that land in the pulling queue.

See [section 4.9.31, “Peer Scheduling”, on page 164](#).

4.9.39.2.iv The `qmove` Command as Mechanism to Move Jobs

You can use the `qmove` command, either manually or via a `cron` job, to move jobs into the desired queues. See [“qmove” on page 175 of the PBS Professional Reference Guide](#).

4.9.39.3 Mechanisms for Filtering Jobs

4.9.39.3.i Resource Limits as Filtering Mechanism

You can filter whether each job is accepted at the server or a queue based on the job's resource request. For example, you can control which jobs are allowed to be submitted to the server, by limiting the amount of memory a job is allowed to request. You can do the same at execution queues. These limits apply regardless of the routing mechanism being used, and apply to jobs being submitted directly to the queue. See [section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251](#).

4.9.39.3.ii Access Controls as Filtering Mechanism

You can filter jobs whether each job is accepted at the server or a queue based on the job's owner, or the job owner's group. At each queue and at the server, you can create a different list of the users who can submit jobs and the users who cannot submit jobs. You can do the same for groups.

For example, you can set up a routing queue and several execution queues, where each execution queue has access controls allowing only certain users and groups. When PBS routes the jobs from the routing queue, it will route them into the execution queues that accept owners of the jobs. See [section 2.3.6.5, “Using Access Control to Route Jobs”, on page 32](#).

4.9.39.3.iii Hooks as Filtering Mechanism

You can filter which jobs are accepted at the server or queues according to any criterion, using a hook. For example, you can write a hook that disallows jobs that request certain combinations of resources. See the PBS Professional Hooks Guide.

4.9.39.4 Mechanisms for Tagging Jobs

4.9.39.4.i Using Hooks to Tag Jobs

You can use a hook to force certain jobs to run on particular hardware, by having the hook set the value of a host-level custom resource in a job's resource request. The hook sets this resource to match the value at the selected vnodes, so that the job must run on one or more of those vnodes. You can use the job's project to determine how the job is tagged. Note that the value at other vnodes should be different, otherwise the job could end up on vnodes you don't want.

- Define a host-level custom resource; see [section 5.14.4, “Configuring Host-level Custom Resources”, on page 265](#).
- Set this resource to a special value on the special vnodes only. See [section 5.7.2, “Setting Values for Static Resources”, on page 238](#).
- Create a hook that filters jobs by size, project, or other characteristic, and sets the value of the custom resource to the special value, in the job's resource request. See the PBS Professional Hooks Guide

If you must use a routing queue, and you need to route on host-level resources (resources in the job's select specification), you can use a hook to tag jobs so that they are routed correctly. The hook reads the job's host-level resource request, and sets the job's server-level resource request accordingly. This server-level resource is used for routing:

- Create a custom server-level resource that you use exclusively for routing; set it to appropriate values on the destination queues; see [section 5.14.3, “Creating Server-level Custom Resources”, on page 263](#)
- Create a submit hook to extract the host-level resource value and use it to populate the custom resource that you use exclusively for routing; see the PBS Professional Hooks Guide

4.9.39.4.ii Using the `qalter` Command to Tag Jobs

You can change a job's resource request using the `qalter` command. This way you can override normal behavior. See [“qalter” on page 130 of the PBS Professional Reference Guide](#).

4.9.40 Scheduler Cycle Speedup

4.9.40.1 Top Job Calculation Speedup

When you are using backfilling, you can choose whether and how much you want to speed up the scheduling cycle (within limits). You can get shorter scheduling cycle duration with coarser granularity in estimating start times for jobs. When you are using backfilling, a scheduler calculates estimated start times for jobs. You can choose not to make this trade-off (keeping fine granularity in start time estimation), or you can choose low, medium, or high speedup. See [section 4.9.3, “Using Backfilling”, on page 109](#).

4.9.40.1.i Configuring Top Job Calculation Speedup

You configure top job calculation speedup by using `qmgr` to set the `opt_backfill_fuzzy` scheduler attribute:

```
Qmgr: set sched opt_backfill_fuzzy [off | low | medium | high]
```

where each option has the following effect:

off

This scheduler uses its normal, finest granularity. No speedup.

low

This scheduler uses fairly fine granularity, not as fine as normal. Some speedup.

medium

This scheduler uses medium granularity. Medium speedup.

high

This scheduler uses the coarsest granularity. Greatest speedup.

The options *off*, *low*, *medium*, and *high* are not case-sensitive. You can use only one option at a time. Since this is an attribute and not a scheduler parameter, it is not a primetime option.

4.9.40.1.ii What Changing Calculation Speed Affects

Changing this attribute takes effect on the next scheduling cycle. If you change this attribute, top jobs are recalculated in the next scheduling cycle.

Once an ASAP reservation is made, it is fixed. If you change `opt_backfill_fuzzy` later, the reservation start time does not change, even if it becomes degraded. PBS finds new vnodes for degraded reservations, but does not change the start times.

4.9.40.1.iii Caveats and Restrictions for Top Job Calculation Speedup

This option is effective only when you are using backfilling.

4.9.41 Shared vs. Exclusive Use of Resources by Jobs

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode's `sharing` attribute and the placement requested by a job.

You can set each vnode's `sharing` attribute so that the vnode or host is always shared, is always exclusive, or so that it honors the job's placement request. If the vnode attribute is set to `force_shared` or `force_excl`, the value of a vnode's `sharing` attribute takes precedence over a job's placement request. If the vnode attribute is set to `default_`, the job request overrides the vnode attribute.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs. If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its `sharing` attribute set to either `default_exclhost` or `force_exclhost`, all vnodes on that host must have the same value for the `sharing` attribute.

For a complete description of the `sharing` attribute, and a table showing the interaction between the value of the `sharing` attribute and the job's placement request, see [“sharing” on page 325 of the PBS Professional Reference Guide](#).

4.9.41.1 Sharing on a Multi-vnode Machine

On a multi-vnode shared-memory machine, a scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

To keep a job in a single vnode, use `-lplace=group=vnode`; if you want to restrict it to larger sets of vnodes, identify those sets using a custom string or `string_array` resource and use it in `-lplace=group=<resource>`. If you already have resources used in `node_group_key` you can usually use these.

4.9.41.2 Setting the sharing Vnode Attribute

To set the `sharing` attribute for a vnode, use either:

- An `exechost_startup` hook; see [“Setting and Unsetting Vnode Resources and Attributes” on page 49 in the PBS Professional Hooks Guide](#)
- A Version 2 configuration file; see [section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 50](#)

4.9.41.3 Viewing Sharing Information

You can use the `qmgr` or `pbsnodes` commands to view sharing information. See [“qmgr” on page 152 of the PBS Professional Reference Guide](#) and [“pbsnodes” on page 36 of the PBS Professional Reference Guide](#).

4.9.41.4 Sharing Caveats

- The term "sharing" is also used to describe the case where MoM manages a resource that is shared among her vnodes, for example an application license shared by the vnodes of a multi-vnode machine.
- The term "sharing" is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are "sharing" a CPU. See [section 8.6.6, "Managing Load Levels on Vnodes", on page 415](#)
- If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.
- For vnodes with *sharing=default_shared*, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The *exec_vnode* attribute will show this allocation.

4.9.42 Using Shrink-to-fit Jobs

4.9.42.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the *min_walltime* and *max_walltime* resources. PBS chooses the actual *walltime*. Any job that requests *min_walltime* is a **shrink-to-fit** job.

4.9.42.1.i Requirements for a Shrink-to-fit Job

A job must have a value for *min_walltime* to be a shrink-to-fit job. Shrink-to-fit jobs are not required to request *max_walltime*, but it is an error to request *max_walltime* and not *min_walltime*.

Jobs that do not have values for *min_walltime* are not shrink-to-fit jobs, and their *walltime* can be specified by the user, inherited through defaults, or set in a hook.

4.9.42.1.ii Comparison Between Shrink-to-fit and Non-shrink-to-fit Jobs

Shrink-to-fit jobs are treated the same as non-shrink-to-fit jobs unless explicitly stated. For example, job priority is not affected by being shrink-to-fit. The only difference between a shrink-to-fit and a non-shrink-to-fit job is how the job's *walltime* is treated. PBS sets the *walltime* at the time the job is run; any *walltime* settings not computed by PBS are ignored.

4.9.42.2 Where to Use Shrink-to-fit Jobs

If you have jobs that can run for less than the expected time to completion and still make useful progress, you can use them as shrink-to-fit jobs in order to maximize utilization.

You can use shrink-to-fit jobs for the following:

- Jobs that are internally checkpointed. This includes jobs which are part of a larger effort, where a job does as much work as it can before it is killed, and the next job in that effort takes up where the previous job left off.
- Jobs using periodic PBS checkpointing
- Jobs whose real running time might be much less than the expected time
- When you have set up dedicated time for system maintenance, and you want to keep machines well-utilized right up until shutdown, submitters who want to risk having a job killed before it finishes can run speculative shrink-to-fit jobs. Similarly, speculative jobs can take advantage of the time just before a reservation starts
- Any job where the submitter does not mind running the job as a speculative attempt to finish some work

4.9.42.3 Running Time of a Shrink-to-fit Job

4.9.42.3.i Setting Running Time Range for Shrink-to-fit Jobs

It is only required that the job request `min_walltime` to be a shrink-to-fit job. If a job requests `min_walltime` but does not request `max_walltime`, you may want to use a hook or defaults to set a reasonable value for `max_walltime`. If you use defaults, you may want to route shrink-to-fit jobs to a special queue where they inherit a value for `max_walltime` if they haven't got one already. See [section 4.9.39, “Routing Jobs”, on page 205](#).

Requesting `max_walltime` without requesting `min_walltime` is an error.

A job can end up with a value for `min_walltime` and `max_walltime` when the user specifies them, when it inherits them from server or queue defaults, or when they are set in a hook.

Job submitters can set the job's running time range by requesting `min_walltime` and `max_walltime`, for example:

```
qsub -l min_walltime=<min walltime>, max_walltime=<max walltime> <job script>
```

You can set `min_walltime` or `max_walltime` using a hook, whether or not the job requests it. You can set up defaults so that the job inherits these resources if they are not explicitly requested or set in a hook.

4.9.42.3.ii Inheriting Values for `min_walltime` and `max_walltime`

The `min_walltime` and `max_walltime` resources inherit values differently. A job can inherit a value for `max_walltime` from `resources_max.walltime`; the same is not true for `min_walltime`. This is because once a job is shrink-to-fit, PBS can use a `walltime` limit for `max_walltime`.

If a job is submitted without a value for `min_walltime`, the value for `min_walltime` for the job becomes the first of the following that exists:

- Server's default `qsub` arguments
- Queue's `resources_default.min_walltime`
- Server's `resources_default.min_walltime`

If a shrink-to-fit job is submitted without a value for `max_walltime`, the value for `max_walltime` for the job becomes the first of the following that exists:

- Server's default `qsub` arguments
- Queue's `resources_default.max_walltime`
- Server's `resources_default.max_walltime`
- Queue's `resources_max.walltime`
- Server's `resources_max.walltime`

4.9.42.3.iii Setting `walltime` for Shrink-to-fit Jobs

For a shrink-to-fit job, PBS sets the `walltime` resource based on the values of `min_walltime` and `max_walltime`, regardless of whether `walltime` is specified for the job. You cannot use a hook to set the job's `walltime`, and any queue or server defaults for `walltime` are ignored, except for the case where the job is run via `qrun -H`; see [section 4.9.42.8.ii, “Using `qrun` With `-H` Option”, on page 214](#).

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's `min_walltime` and `max_walltime`. If the job can fit somewhere, PBS sets the job's `walltime` to a duration that fits the time slot, and runs the job. The chosen value for `walltime` is visible in the job's `Resource_List.walltime` attribute. Any existing `walltime` value, regardless of where it comes from (user, queue default, hook, previous execution), is reset to the new calculated running time.

If a shrink-to-fit job is run more than once, PBS recalculates the job's running time to fit an available time slot that is between `min_walltime` and `max_walltime`, and resets the job's `walltime`, each time the job is run.

4.9.42.4 How PBS Places Shrink-to-fit Jobs

A PBS scheduler treats shrink-to-fit jobs the same way as it treats non-shrink-to-fit jobs when it schedules them to run. A scheduler looks at each job in order of priority, and tries to run it on available resources. If a shrink-to-fit job can be shrunk to fit in an available slot, a scheduler runs it in its turn. A scheduler chooses a time slot that is at least as long as the job's `min_walltime` value. A shrink-to-fit job may be placed in a time slot that is shorter than its `max_walltime` value, even if a longer time slot is available.

For a multi-vnode job, PBS chooses a `walltime` that works for all of the chunks required by the job, and places job chunks according to the placement specification.

4.9.42.5 Shrink-to-fit Jobs and Time Boundaries

The time boundaries that constrain job running time are the following:

- Reservations
- Dedicated time
- Primetime
- Start time for a top job

Time boundaries are not affected by shrink-to-fit jobs.

A shrink-to-fit job can shrink to avoid time boundaries, as long as the available time slot before the time boundary is greater than `min_walltime`.

If any job is already running, whether or not it is shrink-to-fit, and you introduce a new period of dedicated time that would impinge on the job's running time, PBS does not kill or otherwise take any action to prevent the job from hitting the new boundary.

4.9.42.5.i Shrink-to-fit Jobs and Prime Time

If you have enabled prime time by setting `backfill_prime` to *True*, shrink-to-fit jobs will honor the boundary between primetime and non-primetime. If `prime_spill` is *True*, shrink-to-fit jobs are scheduled so that they cross the prime-non-prime boundary by up to `prime_spill` duration only. If `prime_exempt_anytime_queues` is set to *True*, a job submitted in an anytime queue is not affected by primetime boundaries.

4.9.42.6 Shrink-to-fit Jobs and Resource Limits

4.9.42.6.i Shrink-to-fit Jobs and Gating at Server or Queue

Shrink-to-fit jobs must honor any resource limits at the server or queues. If a `walltime` limit is specified:

- Both `min_walltime` and `max_walltime` must be greater than or equal to `resources_min.walltime`.
- Both `min_walltime` and `max_walltime` must be less than or equal to `resources_max.walltime`.

If resource limits are not met, a job submission or modification request will fail with the following error:

```
"Job exceeds queue and/or server resource limits"
```

4.9.42.6.ii Gating Restrictions

You cannot set `resources_min` or `resources_max` for `min_walltime` or `max_walltime`. If you try, you will see the following error message, for example for `min_walltime`:

```
"Resource limits can not be set for min_walltime"
```

4.9.42.7 Shrink-to-fit Jobs and Preemption

When preempting other jobs, shrink-to-fit jobs do not shrink. Their `walltime` is set to their `max_walltime`.

4.9.42.8 Using `qrun` on Shrink-to-fit Jobs

If you use `qrun` on a shrink-to-fit job, its behavior depends on whether you use the `-H` option to `qrun`.

4.9.42.8.i Using `qrun` Without `-H` Option

When a shrink-to-fit job is run via `qrun`, it can shrink into available space to run. However, if preemption is enabled and there is a preemptable job that must be preempted in order to run the shrink-to-fit job, the preemptable job is preempted and the shrink-to-fit job shrinks and runs.

When a shrink-to-fit job is run via `qrun`, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job's `max_walltime` but not its `min_walltime`, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.
- If preemption is enabled and there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.9.42.8.ii Using `qrun` With `-H` Option

When a shrink-to-fit job is run via `qrun -H`, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via `qrun -H`, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for `walltime`, that value is used, instead of one set by PBS when the job runs. If no `walltime` is specified, the job runs without a `walltime`.

4.9.42.9 Modifying Shrink-to-fit and Non-shrink-to-fit Jobs

4.9.42.9.i Modifying `min_walltime` and `max_walltime`

You can change `min_walltime` and/or `max_walltime` for a shrink-to-fit job using `modifyjob` or `queuejob` hooks, or by using the `qalter` command. Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.9.42.9.ii Making Non-shrink-to-fit Jobs into Shrink-to-fit Jobs

You can convert a normal non-shrink-to-fit job into a shrink-to-fit job using the following methods:

- Use a hook that does the following:
 - Sets `max_walltime` to the job's `walltime`
 - Sets `min_walltime` to a useful value
- Use `resources_default` at the server or a queue. For a queue, you might want to set that queue's `from_route_only` attribute to `True`.
- Route to a queue that has `resources_default.min_walltime` set.
- Use the `qalter` command to set values for `min_walltime` and `max_walltime`.

Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.9.42.9.iii Making Shrink-to-fit Jobs into Non-shrink-to-fit Jobs

To make a shrink-to-fit job into a normal, non-shrink-to-fit job, use either a hook or the `qalter` command to do the following:

- Set the job's `walltime` to the value for `max_walltime` (beware of allowing the job to run into existing reservations etc.)
- Unset `min_walltime`
- Unset `max_walltime`

4.9.42.9.iv Hooks for Running Time Limits

If you want to set a new running time limit for shrink-to-fit jobs, you can use a hook. However, this hook must set the value of `max_walltime`, rather than `walltime`, since hook settings for `walltime` for a shrink-to-fit job are ignored.

4.9.42.10 Viewing Running Time for a Shrink-to-fit Job

4.9.42.10.i Viewing min_walltime and max_walltime

You can use `qstat -f` to view the values of the `min_walltime` and `max_walltime`. For example:

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<job ID>
% qstat -f <job ID>
...
resource_list.min_walltime=01:00:15
resource_list.max_walltime=03:30:00
```

You can use `tracejob` to display `max_walltime` and `min_walltime` as part of the job's resource list. For example:

```
12/16/2011 14:28:55 A user=pbsadmin group=Users project=_pbs_project_default
...
Resource_List.max_walltime=10:00:00
Resource_List.min_walltime=00:00:10
```

4.9.42.10.ii Viewing walltime for a Shrink-to-fit Job

PBS sets a job's `walltime` only when the job runs. While the job is running, you can see its `walltime` via `qstat -f`. While the job is not running, you cannot see its real `walltime`; it may have a value set for `walltime`, but this value is ignored.

You can see the `walltime` value for a finished shrink-to-fit job if you are preserving job history. See [section 10.15, “Managing Job History”, on page 480](#).

You can see the `walltime` value for a finished shrink-to-fit job in the scheduler log.

4.9.42.11 Lifecycle of a Shrink-to-fit Job

4.9.42.11.i Execution of Shrink-to-fit Jobs

Shrink-to-fit jobs are started just like non-shrink-to-fit jobs.

4.9.42.11.ii Termination of Shrink-to-fit Jobs

When a shrink-to-fit job exceeds the `walltime` PBS has set for it, it is killed by PBS exactly as a non-shrink-to-fit job is killed when it exceeds its `walltime`.

4.9.42.12 The min_walltime and max_walltime Resources

max_walltime

Maximum **walltime** allowed for a shrink-to-fit job. Job's actual **walltime** is between **max_walltime** and **min_walltime**. PBS sets **walltime** for a shrink-to-fit job. If this resource is specified, **min_walltime** must also be specified. Must be greater than or equal to **min_walltime**. Cannot be used for **resources_min** or **resources_max**. Cannot be set on job arrays or reservations. If not specified, PBS uses an eternal time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

min_walltime

Minimum **walltime** allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's **walltime**. Job's actual **walltime** is between **max_walltime** and **min_walltime**. Must be less than or equal to **max_walltime**. Cannot be used for **resources_min** or **resources_max**. Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

4.9.42.13 Accounting and Logging for Shrink-to-fit Jobs

4.9.42.13.i Accounting Log Entries for min_walltime and max_walltime

The accounting log will contain values for **min_walltime** and **max_walltime**, as part of the job's **Resource_List** attribute. This attribute is recorded in the S, E, and R records in the accounting log. For example, if the following job is submitted:

```
qsub -l min_walltime="00:01:00",max_walltime="05:00:00" -l select=2:ncpus=1 job.sh
```

This is the resulting accounting record:

```
...S..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18 resources_assigned.ncpus=2

...R..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18

...E..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18.....
```

4.9.42.13.ii Logging

- When a scheduler finds a primetime/dedicated time conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level PBSEVENT_DEBUG2:

```
"Considering shrinking job to duration=<duration>, due to prime/dedicated time conflict"
```

Sample message from the scheduler log:

```
"03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3;Considering shrinking job to duration=1:06:05,
due to a prime/dedicated time conflict"
```

This message doesn't indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job's maximum running time conflicted with primetime and the job can still be run by shrinking its running time.

- When a scheduler finds a reservation/top job conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level PBSEVENT_DEBUG2:
- ```
"Considering shrinking job to duration=<duration>", due to reservation/top job conflict"
```

Sample log message from the scheduler log:

```
"03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3; Considering shrinking job to
duration=1:06:05, due to reservation/top job conflict"
```

This message doesn't indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job's maximum running time conflicted with a reservation or top job and the job can still be run by shrinking its running time.

- When a scheduler runs the shrink-to-fit job, the following message is logged in the scheduler logs with log level PBSEVENT\_DEBUG2:

```
"Job will run for duration=<duration>"
```

Sample scheduler log message:

```
"03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3;Job will run for duration=1:06:05"
```

#### 4.9.42.14 Caveats and Restrictions for Shrink-to-fit Jobs

- It is erroneous to specify `max_walltime` for a job without specifying `min_walltime`. If a `queuejob` or `modifyjob` hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:  
'Can not have "max\_walltime" without "min\_walltime"'
- It is erroneous to specify a `min_walltime` that is greater than `max_walltime`. If a `queuejob` or `modifyjob` hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:  
'"min\_walltime" can not be greater than "max\_walltime"'
- Job arrays cannot be shrink-to-fit. You cannot have a shrink-to-fit job array. It is erroneous to specify a `min_walltime` or `max_walltime` for a job array. If a `queuejob` or `modifyjob` hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:  
'"min\_walltime" and "max\_walltime" are not valid resources for a job array'
- Reservations cannot be shrink-to-fit. You cannot have a shrink-to-fit reservation. It is erroneous to set `min_walltime` or `max_walltime` for a reservation. If attempted via `pbs_rsub`, the following error is printed:  
'"min\_walltime" and "max\_walltime" are not valid resources for reservation.'
- It is erroneous to set `resources_max` or `resources_min` for `min_walltime` and `max_walltime`. If attempted, the following error message is displayed, whichever is appropriate:  
"Resource limits can not be set for min\_walltime"  
"Resource limits can not be set for max\_walltime"

### 4.9.43 SMP Cluster Distribution

This tool is **deprecated**. PBS provides a method for distributing single-chunk jobs to a cluster of single-vnode machines according to a simple set of rules. The method is called *SMP cluster distribution*. It takes into account the resources specified on the `resources:` line in `<sched_priv directory>/sched_config`. The SMP cluster distribution method allows you to choose one of three job distribution systems:

**Table 4-18: SMP Cluster Distribution Options**

| Option      | Meaning                                                                             |
|-------------|-------------------------------------------------------------------------------------|
| pack        | Pack all jobs onto one vnode, until that vnode is full, then move to the next vnode |
| round_robin | Place one job on each vnode in turn, before cycling back to the first vnode         |

### 4.9.43.1 How to Use SMP Cluster Distribution

To use SMP cluster distribution, do the following:

- Set the `smp_cluster_dist` scheduler parameter to the desired value. For example, to enable SMP cluster distribution using the round robin algorithm during primetime, and the pack algorithm during non-primetime, set the following in the scheduler's configuration file:  

```
smp_cluster_dist: round_robin prime
smp_cluster_dist: pack non_prime
```
- Set `resources_available.<resource name>` to the desired limit on each vnode. You do not need to set any of the resources that are automatically set by PBS. For a list of these, see [section 5.7.1.1, “How Vnode Available Resource Values are Set”](#), on page 236.
- Specify the resources to use during scheduling, in `<sched_priv directory>/sched_config`:  

```
resources: "ncpus, mem, arch, host, ..."
```

The `smp_cluster_dist` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

### 4.9.43.2 How To Disable SMP Cluster Distribution

To ensure that SMP cluster distribution does not interfere with your scheduling policy, leave the `smp_cluster_dist` parameter set to its default value:

```
smp_cluster_dist pack all
```

### 4.9.43.3 SMP Cluster Distribution Caveats and Advice

- This feature was intended for early implementations of complexes, and probably is not useful for you.
- If you use this feature, you are committed to using it for the entire partition or complex; you cannot designate some machines where it will be used and others where it will not be used.
- If `smp_cluster_dist` with either *round\_robin* is used with `node_sort_key` set to *unused* or *assigned*, `smp_cluster_dist` is set to *pack*.
- The `avoid_provision` provisioning policy is incompatible with the `smp_cluster_dist` scheduler configuration parameter. If a job requests an AOE, the `avoid_provision` policy overrides the behavior of `smp_cluster_dist`.
- This feature is applied only to single-chunk jobs that specify an arrangement of pack. Multi-chunk jobs are ignored.
- This feature is useful only for single-vnode machines. On a multi-vnoded machine, this feature distributes jobs across vnodes, but those jobs can end up all stuck on a single host.
- The choice of `smp_cluster_dist` with *round\_robin* can be replaced by sorting vnodes according to unused CPUs, which does a better job:  

```
node_sort_key: "ncpus HIGH unused"
```

## 4.9.44 Using Soft Walltime

A scheduler requires walltime to do backfilling. Job submitters want to avoid having their jobs killed if they run over their walltimes, so they may overestimate job walltimes. You can give a scheduler tighter time slots by giving jobs soft walltimes. Jobs are not killed if they go over their soft walltimes. If a job has both a walltime and a soft walltime, a scheduler uses the soft walltime.

When a job exceeds its soft walltime, PBS estimates a new soft walltime, and records the estimate in the job's `estimated.soft_walltime` attribute. The `estimated.soft_walltime` job attribute is readable by all, but writable only by PBS.

### 4.9.44.1 Assigning Soft Waltime to Jobs

You can set a soft walltime for a job by having it request the `soft_walltime` resource. You can set it in a server hook, or by using `qalter` or `resources_default`.

The `soft_walltime` resource can be requested for a job only by PBS Managers. The `soft_walltime` resource cannot be set at job submission time, except by a `queuejob` hook, because job submission uses user permissions. Soft walltime cannot be set in MoM hooks.

You can create a custom resource and allow users to request it, and then set the value of `soft_walltime` to that resource. See an example in [section 4.9.44.4, “Allowing Job Submitters to Set Soft Waltime”, on page 220](#).

### 4.9.44.2 How Soft and Hard Walltimes Are Used

When a job is queued:

- If the job is a top job, its `soft_walltime` is used in determining where the job fits into the calendar
- If the job is a filler job, its `soft_walltime` is used in determining whether the job conflicts with top jobs
- If the job is a filler job, its `hard_walltime` is used in determining whether the job conflicts with confirmed reservations
- If dedicated time is used, `soft_walltime` is used in determining whether the job will finish before dedicated time starts
- If `backfill_prime` is set, `soft_walltime` is used in determining whether the job will finish before the next prime boundary + `prime_spill`

When a job is running:

- If `resources_used.walltime`  $\leq$  `soft_walltime`, the job continues to run
- If `resources_used.walltime`  $>$  `soft_walltime`, the job has exceeded its `soft_walltime`. The job is not killed; the job's `soft_walltime` is extended:
  - Every time the job exceeds its `soft_walltime`, it is extended by 100% of its original `soft_walltime`
  - If both a `soft_walltime` and a `hard_walltime` are set, the `soft_walltime` is never extended past the job's `hard_walltime`
  - If a job exceeds its `soft_walltime` and crosses over into `dedicated_time`, PBS does not kill the job
  - The value of `Resource_List.soft_walltime` does not change. A scheduler sets the `estimated.soft_walltime` job attribute to the new `soft_walltime` estimate
- If a job is a preemption candidate, and `preempt_order` is based on the percentage the job has completed (e.g., `preempt_order SCR 20 S`), the initial `soft_walltime` request is used to determine the percentage of completion
  - If the job runs past its initial `soft_walltime` request, `preempt_order` behaves as if the job is 100% complete. It remains at 100% complete for the remainder of the job regardless of how many times the `soft_walltime` is extended. For example, if a job has `soft_walltime=1:00:00`, at 59m, the job is at 99% complete. At 1:00:00, the `soft_walltime` is extended to 2:00:00. At 1:30:00 the job remains at 100% complete since it has reached its original `soft_walltime` request

When confirming reservations:

- Only a job's `hard_walltime` is used in determining when jobs end
- A job's `soft_walltime` is not used when confirming reservations

### 4.9.44.3 Examples of Using Soft Waltime

Example 4-38: Job J has a `soft_walltime=1:00:00` but no `hard_walltime`

J exceeds its `soft_walltime`. J is extended by its original `soft_walltime` to 2:00:00.

If J exceeds its `soft_walltime` again, J is extended again by its original `soft_walltime` to 3:00:00

Example 4-39: Job K has a `soft_walltime`=1:00:00 and a `hard_walltime`=1:30:00

K exceeds its `soft_walltime`. Because 2:00:00 is past its `hard_walltime`, K is extended to its limit of 1:30:00 instead.

#### 4.9.44.4 Allowing Job Submitters to Set Soft Walltime

Example of hook to allow users to directly set `soft_walltime`:

```
import pbs
e = pbs.event()
j = e.job

j.Resource_List["soft_walltime"] = pbs.duration(j.Resource_List["set_soft_walltime"])
```

Job submitters request the new resource:

```
% qsub -l set_soft_walltime=1:00:00 -l select=1:ncpus=1
```

#### 4.9.44.5 Caveats and Restrictions for Soft Walltime

- The `soft_walltime` resource is not sent to the MoM when the job is started.
- A shrink-to-fit job requesting `soft_walltime` is rejected, because a job's `min_walltime` is the minimum amount of time a job needs to get any real work done. A job's `hard_walltime` can be set to its `min_walltime`. A job's `soft_walltime` has to be shorter than its `hard_walltime`. This means that the `soft_walltime` would have to be shorter than the job's minimum amount of time to get any real work done. The two features do not make sense together.

### 4.9.45 Sorting Jobs on a Key

PBS allows you to sort jobs on a key that you specify. This can be used when setting both execution and preemption priority. Sorting jobs comes into play after jobs have been divided into classes, because each class may contain more than one job. You can sort on one or more of several different keys, and for each key, you can sort either from low to high or from high to low.

You configure sorting jobs on a key by setting values for the `job_sort_key` scheduler parameter. When preemption is enabled, jobs are automatically sorted by preemption priority. [Table 4-8, “Job Execution Classes,” on page 137](#) shows where this step takes place.

You can create an invisible, unrequestable custom resource, and use a hook to set the value of this resource for each job. The hook modifies the job's resource request to include the new resource, and sets the value to whatever the hook computes. Then you can sort jobs according to the value of this resource.

The `job_sort_key` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

#### 4.9.45.1 job\_sort\_key Syntax

```
job_sort_key: "<sort key> HIGH | LOW <primetime option>"
```

You can use the following keys for sorting jobs:

**Table 4-19: Keys for Sorting Jobs**

| Sort Key       | Allowed Order | Description                                                                                                                                                                                                                                                    |
|----------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <PBS resource> | HIGH   LOW    | Sorts jobs according to how much of the specified resource they request.                                                                                                                                                                                       |
| fairshare_perc | HIGH   LOW    | Sorts according to fairshare percentage allotted to entity that owns job. This percentage is defined in the <code>resource_group</code> file.<br>If user A has more priority than user B, all of user A's jobs are always run first. Past history is not used. |
| job_priority   | HIGH   LOW    | Sorts jobs by the value of each job's Priority attribute.                                                                                                                                                                                                      |
| sort_priority  | HIGH   LOW    | <b>Deprecated.</b> Replaced by <code>job_priority</code> option.                                                                                                                                                                                               |

You can sort on up to 20 keys.

The argument to the `job_sort_key` parameter is a quoted string. The default for `job_sort_key` is that it is not in force.

See [“job\\_sort\\_key” on page 253 of the PBS Professional Reference Guide](#).

## 4.9.45.2 Configuring Sorting Jobs on a Key

You can specify more than one sort key, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for `job_sort_key`, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the `job_sort_key` scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- Specify the primetime behavior

A scheduler's configuration file is read on startup and HUP.

## 4.9.45.3 Examples of Sorting Jobs on Key

Example 4-40: Sort jobs so that those with long walltime come first:

```
job_sort_key: "walltime HIGH"
```

Example 4-41: For example, if you want big jobs to run first, where "big" means more CPUs, and if the CPUs are the same, more memory, sort on the number of CPUs requested, then the amount of memory requested:

```
job_sort_key: "ncpus HIGH" all
job_sort_key: "mem HIGH" all
```

Example 4-42: Sort jobs so that those with lower memory come first:

```
job_sort_key: "mem LOW" prime
```

Example 4-43: Sort jobs according to the value of an invisible custom resource called *JobOrder*:

```
job_sort_key: "JobOrder LOW" all
```



### 4.9.45.4 Caveats and Advice for Sorting Jobs on Key

- Do not use `fairshare_perc` as the sort key when using `fairshare`, meaning the `fair_share` scheduler parameter is enabled. If you do this, a scheduler will attempt to sort a set of jobs where each job has the same sort key value. This will not sort the jobs.
- Use the `fairshare_perc` option only when ordering jobs by entity shares. See [section 4.9.14, “Sorting Jobs by Entity Shares \(Was Strict Priority\)”, on page 133](#).
- To run big jobs first, use `ncpus` as the primary sort key for `job_sort_key`:  
`job_sort_key: "ncpus HIGH"`
- The `job_sort_key` parameter is overridden by the job sorting formula and by `fairshare`. It is invalid to set both `job_sort_formula` and `job_sort_key` at the same time. If they are both set, `job_sort_key` is ignored and the following error message is logged:  
"Job sorting formula and job\_sort\_key are incompatible. The job sorting formula will be used."
- A scheduler's configuration file contains an example line for `job_sort_key`. This line is commented out, but shows an example of `job_sort_key` with `"cput"` as the sorting key.
- The `preempt_priority` argument to the `job_sort_key` parameter is **deprecated**. Jobs are now automatically sorted by preemption priority when preemption is enabled.

### 4.9.46 Sorting Jobs by Requested Priority

You can sort jobs according to the priority that was requested for the job. This value is found in the job's `Priority` attribute. You can use this value in the following ways:

- The term `job_priority` represents the value of the job's `Priority` attribute in the job sorting formula. See [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).
- The `job_sort_key` scheduler parameter can take the term `job_priority` as an argument. The term `job_priority` represents the value of the job's `Priority` attribute. See [section 4.9.45, “Sorting Jobs on a Key”, on page 220](#).

You can use a hook to set or change the value of a job's `Priority` attribute. See the PBS Professional Hooks Guide.

### 4.9.47 Sorting Queues into Priority Order

PBS always sorts all the execution queues in your partition or complex according to their priority, and uses that ordering when examining queues individually. Queues are ordered with the highest-priority queue first.

If you want queues to be considered in a specific order, you must assign a different priority to each queue. Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. To set a queue's priority, use the `qmgr` command to assign a value to the `priority` queue attribute.

```
Qmgr: set queue <queue name> priority = <value>
```

Sorting queues into priority order is useful for the following:

- Examining queues one at a time. See [section 4.9.4, “Examining Jobs Queue by Queue”, on page 113](#).
- Selecting jobs from queues in a round-robin fashion. See [section 4.9.38, “Round Robin Queue Selection”, on page 204](#).

#### 4.9.47.1 Caveats and Advice when Sorting Queues

- If you do not set queue priorities, queue ordering is undefined.
- The `sort_queues` parameter is **obsolete** (version 20).

## 4.9.48 Using Strict Ordering

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. Using strict ordering means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

Strict ordering does not change how execution priority is calculated.

### 4.9.48.1 Configuring Strict Ordering

To configure strict ordering, set the `strict_ordering` scheduler parameter to *True*.

The `strict_ordering` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See [“strict\\_ordering” on page 258 of the PBS Professional Reference Guide](#).

### 4.9.48.2 How Strict Ordering Works

When `strict_ordering` is *True*, a scheduler runs jobs in exactly the order of their priority.

Strict ordering does not affect how job priority is calculated, but it does change which execution priority classes a scheduler uses; see [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

### 4.9.48.3 Combining Strict Ordering and Backfilling

Strict ordering alone may cause some resources to stand idle while the top job waits for resources to become available. If you want to prevent this, you can use backfilling with strict ordering. Using backfilling, if the top job cannot run, filler jobs can be squeezed in around the job that cannot run. See [section 4.9.3, “Using Backfilling”, on page 109](#).

### 4.9.48.4 Strict Ordering and Calendaring

If you mark a job's `topjob_ineligible` attribute *True*, PBS does not put that job in the calendar if it cannot run right now. See [section 4.9.17, “Calendaring Jobs”, on page 138](#).

### 4.9.48.5 Strict Ordering Caveats

- It is inadvisable to use strict ordering and backfilling with fairshare. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group as the desired large job will be chosen as the filler job. The usage from these filler jobs will lower the priority of the top job.

For example, if a user has a large job that is the top job, and that job cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from being likely to ever run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

- Using dynamic resources with strict ordering and backfilling may result in unpredictable scheduling. See [section 4.9.3.10, “Backfilling Recommendations and Caveats”, on page 112](#).
- Using preemption with strict ordering and backfilling may change which job is the top job.
- With both round robin and strict ordering, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

## 4.9.49 Sorting Vnodes on a Key

PBS can sort vnodes according to a key that you specify. This can be used when deciding which vnodes to use for jobs. Sorting vnodes comes into play after a placement set has been selected, or when a job will run on vnodes associated with a queue, or when placement sets are not used, because in those cases there may be more vnodes available than are needed. You can sort vnodes on one or more different keys, and for each key, you can sort from high to low, or the reverse.

You can sort on the `last_used_time` and `Priority` vnode attributes, and vnode resources.

The default way to sort vnodes is according to the value of the vnode `priority` attribute, from higher to lower.

When you sort vnodes according to the assigned or unused amount of a resource, the vnode list is re-sorted after every job is run. This is because each job may change the usage for that resource.

You configure sorting vnodes on a key by setting values for the `node_sort_key` scheduler parameter.

The `node_sort_key` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

When vnodes are not sorted on a key, their order is undefined.

### 4.9.49.1 `node_sort_key` Syntax

`node_sort_key: "sort_priority HIGH | LOW" <prime option>`

`node_sort_key: "<resource name> HIGH | LOW" <prime option>`

`node_sort_key: "<resource name> HIGH | LOW total | assigned | unused" <prime option>`

where

*total*

Use the `resources_available` value

*assigned*

Use the `resources_assigned` value

*unused*

Use the value given by `resources_available - resources_assigned`

Specifying a resource such as `mem` or `ncpus` sorts vnodes by the resource specified. Note that a scheduler rounds all resources of type `size`, including `mem`, up to the nearest kb.

Specifying the `sort_priority` keyword sorts vnodes on the vnode `priority` attribute.

The default third argument for a resource is *total*. If the third argument, *total | assigned | unused*, is not specified with a resource, *total* is used. This provides backwards compatibility with previous releases.

The values used for sorting must be numerical.

### 4.9.49.2 Configuring Sorting Vnodes on a Key

You can specify up to 20 sort keys, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for `node_sort_key`, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the `node_sort_key` scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- For sorting on a resource, optionally specify total, assigned, or unused
- Specify the primetime behavior

A scheduler's configuration file is read on startup and HUP.

The argument to the `node_sort_key` parameter is a quoted string. The default for `node_sort_key` is the following:

```
node_sort_key: "sort_priority HIGH" all
```

See [“node\\_sort\\_key” on page 254 of the PBS Professional Reference Guide](#).

### 4.9.49.3 Sorting Vnodes According to Load Average

To place jobs on the vnodes with the lowest load:

- Create a custom host-level resource to reflect current load, named for example "aveload":  
`qmgr -c "create resource aveload type=float,flag=h"`
- Write an `exechost_periodic` hook to set the resource to the value of the load average; see [“Log loads on vnodes” on page 317 in the PBS Professional Installation & Upgrade Guide](#) for an example of an `exechost_periodic` hook that reads the load on the host.
- Use the `aveload` resource as your `node_sort_key`:  
`node_sort_key: "aveload LOW" all`

### 4.9.49.4 Examples of Sorting Vnodes

Example 4-44: This sorts vnodes by the highest number of unused CPUs:

```
node_sort_key: "ncpus HIGH unused" all
```

Example 4-45: This sorts vnodes by the highest amount of memory assigned to vnodes, but only during primetime:

```
node_sort_key: "mem HIGH assigned" prime
```

Example 4-46: This sorts vnodes according to speed. You want to run jobs on the fastest host available. You have 3 machines, where HostA is fast, HostB is medium speed, and HostC is slow.

Set node priorities so that faster machines have higher priority:

```
Qmgr: set node HostA priority = 200
```

```
Qmgr: set node HostB priority = 150
```

```
Qmgr: set node HostC priority = 100
```

Specify that vnodes are sorted according to priority, with highest priority first:

```
node_sort_key: "sort_priority HIGH" ALL
```

Example 4-47: The old "nodepack" behavior can be achieved by this:

```
node_sort_key: "ncpus low unused"
```

Example 4-48: In this example of the interactions between placement sets and `node_sort_key`, we have 8 vnodes numbered 1-8. The vnode priorities are the same as their numbers. However, in this example, when unsorted, the vnodes are selected in the order 4, 1, 3, 2, 8, 7, 5, 6. This is to illustrate the change in behavior due to `node_sort_key`.

We use:

```
node_sort_key: "sort_priority LOW"
```

Using `node_sort_key`, the vnodes are sorted in order, 1 to 8. We have three placement sets:

A: 1, 2, 3, 4 when sorted by `node_sort_key`; 4, 1, 3, 2 when no `node_sort_key` is used

B: 5, 6, 7, 8 when sorted by `node_sort_key`; 8, 7, 5, 6 when no `node_sort_key` is used

C: 1-8 when sorted, 4, 1, 3, 2, 8, 7, 5, 6 when not sorted.

A 6-vnode job will not fit in either A or B, but will fit in C. Without the use of `node_sort_key`, it would get vnodes 4, 1, 3, 2, 8, 7. With `node_sort_key`, it would get vnodes 1 - 6, still in placement set C.

#### 4.9.49.5 Caveats for Sorting Vnodes

- Sorting on a resource and using "unused" or "assigned" cannot be used with `smp_cluster_dist` when it is set to anything but "pack". If both are used, `smp_cluster_dist` will be set to "pack".
- A scheduler rounds all resources of type `size`, including `mem`, up to the nearest kb. This can affect how vnodes are sorted when you are sorting on `mem`.

# Using PBS Resources

This chapter covers PBS resources, including providing resources for user jobs, setting up resources such as application licenses and scratch space, how to make objects inherit resources, and how to use, define, and view resources.

## 5.1 Chapter Contents

|        |                                                                                                         |     |
|--------|---------------------------------------------------------------------------------------------------------|-----|
| 5.1    | Chapter Contents                                                                                        | 227 |
| 5.2    | Introduction to PBS Resources                                                                           | 228 |
| 5.3    | Glossary                                                                                                | 228 |
| 5.4    | Categories of Resources                                                                                 | 230 |
| 5.4.1  | Built-in vs. Custom Resources                                                                           | 231 |
| 5.4.2  | Server vs. Queue vs. Vnode Resources                                                                    | 231 |
| 5.4.3  | Consumable vs. Non-consumable Resources                                                                 | 231 |
| 5.4.4  | Static vs. Dynamic Resources                                                                            | 232 |
| 5.4.5  | Requested vs. Default Resources                                                                         | 232 |
| 5.4.6  | Shared vs. Non-shared Vnode Resources                                                                   | 233 |
| 5.4.7  | Platform-specific vs. Generally Available Resources                                                     | 233 |
| 5.4.8  | Job-wide vs. Chunk Resources                                                                            | 233 |
| 5.5    | Resource Types                                                                                          | 234 |
| 5.6    | Resource Formats                                                                                        | 234 |
| 5.6.1  | Resource Names                                                                                          | 235 |
| 5.7    | Setting Values for Resources                                                                            | 236 |
| 5.7.1  | How Resource Values are Set                                                                             | 236 |
| 5.7.2  | Setting Values for Static Resources                                                                     | 238 |
| 5.7.3  | Setting Values for String Arrays                                                                        | 238 |
| 5.7.4  | When Resource Changes Take Effect                                                                       | 238 |
| 5.7.5  | Caveats for Setting Resource Values                                                                     | 239 |
| 5.8    | Overview of Ways Resources Are Used                                                                     | 239 |
| 5.8.1  | How the Scheduler Uses Resources                                                                        | 240 |
| 5.8.2  | Advice on Using string and string_array Resources                                                       | 240 |
| 5.9    | Resources Allocated to Jobs and Reservations                                                            | 240 |
| 5.9.1  | Allocating Chunks                                                                                       | 241 |
| 5.9.2  | Resources Requested by Job                                                                              | 241 |
| 5.9.3  | Specifying Job Default Resources                                                                        | 241 |
| 5.9.4  | Allocating Default Resources to Jobs                                                                    | 244 |
| 5.9.5  | Dynamic Resource Allocation Caveats                                                                     | 247 |
| 5.9.6  | Period When Resource is Used by Job                                                                     | 247 |
| 5.10   | Using Resources to Track and Control Allocation                                                         | 249 |
| 5.11   | Using Resources for Topology and Job Placement                                                          | 250 |
| 5.11.1 | Restrictions on Using Resources for Job Placement                                                       | 250 |
| 5.12   | Using Resources to Prioritize Jobs                                                                      | 251 |
| 5.13   | Using Resources to Restrict Server or Queue Access                                                      | 251 |
| 5.13.1 | Admittance Limits for <code>walltime</code> , <code>min_walltime</code> , and <code>max_walltime</code> | 251 |
| 5.13.2 | Restrictions on Resources Used for Admittance                                                           | 252 |
| 5.14   | Custom Resources                                                                                        | 252 |
| 5.14.1 | How to Use Custom Resources                                                                             | 252 |

---

|         |                                                                                      |     |
|---------|--------------------------------------------------------------------------------------|-----|
| 5.14.2  | Defining New Custom Resources. . . . .                                               | 254 |
| 5.14.3  | Creating Server-level Custom Resources. . . . .                                      | 263 |
| 5.14.4  | Configuring Host-level Custom Resources . . . . .                                    | 265 |
| 5.14.5  | Using Scratch Space . . . . .                                                        | 269 |
| 5.14.6  | Supplying Application Licenses. . . . .                                              | 270 |
| 5.14.7  | Using GPUs . . . . .                                                                 | 279 |
| 5.14.8  | Using FPGAs . . . . .                                                                | 282 |
| 5.14.9  | Defining Host-level Resource for Applications . . . . .                              | 282 |
| 5.14.10 | Custom Resource Caveats . . . . .                                                    | 282 |
| 5.15    | Managing Resource Usage . . . . .                                                    | 283 |
| 5.15.1  | Managing Resource Usage By Users, Groups, and Projects, at Server & Queues . . . . . | 283 |
| 5.15.2  | Placing Resource Limits on Jobs . . . . .                                            | 300 |
| 5.15.3  | Limiting the Number of Jobs in Queues. . . . .                                       | 305 |
| 5.16    | Where Resource Information Is Kept . . . . .                                         | 305 |
| 5.16.1  | Files . . . . .                                                                      | 305 |
| 5.16.2  | MoM Configuration Parameters. . . . .                                                | 306 |
| 5.16.3  | Attributes . . . . .                                                                 | 306 |
| 5.17    | Viewing Resource Information . . . . .                                               | 307 |
| 5.17.1  | Resource Information in Accounting Logs . . . . .                                    | 308 |
| 5.17.2  | Resource Information in Daemon Logs . . . . .                                        | 308 |
| 5.17.3  | Finding Current Value . . . . .                                                      | 309 |
| 5.17.4  | Restrictions on Viewing Resources . . . . .                                          | 309 |
| 5.18    | Resource Recommendations and Caveats. . . . .                                        | 309 |

## 5.2 Introduction to PBS Resources

PBS resources represent things such as CPUs, memory, application licenses, switches, scratch space, and time. They can also represent whether or not something is true, for example, whether a machine is dedicated to a particular project. PBS provides a set of built-in resources, and allows you to define additional custom resources. For some systems, PBS creates specific custom resources. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs. The administrator can specify which resources are available at the server, each queue, and each vnode.

## 5.3 Glossary

### Reservation

A reservation for a specific set of resources for a specified start time and duration in the future. See [section 4.9.37, “Reservations”, on page 196](#).

### Borrowing vnode

A shared vnode resource is available for use by jobs at more than one vnode, but is managed at just one vnode. A *borrowing vnode* is a vnode where a shared vnode resource is available, but not managed.

### Built-in resource

A resource that is defined in PBS Professional as shipped. Examples of built-in resources are `ncpus`, which tracks the number of CPUs, and `mem`, which tracks memory. See [section 5.4.1, “Built-in vs. Custom Resources”, on page 231](#).



---

**Chunk**

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

**Consumable resource**

A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See [section 5.4.3, “Consumable vs. Non-consumable Resources”, on page 231](#).

**CPU**

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.
2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called "*ncpus*". The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

**Custom resource**

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See [section 5.4.1, “Built-in vs. Custom Resources”, on page 231](#) and [section 5.14, “Custom Resources”, on page 252](#).

**Dynamic resource**

A custom resource that tracks an external quantity. PBS updates the tracking value via a script or an `exechost_periodic` hook. See [section 5.14.1.2, “Dynamic Custom Resources”, on page 253](#).

**Floating license**

A unit of license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).

**Generic group limit**

A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

**Generic user limit**

A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

**Group limit**

Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

**Indirect resource**

A shared vnode resource at `vnode(s)` where the resource is not defined, but which share the resource.

**Individual group limit**

Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

**Individual user limit**

Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

**Limit**

A maximum that can be applied in various situations:

- The maximum number of jobs that can be queued
- The maximum number of jobs that can be running
- The maximum number of jobs that can be queued and running
- The maximum amount of a resource that can be allocated to queued jobs
- The maximum amount of a resource that can be consumed at any time by running jobs
- The maximum amount of a resource that can be allocated to queued and running jobs

**Managing vnode**

The vnode where a shared vnode resource is defined, and which manages the resource.

**Memory-only vnode**

Represents a node board that has only memory resources (no CPUs).

**Non-consumable resource**

A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and `walltime`. See [section 5.4.3, “Consumable vs. Non-consumable Resources”](#), on page 231.

**Overall limit**

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

**Resource**

A *resource* can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, licenses, or time, or it can be an arbitrary item defined for another purpose. PBS provides built-in resources, and allows custom-defined resources.

**Shared resource**

A vnode resource defined and managed at one vnode, but available for use at other vnodes.

**User limit**

Refers to configurable limits on resources and jobs. A user's limit, whether generic or individual.

## 5.4 Categories of Resources

A PBS resource has several defining characteristics describing where and how it is used, how it was defined, etc. Each characteristic puts it in one of a set of categories. A resource inhabits several categories at once; for example, a resource can be a custom static consumable server resource. We describe the sets of categories below.

---

## 5.4.1 Built-in vs. Custom Resources

Built-in resources are the resources that are already defined for you in PBS. PBS supplies built-in resources including number of CPUs, CPU time, and memory. For a list of built-in resources, see [“Resources Built Into PBS” on page 265 of the PBS Professional Reference Guide](#). Custom resources are those that you define, or that PBS creates for some systems. For example, if you wanted a resource to represent scratch space, you could define a resource called *Scratch*, and specify a script which queries for the amount of available scratch space. See [section 5.14, “Custom Resources”, on page 252](#).

## 5.4.2 Server vs. Queue vs. Vnode Resources

PBS resources can be available at the server, queues, both the server and queues, or at vnodes. Any of these resources can be static or dynamic, built-in or custom, and consumable or non-consumable.

### 5.4.2.1 Server Resources

A server resource, also called a server-level resource, is a resource that is available at the server. A server resource is available to be consumed or matched at the server if you set the server's `resources_available.<resource name>` attribute to the available or matching value. For example, you can define a custom resource called *FloatingLicenses* and set the server's `resources_available.FloatingLicenses` attribute to the number of available floating licenses.

A server resource is a job-wide resource. This means that a job can request this resource for the entire job, but not for individual chunks.

An example of a job-wide resource is shared scratch space, or any custom resource that is defined at the server and queue level.

### 5.4.2.2 Queue Resources

A queue resource, also called a queue-level resource, is available to be consumed or matched by jobs in the queue if you set the queue's `resources_available.<resource name>` attribute to the available or matching value.

A queue resource is a job-wide resource. A job can request a queue resource for the entire job, but not for individual chunks.

An example of a job-wide resource is floating licenses, or any custom resource that is defined at both server and queue level.

### 5.4.2.3 Resources Defined at Both Server and Queue

Custom resources can be defined to be available either at vnodes or at both the server and queues. Consumable custom resources that are defined at the server and queue level have their consumption monitored at the server and queue level. In our example, if a job requests one *FloatingLicenses*, then the value of the `resources_assigned.FloatingLicenses` attribute is incremented by one at both the server and the queue in which the job resides.

### 5.4.2.4 Vnode Resources

A vnode resource, also called a vnode-level or host-level resource, is available only at vnodes. A vnode resource is a chunk-level resource, meaning that it can be requested for a job only inside of a chunk.

## 5.4.3 Consumable vs. Non-consumable Resources

A *consumable* resource is one that is reduced by being used. Consumable resources include `ncpus`, `mem` and `vmem` by default, and any custom resource defined with the `-n` or `-f` flags.

A *non-consumable* resource is not reduced through use, meaning that allocation to one job does not affect allocation to other jobs. The scheduler matches jobs to non-consumable resources. Examples of non-consumable resources are `walltime`, `file`, `cput`, `pcput`, `pmem`, `pvmem`, `nice`, or Boolean resources.

The following table shows which resource types are consumable:

**Table 5-1: Consumable and Non-consumable Resources**

| Resource Type | Consumable vs. Non-consumable |
|---------------|-------------------------------|
| Boolean       | Non-consumable                |
| duration      | Non-consumable                |
| float         | Consumable                    |
| long          | Consumable                    |
| Size          | Consumable                    |
| string        | Non-consumable                |
| string_array  | Non-consumable                |

## 5.4.4 Static vs. Dynamic Resources

Static resources are managed by PBS and have values that are fixed until you change them or until you change the hardware and MoM reports a new value for memory or number of CPUs.

Dynamic resources are not under the control of PBS, meaning that they can change independently of PBS. Dynamic resources are reported via a script; PBS runs a query to discover the available amount. Server dynamic resources use a script that runs at the server host. Host-level dynamic resources are updated via an `exechost_periodic` hook.

Static and dynamic resources can be available at the server or host level.

The default timeout for a server dynamic resource script is 30 seconds. You can specify a timeout for server dynamic resources in each scheduler's `server_dyn_res_alarm` attribute. If the script does not finish before the timeout, the scheduler uses a value of zero for the dynamic server resource. If you set the timeout to zero, the scheduler does not place a time limit on the script.

### 5.4.4.1 Dynamic Resource Caveats

- Dynamic resource values are displayed in `qstat`, but the value displayed is the last value retrieved, not the current value. Dynamic resources have no `resources_available.<resource name>` representation anywhere in PBS.
- Dynamic resources can take longer to discover because PBS runs a script to determine each one.

## 5.4.5 Requested vs. Default Resources

A job's requested resources are the resources explicitly requested by the job. Default resources are resources that you specify that each job should have if not requested. For example, you can specify that any job that does not request `walltime` gets 12 hours of `walltime`. For jobs that do request `walltime`, the default of 12 hours is not applied.

For information on default resources, see [section 5.9.3, “Specifying Job Default Resources”, on page 241](#) and [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#).

## 5.4.6 Shared vs. Non-shared Vnode Resources

### 5.4.6.1 Non-shared Vnode Resources

Most vnode resources are not shared. When a resource is defined at one vnode for use by jobs only at that vnode, the resource is not shared. For example, when `resources_available.ncpus` is set to `4` on a single-vnode machine, and no other vnodes have `resources_available.ncpus` defined as a pointer to this resource, this resource is not shared.

### 5.4.6.2 Shared Vnode Resources

When more than one vnode needs access to the same actual resource, that resource can be shared among those vnodes. The resource is defined at one vnode, and the other vnodes that supply the resource contain a pointer to that vnode. Any of the vnodes can supply that resource to a job, but only up to the amount where the total being used by jobs is less than or equal to the total available at the vnode where the resource is defined. For example, if you had a 4-vnode machine which had 8GB of memory, and wanted any single vnode to be able to supply up to 8GB to jobs, you would make the memory a shared resource. See [section 5.14.4.3, “Shared Host-level Resources”, on page 266](#).

## 5.4.7 Platform-specific vs. Generally Available Resources

Most PBS built-in resources are available on, and apply to, all supported platforms. However, PBS provides some resources specifically designed for a given platform. These platform-specific resources are not applicable to any other platform, and cannot be used on platforms other than the one(s) for which they are designed.

## 5.4.8 Job-wide vs. Chunk Resources

### 5.4.8.1 Job-wide Resources

A job-wide resource applies to the entire job, and is available at the server or queue, but not at the host level. Job-wide resources are requested outside of a select statement, using this form:

```
-l <resource name>=<value>
```

For example, to request one hour of `walltime` for a job:

```
-l walltime=1:00:00
```

Examples of job-wide resources are `walltime`, scratch space, and licenses.

### 5.4.8.2 Chunk Resources

A chunk resource applies to the part of the job running on that chunk, and is available at the host level. Chunk resources are requested inside a select statement. A single chunk is requested using this form:

```
-l select=<resource name>=<value>:<resource name>=<value>
```

For example, one chunk might have 2 CPUs and 4GB of memory:

```
-l select=ncpus=2:mem=4gb
```

To request multiples of a chunk, prefix the chunk specification by the number of chunks:

```
-l select=[number of chunks]<chunk specification>
```

For example, to request six of the previous chunk:

```
-l select=6:ncpus=2:mem=4gb
```

To request different chunks, concatenate the chunks using the plus sign ("+"):

```
-l select=[number of chunks]<chunk specification>+[number of chunks]<chunk specification>
```

For example, to request two kinds of chunks, one with 2 CPUs per chunk, and one with 8 CPUs per chunk, both kinds with 4GB of memory:

```
-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4GB
```

## 5.5 Resource Types

PBS supplies the following types of resources:

[Boolean](#)

[Duration](#)

[Float](#)

[Long](#)

[Size](#)

[String](#)

[String Array](#)

## 5.6 Resource Formats

Custom resources follow the same rules as built-in resources: custom resource names must be PBS NAMES, allowable values for float and long resources are the same as for built-in resources, and custom Boolean, time, size, string or string array resources must have the same format as built-in resources.

### Boolean

Name of Boolean resource is a string.

Values:

*TRUE, True, true, T, t, Y, y, 1*

*FALSE, False, false, F, f, N, n, 0*

### Duration

A period of time, expressed either as

*An integer whose units are seconds*

or

*[[hours:]minutes:]seconds[.milliseconds]*

in the form:

*[[[HH]HH:]MM:]SS[.milliseconds]*

Milliseconds are rounded to the nearest second.

### Float

Floating point. Allowable values: [+ -] 0-9 [[0-9] ...][.][[0-9] ...]

### Long

Long integer. Allowable values: 0-9 [[0-9] ...], and + and -

`<queue name>@<server name>`

### Size

Number of bytes or words. The size of a word is 64 bits.

Format: `<integer>[<suffix>]`

where *suffix* can be one of the following:

**Table 5-2: Size in Bytes**

| Suffix   | Meaning                | Size                                  |
|----------|------------------------|---------------------------------------|
| b or w   | Bytes or words         | 1                                     |
| kb or kw | Kilobytes or kilowords | 2 to the 10th, or 1024                |
| mb or mw | Megabytes or megawords | 2 to the 20th, or 1,048,576           |
| gb or gw | Gigabytes or gigawords | 2 to the 30th, or 1,073,741,824       |
| tb or tw | Terabytes or terawords | 2 to the 40th, or 1024 gigabytes      |
| pb or pw | Petabytes or petawords | 2 to the 50th, or 1,048,576 gigabytes |

Default: *bytes*

Note that when sorting vnodes, a scheduler rounds all resources of type **size** up to the nearest kb.

### String

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values: `[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ' { | } ~] ...]`

String resource values are case-sensitive. No limit on length.

### String Array

Comma-separated list of strings.

Strings in `string_array` may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

## 5.6.1 Resource Names

Resource names are case-sensitive PBS NAMES. Resource names can be 64 characters in length.

"PBS NAME" is a generic term, used to describe various PBS entities. For example, attribute names are PBS NAMES.

Must start with an alphabetic character, and may contain only the following: alpha-numeric, underscore ("\_"), or dash ("-").

Do not use PBS keywords as PBS NAMES.



## 5.7 Setting Values for Resources

### 5.7.1 How Resource Values are Set

PBS automatically collects information about some resources such as `ncpus` and `mem` and sets their initial values accordingly. If you explicitly set the value for a resource, that value is carried forth across server restarts.

Since the value for each dynamic resource is set by PBS to the value returned by a script or command, it makes sense set values for static resources only.

Resources that are not explicitly set can inherit their values from defaults. Jobs can inherit default resources; see [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#).

You set values for custom and built-in resources using the same methods. You can set resource values using the following methods:

- Using `qmgr`:

To set the available amount of a non-string\_array resource, use the `qmgr` command, either from the command line or within `qmgr`:

```
qmgr -c "set <object> resources_available.<resource name> = <value>"
Qmgr: set <object> resources_available.<resource name> = <value>
```

To set or change the available amount of a string\_array resource, use the `qmgr` command, either from the command line or within `qmgr`:

```
qmgr -c "set <object> resources_available.<resource name> = <value>"
qmgr -c 'set <object> resources_available.<resource name> = "<value,value>"'
qmgr -c 'set <object> resources_available.<resource name> += <value>'
qmgr -c 'set <object> resources_available.<resource name> -= <value>'
Qmgr: set <object> resources_available.<resource name> = <value>
Qmgr: set <object> resources_available.<resource name> = '<value,value>'
Qmgr: set <object> resources_available.<resource name> += <value>
Qmgr: set <object> resources_available.<resource name> -= <value>
```

To unset the value of an attribute:

```
qmgr -c "unset <object> resources_available.<resource name>"
Qmgr: unset <object> resources_available.<resource name>
```

where `<object>` is one of *server*, *queue*, *hook*, *node*, or *sched*.

For example, to set `resources_max.walltime` at the server to be 24 hours:

```
Qmgr: set server resources_max.walltime = 24:00:00
```

See [“qmgr” on page 152](#).

- Using a Version 2 configuration file; see [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).
- Setting the value in a hook; see [“Using Attributes and Resources in Hooks” on page 45 in the PBS Professional Hooks Guide](#).

#### 5.7.1.1 How Vnode Available Resource Values are Set

PBS stores values for the resources available at a vnode in that vnode's `resources_available.<resource name>` attribute.

### 5.7.1.1.i Vnode Resources Set by PBS

PBS automatically sets the value for certain resources available at each vnode, meaning that PBS sets the value for the vnode's `resources_available.<resource name>` attribute. For example, PBS automatically sets the value of `resources_available.ncpus` at each vnode. The following table lists the vnode resources that are set automatically by PBS.

**Table 5-3: Resources Set by PBS**

| Resource Name | Initial Value                                        | Notes                                                                                     |
|---------------|------------------------------------------------------|-------------------------------------------------------------------------------------------|
| arch          | <i>Value reported by OS</i>                          | Settable. If you unset the value, it remains unset until MoM is restarted.                |
| host          | <i>Short form of hostname in Mom vnode attribute</i> | Settable. If you unset the value, it remains unset until MoM is restarted.                |
| mem           | <i>Amount reported by OS</i>                         | Settable. If you unset the value, it remains unset until MoM is restarted.                |
| ncpus         | <i>Number of CPUs reported by OS</i>                 | Settable. If you unset this value, the MoM will reset it to the value reported by the OS. |
| router        | <i>Name of router, from topology file</i>            | Applies to vnodes on certain HPE systems only                                             |
| vnode         | <i>Name of the vnode</i>                             | Vnode name must be specified via the <code>qmgr create node</code> command.               |

### 5.7.1.1.ii Setting Vnode Resources Manually

You can set values for available vnode resources:

- You can set values for a vnode's `resources_available` in a hook. See ["Setting and Unsetting Vnode Resources and Attributes" on page 49 in the PBS Professional Hooks Guide](#). If you set a vnode resource in a hook, MoM will no longer update the resource.
- You can set values for a vnode's `resources_available` attribute in a Version 2 configuration file; see [section 3.4.3, "Version 2 Vnode Configuration Files", on page 46](#).
- You can set most values for a vnode's `resources_available` attribute using `qmgr`, but not for `resources_available.host`; see ["Operating on Attributes and Resources" on page 161 of the PBS Professional Reference Guide](#).

### 5.7.1.2 Setting Server and Queue Resource Values

You can set resources, such as default and available resources, for queues and for the server, using `qmgr`:

```
Qmgr: set queue <queue name> resources_default<resource name> = <value>
Qmgr: set queue <queue name> resources_available.<resource name> = <value>
Qmgr: set server resources_available.<resource name> = <value>
```

### 5.7.1.3 Setting Job Resources

#### 5.7.1.3.i Setting Requested Resource Values

Job resources, stored in the `Resource_List` job attribute, can be set initially at submission in the job request. You can augment or change these values. You can set values for a job's `Resource_List` attribute using hooks. See ["Setting Job Resources in Hooks" on page 50 in the PBS Professional Hooks Guide](#).

### 5.7.1.3.ii Setting Used Resource Values

The resources used by a job are set in the job's `resources_used` attribute

You can set values for a job's `resources_used` attribute using hooks. These values will appear in the accounting log and in `qstat -f` output. See ["Setting Job Resources in Hooks" on page 50 in the PBS Professional Hooks Guide](#).

### 5.7.1.3.iii Setting Estimated Values

If the `PBS_est` built-in hook is enabled, PBS automatically sets the value of the `estimated.start_time` job resource to the estimated start time for each job. Otherwise, PBS sets the value only for top jobs.

## 5.7.2 Setting Values for Static Resources

To set the value for a vnode, queue, or server resource, use the `qmgr` command to set the value of the appropriate `resources_available.<resource name>` attribute.

Example 5-1: Set the value of `floatlicenses` at the server to `10`:

```
Qmgr: set server resources_available.floatlicenses = 10
```

Example 5-2: Set the value of `RunsMyApp` to `True` at the vnode named `vnode1`:

```
Qmgr: set node vnode1 resources_available.RunsMyApp = True
```

### 5.7.2.1 Restrictions on Setting Values for Static Resources

When setting static vnode resources on multi-vnode machines, follow the rules in [section 3.4.5, "Configuring Vnode Resources", on page 51](#).

## 5.7.3 Setting Values for String Arrays

A string array that is defined on vnodes can be set to a different set of strings on each vnode.

Example of defining and setting a string array:

- Define a new resource:  

```
Qmgr: create resource foo_arr type=string_array, flag=h
```
- Setting via `qmgr`:  

```
Qmgr: set node n4 resources_available.foo_arr="f1, f3, f5"
```
- Vnode `n4` has 3 values of `foo_arr`: `f1`, `f3`, and `f5`. We add `f7`:  

```
Qmgr: set node n4 resources_available.foo_arr+=f7
```
- Vnode `n4` now has 4 values of `foo_arr`: `f1`, `f3`, `f5` and `f7`.
- We remove `f1`:  

```
Qmgr: set node n4 resources_available.foo_arr-=f1
```
- Vnode `n4` now has 3 values of `foo_arr`: `f3`, `f5`, and `f7`.
- Submission:  

```
qsub -l select=1:ncpus=1:foo_arr=f3
```

### 5.7.4 When Resource Changes Take Effect

If you change the value of a resource via the `qmgr` command, the change takes effect immediately.

If you change the value of a resource in a configuration file, the change takes effect the next time the configuration file is read.

## 5.7.5 Caveats for Setting Resource Values

- It is not recommended to set the value for `resources_available.ncpus`. The exception is when you want to oversubscribe CPUs. See [section 8.6.6.1.iii, “How To Share CPUs”, on page 416](#).
- Do not attempt to set values for `resources_available.<resource name>` for dynamic resources.
- Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set `resources_available.walltime` at the server to `10:00:00`, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

### 5.7.5.1 Caveats for Setting Resource Values at Multi-vnode Machines

- When setting static vnode resources on multi-vnode machines, follow the rules in [section 3.4.5, “Configuring Vnode Resources”, on page 51](#).
- It is not recommended to change the value of `ncpus` at vnodes on a multi-vnoded machine.
- On the parent vnode, all values for `resources_available.<resource name>` should be **zero (0)**, unless the resource is being shared among other vnodes via indirection.

## 5.8 Overview of Ways Resources Are Used

Resources are used in several ways in PBS. The following table lists the ways resources are used, and gives links to the section describing each one:

**Table 5-4: How Resources Are Used**

| Use                                     | Description                                                                                         |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------|
| Allocation to and use by jobs           | See <a href="#">section 5.9, “Resources Allocated to Jobs and Reservations”, on page 240</a>        |
| Limiting job resource usage             | See <a href="#">section 5.15.2, “Placing Resource Limits on Jobs”, on page 300</a>                  |
| Restricting access to server and queues | See <a href="#">section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251</a> |
| Routing jobs                            | See <a href="#">section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 28</a>     |
| Describing topology and placing jobs    | See <a href="#">section 5.11, “Using Resources for Topology and Job Placement”, on page 250</a>     |
| Setting job execution priority          | See <a href="#">section 5.12, “Using Resources to Prioritize Jobs”, on page 251</a>                 |
| Reserving resources ahead of time       | See <a href="#">section 4.9.37, “Reservations”, on page 196</a> .                                   |
| Tracking and controlling allocation     | See <a href="#">section 5.10, “Using Resources to Track and Control Allocation”, on page 249</a>    |
| Determining job preemption priority     | See <a href="#">section 4.9.33, “Using Preemption”, on page 180</a>                                 |

## 5.8.1 How the Scheduler Uses Resources

How the scheduler uses resources is described in [section 4.9.28, “Matching Jobs to Resources”, on page 159](#).

## 5.8.2 Advice on Using string and string\_array Resources

Resource names are case-sensitive.

### 5.8.2.1 Using string Resources

Format of **string** resources:

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values: `[_a-zA-Z0-9][[_a-zA-Z0-9!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~] ...`

String resource values are case-sensitive. No limit on length.

Non-consumable.

We do not recommend using non-printing characters.

When using `qsub -l <string resource>=<string value>`, you must escape string values for both `qsub` and the shell. Example:

```
qsub -lteststring=\"abc def\"
```

The final quote should be single, not double.

### 5.8.2.2 Using string\_array Resources

Format of **string\_array** resources:

Comma-separated list of strings.

Strings in **string\_array** may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string.

The value of `resources_default.<string array resource>` can only be one string.

## 5.9 Resources Allocated to Jobs and Reservations

Resources allocated to jobs provide the job with amounts of CPUs and memory to be consumed by the job's processes, as well as qualities such as architecture and host. The resources allocated to a job are those that the job requests and those that are assigned to it through resource defaults that you define, or by hooks you write.

Jobs use resources at the job-wide and chunk level. Job-wide resources such as `walltime` or `vmem` are applied to and requested by the job as a whole. Chunk-level resources, such as `ncpus`, are applied and requested in individual chunks.

Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See [“Resources Built Into PBS” on page 265 of the PBS Professional Reference Guide](#) and [“Requesting Resources”, on page 53 of the PBS Professional User’s Guide](#).

Jobs inherit resource defaults for resources not explicitly requested. See [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#).

Chunk-level resources are made available at the host (vnode) level by defining them via `resources_available.<resource name>` at the vnode, and are requested using `-l select=<resource name>=<value>`.

Job-wide resources are made available by defining them via `resources_available.<resource name>` at the queue or server. These resources are requested using `-l <resource name> =<value>`.

The scheduler matches requested resources with available resources, according to rules defined by the administrator.

When a job is requesting a string array resource, it can request only one of the values set in the string array resource. The job will only be placed on a vnode where the job's requested string matches one of the values of the string array resource. For example, if the resource named *Colors* is set to *"red, blue, green"* on vnode V1, and *"red, blue"* on V2:

- A job can request only one of *"red"*, *"blue"*, or *"green"*
- A job requesting `Colors=green` will only be placed on V1

## 5.9.1 Allocating Chunks

Chunks cannot be split across hosts. Chunks can be made up of vchunks. If a chunk is broken up over multiple vnodes, all participating vnodes must belong to the same execution host. Each vnode supplies a vchunk. These participating vnodes are supplying the vchunks that make up the chunk. A chunk defines a logical set of resources, for example, those needed for an MPI task. The resources must come from a single host, but if the requested resources exceed that of any one vnode, the physical resources can be taken from multiple vnodes on the same host.

## 5.9.2 Resources Requested by Job

The job's `Resource_List` attribute lists the following resources requested by the job:

- Job-wide resources explicitly requested by the job, inherited from defaults, or assigned by hooks
- The following built-in chunk-level resources explicitly requested by the job, inherited from defaults, or assigned by hooks:
  - `mpiprocs`
  - `ncpus`
  - `mem`
  - `vmem`
- Custom vnode-level (chunk-level) resources that have the `n`, `q`, or `f` flags set, explicitly requested by the job, inherited from defaults, or assigned by hooks

## 5.9.3 Specifying Job Default Resources

You can specify which resources are automatically added to job resource requests. When a job does not request a specific resource, the default value for that resource is automatically added to the job's resource request.

You can also use hooks to add resources to a job's resource request, but we describe that elsewhere in the PBS Professional Hooks Guide.

The amount of each resource a job is allowed to use is the amount in its resource request. See [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#). Therefore you may wish to add default limits on resource usage. This is done by adding default resources to the job's resource request. For example, if a job does not request `walltime`, but you do not want jobs not specifying `walltime` to run for more than 12 hours, you can specify a default of 12 hours for `walltime`. Jobs that do specify `walltime` do not inherit this default; they keep their requested amount.

You can use default resources to manage jobs. For example, if you want to keep track of and limit the number of jobs using something such as a disk arm, you can have each job using the disk arm automatically request one counting resource. Then you can place a limit on the amount of this resource that can be in use at one time. This technique is described in [section 5.10, “Using Resources to Track and Control Allocation”, on page 249](#).

Default resources can be defined for the server and for each queue. Default resources defined at the server are applied to all jobs. Default resources at a queue are applied only to the jobs that are in that queue.

Default resources on the server and queue can be job-wide, which is the same as adding `-l <resource name>=<value>` to the job's resource request, or they can be chunk resources, which is the same as adding `:<resource name>=<value>` to a chunk.

Job-wide resources are specified via `resources_default` on the server or queue, and chunk resources are specified via `default_chunk` on the server or queue. You can also specify default resources to be added to any `qsub` arguments. In addition, you can specify default placement of jobs.

### 5.9.3.1 Specifying Job-wide Default Resources at Server

To specify a server-level job-wide default resource, use the `qmgr` command to set the server's `resources_default` attribute:

```
Qmgr: set server resources_default.<resource name>=<value>
```

For example, to set the default architecture on the server:

```
Qmgr: set server resources_default.arch=linux
```

### 5.9.3.2 Setting Server and Queue Default Job Chunk Resource Values

If a job doesn't request a specific resource, PBS can assign a default value you specify. PBS stores default values for job chunk resources in the `default_chunk.<resource name>` attribute for the server and each queue.

PBS automatically sets the value for `default_chunk.ncpus` to 1 at the server and queues.

#### 5.9.3.2.i Specifying Chunk Default Resources at Server

To specify a server-level chunk default resource, use the `qmgr` command to set the server's `default_chunk` attribute:

```
Qmgr: set server default_chunk.<resource name>=<value>
```

For example, if you want all chunks that don't specify `ncpus` or `mem` to inherit the values you specify:

```
Qmgr: set server default_chunk.ncpus=1
Qmgr: set server default_chunk.mem=1gb
```

#### 5.9.3.2.ii Specifying Chunk Default Resources at Queue

To specify a queue-level chunk default resource, use the `qmgr` command to set the queue's `default_chunk` attribute:

```
Qmgr: set queue <queue name> default_chunk.<resource name>=<value>
```

For example, if you want all chunks that don't specify `ncpus` or `mem` to inherit the values you specify:

```
Qmgr: set queue small default_chunk.ncpus=1
Qmgr: set queue small default_chunk.mem=512mb
```



### 5.9.3.3 Specifying Job-wide Default Resources at Queue

To specify a default for a job-wide resource at a queue, use the `qmgr` command to set the queue's `resources_default` attribute:

```
Qmgr: set queue <queue name> resources_default.<resource name> = <value>
```

### 5.9.3.4 Specifying Default `qsub` Arguments

You can set defaults for any `qsub` arguments not explicitly requested by each job. You do this at the server by using the `qmgr` command to set the server's `default_qsub_arguments` attribute:

```
Qmgr: set server default_qsub_arguments=<string containing arguments>
```

For example, to set the default for the `Rerunnable` job attribute in each job's resource request, and the name of the job:

```
Qmgr: set server default_qsub_arguments= "-r y -N MyJob"
```

Or to set a default Boolean in each job's resource request so that jobs don't run on *Red* unless they explicitly ask to do so:

```
Qmgr: set server default_qsub_arguments="-l Red=False"
```

### 5.9.3.5 Specifying Default Job Placement

You can specify job placement defaults at both the server and queue level. You use the `qmgr` command to set the `resources_default.place` attribute at the server or queue:

```
Qmgr: set queue <queue name> resources_default.place=<value>
```

For example, to set the default job placement for a queue:

```
Qmgr: set queue Q1 resources_default.place=free
```

When setting default placement involving a colon, enclose the value in double quotes:

```
Qmgr: set server resources_default.place="<value>"
```

For example, to set default placement at the server to `pack:shared`, do the following:

```
Qmgr: set server resources_default.place= "pack:shared"
```

See ["Specifying Job Placement", on page 66 of the PBS Professional User's Guide](#) for detailed information about how `-l place` is used.

### 5.9.3.6 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of `resources_max.<resource name>` is used. If this is set only at the server, the job inherits the value set at the server.

### 5.9.3.7 Default Resource Caveats

- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue `resources_default.<resource name>` attribute.
- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue `resources_default` or `default_chunk` as an input argument.
- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a `select` and a `place`. This means that if you specify a default placement, such as `excl`, with `-lplace=excl`, and the user specifies an arrangement, such as `pack`, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for `select`; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

## 5.9.4 Allocating Default Resources to Jobs

Jobs inherit default resources, job-wide or per-chunk, with the following order of precedence.

**Table 5-5: Order In Which Default Resources Are Assigned to Jobs**

| Order of assignment | Default value                           | Affects Chunks? | Job-wide?    |
|---------------------|-----------------------------------------|-----------------|--------------|
| 1                   | Default <code>qsub</code> arguments     | If specified    | If specified |
| 2                   | Queue's <code>default_chunk</code>      | Yes             | No           |
| 3                   | Server's <code>default_chunk</code>     | Yes             | No           |
| 4                   | Queue's <code>resources_default</code>  | No              | Yes          |
| 5                   | Server's <code>resources_default</code> | No              | Yes          |
| 6                   | Queue's <code>resources_max</code>      | No              | Yes          |
| 7                   | Server's <code>resources_max</code>     | No              | Yes          |

See [section 5.9.3, “Specifying Job Default Resources”, on page 241](#) for how to set these defaults.

For each chunk in the job's selection statement, first default `qsub` arguments are applied, then queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. The chunk defaults are specified in the `default_chunk.<resource name>` server or queue attribute.

For example, if the queue in which the job is enqueued has the following defaults defined,

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

then a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the `default_chunk` elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb
```

In the above, `mem=2gb` and `ncpus=1` are inherited from `default_chunk`.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults, then against the queue's `resources_max.<resource name>`, then against the server's `resources_max.<resource name>`. If a default or maximum resource is defined which is not specified in the resource request, it is added to the resource request.

### 5.9.4.1 Default Resource Allocation for min\_walltime and max\_walltime

The min\_walltime and max\_walltime resources inherit values differently. A job can inherit a value for max\_walltime from resources\_max.walltime; the same is not true for min\_walltime. This is because once a job is shrink-to-fit, PBS can use a walltime limit for max\_walltime. See [section 4.9.42.3.ii, “Inheriting Values for min\\_walltime and max\\_walltime”, on page 212](#).

### 5.9.4.2 Default Resource Allocation Caveats

- Resources assigned from the default\_qsub\_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default\_qsub\_arguments. Be aware that creating custom resources with permission flags and then using these in the default\_qsub\_arguments server attribute can cause jobs to be rejected. See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).
- Default qsub arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as *excl*, with `-lplace=excl`, and the user specifies an arrangement, such as *pack*, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for *select*; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

### 5.9.4.3 Moving Jobs Between Queues or Servers Changes Defaults

If the job is moved from the current queue to a new queue or server, any default resources in the job's Resource\_List inherited from the current queue or server are removed. The job then inherits any new default resources. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Jobs may be moved between servers when peer scheduling is in operation. Given the following set of queue and server default values:

- Server  
resources\_default.ncpus=1
- Queue QA  
resources\_default.ncpus=2  
default\_chunk.mem=2GB
- Queue QB  
default\_chunk.mem=1GB  
no default for ncpus

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

Example 5-3: Submission:

- `qsub -l ncpus=1 -lmem=4gb`
- In QA:  
`select=1:ncpus=1:mem=4gb`

- No defaults need be applied
- In QB:  
`select=1:ncpus=1:mem=4gb`
- No defaults need be applied

Example 5-4: Submission:

- ```
qsub -l ncpus=1
```
- In QA:
`select=1:ncpus=1:mem=2gb`
- Picks up 2GB from queue default chunk and 1 ncpus from qsub
 - In QB:
`select=1:ncpus=1:mem=1gb`
- Picks up 1GB from queue default_chunk and 1 ncpus from qsub

Example 5-5: Submission:

- ```
qsub -lmem=4gb
```
- In QA:  
`select=1:ncpus=2:mem=4gb`  
- Picks up 2 ncpus from queue level job-wide resource default and 4GB mem from qsub
  - In QB:  
`select=1:ncpus=1:mem=4gb`  
- Picks up 1 ncpus from server level job-wide default and 4GB mem from qsub

Example 5-6: Submission:

- ```
qsub -lnodes=4
```
- In QA:
`select=4:ncpus=1:mem=2gb`
- Picks up a queue level default memory chunk of 2GB. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)
 - In QB:
`select=4:ncpus=1:mem=1gb`
(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

Example 5-7: Submission:

- ```
qsub -l mem=16gb -lnodes=4
```
- In QA:  
`select=4:ncpus=1:mem=4gb`  
(This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)
  - In QB:  
`select=4:ncpus=1:mem=4gb`  
(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

## 5.9.5 Dynamic Resource Allocation Caveats

When a job requests a dynamic resource, PBS checks to see how much of the resource is available, but cannot know how much will be used by another job while this job executes. This can lead to a resource shortage. For example, there is 20GB of scratch on a disk, no jobs are running, and a job requests 15GB. This job writes to 5GB during the first part of its execution, then another job requests 10GB. The second job is started by PBS, because there is 15GB available. Now there is a shortage of scratch space.

You can avoid this problem by configuring a static consumable resource to represent scratch space. Set it to the amount of available scratch space. See [section 5.14.5.3, “Static Server-level Scratch Space”, on page 270](#) and [section 5.14.5.4, “Static Host-level Scratch Space”, on page 270](#).

## 5.9.6 Period When Resource is Used by Job

### 5.9.6.1 Exiting Job Keeps Resource

A job that is exiting is still consuming resources assigned to it. Those resources are available for other jobs only when the job is finished.

### 5.9.6.2 Job Suspension and Resource Usage

#### 5.9.6.2.i Resource Usage on Suspension

When suspended, a job is not executing and is not charged for walltime.

#### 5.9.6.2.ii Releasing Resources on Suspension

You can specify which consumable resources should be released by PBS when a job is suspended, using the `restrict_res_to_release_on_suspend` server attribute. In this attribute you list all of the resources that should be released when a job is suspended. If you leave this attribute unset, PBS releases all of a job's consumable resources when the job is suspended. This does not include the licenses used by the application, if any. You can modify the list to add and remove resources using "+" and "-" operators.

Server attribute where you specify resources to be released:

#### `restrict_res_to_release_on_suspend`

Comma-separated list of consumable resources to be released when jobs are suspended. If unset, all consumable resources are released on suspension.

Format: *Comma-separated list*

Python type: *list*

Default value: unset, meaning all consumable resources are released on suspension

You can see which resources have been released for a given job by looking at these job attributes:

#### `resources_released`

Listed by vnode, consumable resources that were released when the job was suspended. Populated only when `restrict_res_to_release_on_suspend` server attribute is set. Set by server.

Format: String of the form: (`<vnode>:<resource name>=<value>:<resource name>=<value>:...`)+(`<vnode>:<resource name>=<value>:...`)

Python type: *str*

**resource\_released\_list**

Sum of each consumable resource requested by the job that was released when the job was suspended. Populated only when `restrict_res_to_release_on_suspend` server attribute is set. Set by server. You will also see this amount released at the queue and/or server.

Format: String of the form: *resource\_released\_list.<resource name>=<value>,resource\_released\_list.<resource name>=<value>, ...*

Python type: *pbs.pbs\_resource*

Jobs are suspended when they are preempted and via `qsig -s suspend`.

A job is resumed only when sufficient resources are available. When a person resumes a job via `qsig -s resume`, the job is not run until resources are available.

### 5.9.6.2.iii Suspension/resumption Resource Caveats

Dynamic resources can cause problems with suspension and resumption of jobs.

When a job is suspended, its resources are freed, but the scratch space written to by the job is not available.

A job that uses scratch space may not suspend and resume correctly. This is because if the job writes to scratch, and is then suspended, when PBS queries for available scratch to resume the job, the script may return a value too small for the job's request. PBS cannot determine whether the job itself is the user of the scratch space; PBS can only determine how much is still unused. If a single suspended job has left less scratch space available than it requests, that job cannot be resumed.

The above is true for any dynamic resource, such as application licenses.

When suspended, a job is not executing and is not charged for walltime.

### 5.9.6.3 Shrink-to-fit Jobs Get walltime When Executed

PBS computes the `walltime` value for each shrink-to-fit job when the scheduler runs the job, not before. See [section 4.9.42.3.iii, "Setting walltime for Shrink-to-fit Jobs", on page 212](#).

## 5.10 Using Resources to Track and Control Allocation

You can use resources to track and control usage of things like hardware and application licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. There is more than one way to accomplish this.

Example 5-8: You can set a complex-wide limit on the number of jobs using a type of complex-wide floating application license. This example uses a single queue for the entire complex. This method requires job submitters to request one of a `floatlicensecount` resource in order to be able to use the license. To set a complex-wide limit, take the following steps:

1. Create a custom static integer license resource that will be tracked at the server and queue:
  - a. Use `qmgr` to create the resource:
 

```
Qmgr: create resource floatlicensecount type=long, flag=q
```
  - b. Add the resource to the `resources:` line in `<sched_priv directory>/sched_config:`

```
resources: "[...], floatlicensecount"
```
2. HUP the scheduler:
 

```
kill -HUP <scheduler PID>
```
3. Set the available resource at the server using `qmgr`. If you have enough floating licenses for 4 jobs:
 

```
Qmgr: set server resources_available.floatlicensecount = 4
```
4. Inform job submitters that jobs using they must request one job-wide `floatlicensecount` resource via the following:
 

```
qsub -l floatlicensecount=1
```

The scheduler will schedule up to 4 jobs at a time using the licenses. You do not need to set the resource at any queue.

Example 5-9: Here, your job submitters don't need to request a counting resource. Jobs are routed based on the size of the request for memory, and the counting resource is inherited from a default. In this example, we are limiting the number of jobs from each group that can use a particular vnode that has a lot of memory. This vnode is called *Mem-Node*.



Jobs that request 8GB or more of memory are routed into queue *BigMem*, and inherit a default counting resource called *memcount*. All other jobs are routed into queue *SmallMem*. The routing queue is called *RouteQueue*.

1. Create a custom static integer *memcount* resource that will be tracked at the server and queue:

- a. Use *qmgr* to create the resource:

```
Qmgr: create resource memcount type=long, flag=q
```

- b. Add the resource to the resources: line in <sched\_priv directory>/sched\_config:

```
resources: "[...], memcount"
```

2. HUP the scheduler:

```
kill -HUP <scheduler PID>
```

3. Set limits at *BigMem* and *SmallMem* so that they accept the correct jobs:

```
Qmgr: set queue BigMem resources_min.mem = 8gb
```

```
Qmgr: set queue SmallMem resources_max.mem = 8gb
```

4. Set the order of the destinations in the routing queue so that *BigMem* is tested first, so that jobs requesting exactly 8GB go into *BigMem*:

```
Qmgr: set queue RouteQueue route_destinations = "BigMem, SmallMem"
```

5. Set the available resource at *BigMem* using *qmgr*. If you want a maximum of 6 jobs from *BigMem* to use *MemNode*:

```
Qmgr: set queue BigMem resources_available.memcount = 6
```

6. Set the default value for the counting resource at *BigMem*, so that jobs inherit the value:

```
Qmgr: set queue BigMem resources_default.memcount = 1
```

7. Associate the vnode with large memory with the *BigMem* queue. See [section 4.9.2, “Associating Vnodes with Queues”](#), on page 107.

The scheduler will only schedule up to 6 jobs from *BigMem* at a time on the vnode with large memory.

## 5.11 Using Resources for Topology and Job Placement

Using the topology information in the server's *node\_group\_key* attribute, PBS examines the values of resources at vnodes, and uses those values to create placement sets. Jobs are assigned to placement sets according to their resource requests. Users can specify particular placement sets by requesting the resources that define that particular placement set. For example, if the switch named *A25* connects the desired set of vnodes, a user can request the following:

```
-l switch=A25
```

See [section 4.9.32, “Placement Sets”](#), on page 168.

### 5.11.1 Restrictions on Using Resources for Job Placement

Only vnode-level resources can be used to direct jobs to particular vnodes.

## 5.12 Using Resources to Prioritize Jobs

You can define the formula the scheduler uses to compute job execution priorities. Elements in this formula can be inherited default custom resources. These resources must be job-wide numeric resources, or consumable host-level resources. See [section 5.9.3, “Specifying Job Default Resources”, on page 241](#) and [section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 151](#).

You can make jobs inherit numeric resources according to non-numeric qualities, such as the job owner's group or whether the job requests a Boolean or string resource. You can do this by either of the following methods:

- Use a hook to identify the jobs you want and alter their resource requests to include the custom resources for the formula. See the PBS Professional Hooks Guide
- Use a routing queue and minimum and maximum resource limits to route jobs to queues where they inherit the default custom resources for the formula. See [section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 28](#)

For details on how job execution priority is calculated, see [section 4.9.16, “Calculating Job Execution Priority”, on page 136](#).

For a complete description of how PBS prioritizes jobs, see [section 4.3.5, “Job Prioritization and Preemption”, on page 67](#).

## 5.13 Using Resources to Restrict Server or Queue Access

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than the complex or a queue can supply. You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using them in the job sorting formula or to route jobs to particular vnodes.

You set a maximum for each resource at the server using the `resources_max.<resource name>` server attribute; there is no `resources_min.<resource name>` at the server.

You can set a minimum and a maximum for each resource at each queue using the `resources_min.<resource name>` and `resources_max.<resource name>` queue attributes.

Job resource requests are compared to resource limits the same way, whether at the server or a queue. For a complete description of how jobs are tested against limits, see [section 2.3.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 29](#).

Job resource requests are compared first to queue admittance limits. If there is no queue admittance limit for a particular resource, the job's resource request is compared to the server's admittance limit.

### 5.13.1 Admittance Limits for `walltime`, `min_walltime`, and `max_walltime`

Because `min_walltime` and `max_walltime` are themselves limits, they behave differently from other time-based resources. When a shrink-to-fit job (a job with a value for `min_walltime`) is compared to server or queue limits, the following must be true in order for the job to be accepted:

- Both `min_walltime` and `max_walltime` must be greater than or equal to `resources_min.walltime`.
- Both `min_walltime` and `max_walltime` must be less than or equal to `resources_max.walltime`.

You cannot set `resources_min` or `resources_max` for `min_walltime` or `max_walltime`.

## 5.13.2 Restrictions on Resources Used for Admittance

For a list of resources that are compared to admittance limits, see [section 2.3.6.4.iii, “Resources Used for Routing and Admittance”, on page 29](#). For information on using strings, string arrays, and Booleans for admittance controls, see [section 2.3.6.4.iv, “Using String, String Array, and Boolean Values for Routing and Admittance”, on page 30](#).

## 5.14 Custom Resources

You can define, that is, create, new resources within PBS. This section describes how to define and use custom resources. Once new resources are defined, jobs may request these new resources and the scheduler can schedule on the new resources.

Using this feature, it is possible to schedule resources where the number or amount available is outside of PBS's control.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#). A user will not be able to print or list custom resources which have been made either invisible or unalterable.

PBS provides certain custom resources that are designed to reflect resources or properties found on specific systems. Do not create custom resources with the names that PBS uses for these resources. See [“Resources Built Into PBS” on page 265 of the PBS Professional Reference Guide](#).

### 5.14.1 How to Use Custom Resources

Custom static resources can be server-level or host-level. They can also be shared or not. Custom dynamic resources can be server-level and shared or not.

#### 5.14.1.1 Choosing the Resource Category

Use dynamic resources for quantities that PBS does not control, such as externally-managed application licenses or scratch space. PBS runs a script or program that queries an external source for the amount of the resource available and returns the value via `stdout`. Use static resources for things PBS does control. PBS tracks these resources internally.

Use server-level resources for things that are not tied to specific hosts, that is, they can be available to any of a set of hosts. An example of this is a floating application license.

##### 5.14.1.1.i Examples of Configuring a Custom Resource

The following table gives examples of configuring each kind of custom resource:

**Table 5-6: Examples of Configuring Custom Resources**

| Use for Resource                                    | Link to Example                                                                                                       |
|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| License: Floating, externally-managed               | See <a href="#">section 5.14.6.3.i, “Example of Floating, Externally-managed License”, on page 271</a>                |
| License: Floating, externally-managed with features | See <a href="#">section 5.14.6.3.ii, “Example of Floating, Externally-managed License with Features”, on page 272</a> |
| License: Floating, PBS-managed                      | See <a href="#">section 5.14.6.3.iii, “Example of Floating License Managed by PBS”, on page 273</a>                   |

**Table 5-6: Examples of Configuring Custom Resources**

| Use for Resource                 | Link to Example                                                                                                                                                                         |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| License: Node-locked, per-host   | See <a href="#">section 5.14.6.4.iv, “Example of Per-host Node-locked Licensing”, on page 275</a>                                                                                       |
| License: Node-locked, per-CPU    | See <a href="#">section 5.14.6.4.vi, “Example of Per-CPU Node-locked Licensing”, on page 277</a>                                                                                        |
| License: Node-locked, per-use    | See <a href="#">section 5.14.6.4.v, “Example of Per-use Node-locked Licensing”, on page 276</a>                                                                                         |
| FPGAs                            | See <a href="#">section 5.14.8, “Using FPGAs”, on page 282</a>                                                                                                                          |
| GPUs                             | See <a href="#">section 5.14.7, “Using GPUs”, on page 279</a>                                                                                                                           |
| Scratch space: shared            | See <a href="#">section 5.14.5.1, “Dynamic Server-level (Shared) Scratch Space”, on page 269</a> and <a href="#">section 5.14.5.3, “Static Server-level Scratch Space”, on page 270</a> |
| Scratch space: local to a host   | See <a href="#">section 5.14.5.2, “Dynamic Host-level Scratch Space”, on page 270</a> and <a href="#">section 5.14.5.4, “Static Host-level Scratch Space”, on page 270</a>              |
| Generic dynamic server-level     | See <a href="#">section 5.14.3.1.iii, “Example of Configuring Dynamic Server-level Resource”, on page 264</a>                                                                           |
| Generic static server-level      | See <a href="#">section 5.14.3.2.i, “Example of Configuring Static Server-level Resource”, on page 265</a>                                                                              |
| Generic dynamic host-level       | See <a href="#">section 5.14.4.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 265</a>                                                                               |
| Generic static host-level        | See <a href="#">section 5.14.4.2.i, “Example of Configuring Static Host-level Resource”, on page 266</a>                                                                                |
| Generic shared static host-level | See <a href="#">section 5.14.4.3.iii, “Configuring Shared Static Resources”, on page 267</a>                                                                                            |

## 5.14.1.2 Dynamic Custom Resources

A dynamic resource is one which is not under the control of PBS, meaning it can change independently of PBS. In order to use a dynamic resource, PBS must run a query to discover the available amount of that resource. Dynamic custom resources can be defined at the server.

### 5.14.1.2.i Dynamic Server-level Custom Resources

A dynamic server-level custom resource is used to track a resource that is available at the server. You use a dynamic server-level resource to track something that is not under the control of PBS, and changes outside of PBS, for example, floating application licenses. At each scheduler cycle, the scheduler runs a script at the server host to determine the available amount of that resource. Server-level custom resources are used as job-wide resources.

### 5.14.1.2.ii Dynamic Host-level Custom Resources

A dynamic host-level custom resource is used to track a resource that is available at the execution host or hosts. You use a dynamic host-level resource for a resource that is not under the control of PBS, and changes outside of PBS, for example, scratch space. You use an `exechost_periodic` hook to update the value of the resource being tracked. Host-level dynamic resources are used inside chunks.

## 5.14.1.3 Static Custom Resources

A static resource is one which is under the control of PBS. Any changes to the value are performed by PBS or by the administrator. Static custom resources are defined ahead of time, at the server, queues or vnodes.

### 5.14.1.3.i Static Custom Resources

Static custom resource values at vnode, queue and server are set via `qmgr`, by setting `resources_available.<custom resource name> = <value>`. These resources are available at the server, queues, or vnodes.

### 5.14.1.4 Shared Vnode Resources

A shared vnode resource is managed at one vnode, but available to be used by jobs at others. This allows flexible allocation of the resource. See [section 5.14.4.3, “Shared Host-level Resources”, on page 266](#) for information on resources shared across vnodes.

### 5.14.1.5 Using Custom Resources for Application Licenses

The following table lists application licenses and what kind of custom resource to define for them. See [section 5.14.6, “Supplying Application Licenses”, on page 270](#) for specific instructions on configuring each type of license and examples of configuring custom resources for application licenses.

**Table 5-7: Custom Resources for Application Licenses**

| Floating or Node-locked | Unit Being Licensed     | How License is Managed   | Level  | Resource Type |
|-------------------------|-------------------------|--------------------------|--------|---------------|
| Floating (site-wide)    | Token                   | External license manager | Server | Dynamic       |
| Floating (site-wide)    | Token                   | PBS                      | Server | Static        |
| Node-locked             | Host                    | PBS                      | Host   | Static        |
| Node-locked             | CPU                     | PBS                      | Host   | Static        |
| Node-locked             | Instance of Application | PBS                      | Host   | Static        |

### 5.14.1.6 Using Custom Resources for Scratch Space

You can configure a custom resource to report how much scratch space is available on machines. Jobs requiring scratch space can then be scheduled onto machines which have enough. This requires an `exechost_periodic` hook to keep the resource updated. See the *PBS Professional Hooks (Plugins) Guide*.

## 5.14.2 Defining New Custom Resources

You can define new custom resources as follows:

- To define any custom resource, you can use `qmgr`.
- To define custom host-level non-consumable resources at vnodes, you can use hooks; see [“Adding Custom Host-level Resources” on page 69 in the PBS Professional Hooks Guide](#).

### 5.14.2.1 Defining and Setting Static and Dynamic Custom Resources

The following table lists the differences in defining and setting static and dynamic custom resources at the server, queue and host level.

**Table 5-8: Defining and Setting New Custom Resources**

| Resource Type | Server-level                                                                                      | Queue-level               | Host-level                                                        |
|---------------|---------------------------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------|
| static        | Set via <code>qmgr</code>                                                                         | Set via <code>qmgr</code> | Set via <code>qmgr</code> or hook                                 |
| dynamic       | Add to <code>server_dyn_res</code> line in <code>&lt;sched_priv directory&gt;/sched_config</code> | Cannot be used.           | Use an <code>exechost_periodic</code> hook to update the resource |

### 5.14.2.2 Custom Resource Values

The rules for custom resource values are the same as for built-in resource values. See [“Resource Formats” on page 359 of the PBS Professional Reference Guide](#).

If a `string` resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want. If the `string` resource value contains commas, the string must be enclosed in an additional set of quotes so that the command (e.g. `qsub`, `qalter`) will parse it correctly. If the `string` resource value contains quotes, plus signs, equal signs, colons or parentheses, the `string` resource value must be enclosed in yet another set of additional quotes.

### 5.14.2.3 Specifying Resource Level and Consumability

When you define a custom resource, you specify whether it is server-level or host-level, and whether it is consumable or not by setting resource accumulation flags via `qmgr`. A consumable resource is tracked, or accumulated, in the server, queue or vnode `resources_assigned` attribute. The resource accumulation flags determine where the value of `resources_assigned.<resource name>` is incremented.

### 5.14.2.3.i Allowable Values for Resource Accumulation Flags

The value of `<resource flags>`, which is the resource accumulation flag for a resource can be one of the following:

**Table 5-9: Resource Accumulation Flags**

| Flag       | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (no flags) | Indicates a queue-level or server-level resource that is not consumable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>fh</i>  | <p>The amount is consumable at the host level for only the first vnode allocated to the job (vnode with first task.) Must be consumable or time-based. Cannot be used with Boolean or string resources. .</p> <p>This flag specifies that the resource is accumulated at the first vnode, meaning that the value of <code>resources_assigned.&lt;resource&gt;</code> is incremented only at the first vnode when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.</p>                                                                                                                                                                                                                                                                          |
| <i>h</i>   | <p>Indicates a host-level resource. Used alone, means that the resource is not consumable. Required for any resource that will be used inside a select statement. This flag selects hardware. This flag indicates that the resource must be requested inside of a select statement.</p> <p>Example: for a Boolean resource named "green":</p> <pre>Qmgr: create resource green type=boolean, flag=h</pre>                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>nh</i>  | <p>The amount is consumable at the host level, for all vnodes assigned to the job. Must be consumable or time-based. Cannot be used with Boolean or string resources.</p> <p>This flag specifies that the resource is accumulated at the vnode level, meaning that the value of <code>resources_assigned.&lt;resource&gt;</code> is incremented at relevant vnodes when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.</p> <p>This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.</p>                                                                                                                                                                                |
| <i>q</i>   | <p>The amount is consumable at the queue and server level. When a job is assigned one unit of a resource with this flag, the <code>resources_assigned.&lt;resource&gt;</code> attribute at the server and any queue is incremented by one. Must be consumable or time-based.</p> <p>This flag specifies that the resource is accumulated at the queue and server level, meaning that the value of <code>resources_assigned.&lt;resource&gt;</code> is incremented at each queue and at the server when a job is allocated this resource. When a reservation starts, allocated resources are added to the server's <code>resources_assigned</code> attribute.</p> <p>This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.</p> |



### 5.14.2.3.ii When to Use Accumulation Flags

The following table shows when to use accumulation flags.

**Table 5-10: When to Use Accumulation Flags**

| Resource Category      | Server                                                                | Queue                         | Host                                                |
|------------------------|-----------------------------------------------------------------------|-------------------------------|-----------------------------------------------------|
| Static, consumable     | flag = q                                                              | flag = q                      | flag = nh or fh                                     |
| Static, not consumable | flag = (none of h, n, q or f)                                         | flag = (none of h, n, q or f) | flag = h                                            |
| Dynamic                | server_dyn_res line in sched_config,<br>flag = (none of h, n, q or f) | (cannot be used)              | Tracked using an exechost_periodic hook<br>flag = h |

### 5.14.2.3.iii Example of Resource Accumulation Flags

When defining a static consumable host-level resource, such as a node-locked application license, you would use the "n" and "h" flags.

When defining a dynamic resource such as a floating license, you would use no flags.

### 5.14.2.3.iv Resource Accumulation Flag Restrictions and Caveats

Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be under-used. These resources are tracked automatically by the scheduler.

## 5.14.2.4 Specifying Resource Visibility

When you define a custom resource, you can specify whether unprivileged users have permission to view or request the resource, and whether users can `qalter` a request for that resource. This is done by setting a resource permission flag via `qmgr`.

### 5.14.2.4.i Allowable Values for Resource Permission Flags

The permission flag for a resource can be one of the following:

**Table 5-11: Resource Permission Flags**

| Flag      | Meaning                                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------------------|
| (no flag) | Users can view and request the resource, and <code>qalter</code> a resource request for this resource.                         |
| <i>i</i>  | "Invisible". Users cannot view or request the resource. Users cannot <code>qalter</code> a resource request for this resource. |
| <i>r</i>  | "Read only". Users can view the resource, but cannot request it or <code>qalter</code> a resource request for this resource.   |

#### 5.14.2.4.ii Effect of Resource Permission Flags

- PBS Operators and Managers can view and request a resource, and `qalter` a resource request for that resource, regardless of the `i` and `r` flags.
- Users, operators and managers cannot submit a job which requests a restricted resource. Any job requesting a restricted resource will be rejected. If a manager needs to run a job which has a restricted resource with a different value from the default value, the manager must submit the job without requesting the resource, then `qalter` the resource value.
- While users cannot request these resources, their jobs can inherit default resources from `resources_default.<resource name>` and `default_chunk.<resource name>`.

If a user tries to request a resource or modify a resource request which has a resource permission flag, they will get an error message from the command and the request will be rejected. For example, if they try to `qalter` a job's resource request, they will see an error message similar to the following:

```
"qalter: Cannot set attribute, read only or insufficient permission Resource_List.hps 173.mars"
```

#### 5.14.2.4.iii Resource Permission Flag Restrictions and Caveats

- You can specify only one of the `i` or `r` flags per resource. If both are specified, the resource is treated as if only the `i` flag were specified, and an error message is logged at the default log level and printed to standard error.
- Resources assigned from the `default_qsub_arguments` server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag whether that resource was requested by the user or came from `default_qsub_arguments`.
- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

`pbsnodes`

Users cannot view restricted host-level custom resources.

`pbs_rstat`

Users cannot view restricted reservation resources.

`pbs_rsub`

Users cannot request restricted custom resources for reservations.

`qalter`

Users cannot alter a restricted resource.

`qmgr`

Users cannot print or list a restricted resource.

`qselect`

Users cannot specify restricted resources via `-l Resource_List`.

`qsub`

Users cannot request a restricted resource.

`qstat`

Users cannot view a restricted resource.

### 5.14.2.5 Specifying Whether Resource is Cached at MoM

You can make it faster for execution hooks to read custom job resources. Execution hooks cannot read custom job resources via the event, only via the server. However, you can cache a copy of a custom job resource at the MoMs for faster local reading by execution hooks, by setting the *m* flag for the resource. The job resources that can be cached are found in the following job attributes:

```
exec_vnode
Resource_List
resources_used
```

To create a resource with the *m* flag set, include the flag. For example, to create two host-level consumable resources *r1* and *r2* of type *long* that will be cached at MoMs:

```
qmgr -c "create resource r1,r2 type=long,flag=mnh"
```

To unset this flag for *r1*:

```
qmgr -c "set resource r1 flag=nh"
```

You can combine this flag with any other resource flag. Job resources created in an *exechost\_startup* hook have the *m* flag set automatically.

#### 5.14.2.5.i Caveats for Caching Custom Job Resources

Large numbers of job resources that are cached at MoMs can slow things down. If you don't need execution hooks to be able to read a custom job resource often, don't cache the resource at the MoMs.

### 5.14.2.6 Defining Custom Resources via qmgr

You can use *qmgr* to create and delete custom resources, and to set their type and flags.

You must have PBS Manager privilege to operate on resources via *qmgr*.

#### 5.14.2.6.i Creating Custom Resources via qmgr

When you define or change a custom resource via *qmgr*, the changes take place immediately, and you do not have to restart the server, but you do have to restart scheduler(s).

To create a resource:

```
qmgr -c 'create resource <resource name>[,<resource name>] [type = <type>], [flag = <flags>]'
```

For example:

```
Qmgr: create resource foo type=long,flag=q
```

To create multiple resources of the same type and flag, separate each resource name with a comma:

```
qmgr -c "create resource r1,r2 type=long,flag=nh"
```

You can abbreviate "resource" to "r":

```
qmgr -c "create r foo type=long,flag=nh"
```

You cannot create a resource with the same name as an existing resource.

After you have defined your new custom resource, tell the scheduler how to use it. See [section 5.14.2.8, “Allowing Jobs to Use a Resource”, on page 261](#).

#### 5.14.2.6.ii Examples of Defining Custom Resources and Setting Flags via qmgr

To set the type for a resource:

```
set resource <resource name> type = <type>
```

For example:

```
qmgr -c "set resource foo type=string_array"
```

To set the flags for a resource:

```
set resource <resource name> flag=<flag(s)>
```

For example:

```
qmgr -c "set resource foo flag=nh"
```

To set the type and flags for a resource:

```
set resource <resource name> type=<type>, flag=<flag(s)>
```

For example:

```
qmgr -c "set resource foo type=long, flag=nhi"
```

You can set multiple resources by separating the names with commas. For example:

```
qmgr -c "set resource r1, r2 type=long"
```

You cannot set the *nh*, *fh*, or *q* flag for a resource of type string, string\_array, or Boolean.

You cannot set both the *n* and the *f* flags on one resource.

You cannot have the *n* or *f* flags without the *h* flag.

You cannot set both the *i* and *r* flags on one resource.

You cannot unset the type for a resource.

You cannot set the type for a resource that is requested by a current or history job or reservation, or set on a server, queue, or vnode.

You cannot set the flag(s) to *h*, *nh*, *fh*, or *q* for a resource that is currently requested by a current or history job or reservation.

You cannot unset the flag(s) for a resource that is currently requested by a current or history job or a reservation, or set on any server, queue, or vnode.

You cannot alter a built-in resource.

You can unset custom resource flags, but not their type.

### 5.14.2.6.iii Caveats for Defining Host-level Custom Resources via qmgr

If you plan on using a hook to set a job's `resources_used` value for a custom host-level resource, or you want to have a custom resource summed in a job's `resources_used` attribute and shown in the accounting log, you must create that resource using a hook.

### 5.14.2.6.iv Deleting Custom Resources

If you want to be able to delete a custom resource, make sure that the resource is not requested by any current or history jobs or current reservations. You can let those jobs finish, `qalter` them, or delete them. Delete and re-create any reservations that request the resource.

Before you delete a custom resource, you must remove all references to that resource, including where it is used in hooks or the scheduling formula. When you delete a resource that is set on the server, a queue, or a vnode, PBS unsets the resource for you.

You cannot delete a custom resource that is listed in the `restrict_res_to_release_on_suspend` server attribute. You must first remove the resource from the list:

```
Qmgr: set server restrict_res_to_release_on_suspend -= <resource name>
```

You cannot delete a built-in resource.

To remove a custom resource:

1. Remove all references to the resource
  - Remove it from the formula
  - Remove it from hooks
  - Let jobs requesting it finish, requeue and then `qalter` them while they are queued, or delete them
  - Delete and re-create any reservations that request the resource
2. Edit the `resources:` line in `<sched_priv directory>/sched_config` to remove the unwanted resource name:
  - If the resource is a server dynamic resource, remove the resource name from the `server_dyn_res:` line
3. For each MoM whose Version 2 configuration file contains references to the resource, use the `pbs_mom -s insert` command to update the Version 2 configuration file. See [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).
4. HUP each MoM; for Linux, see [“Restarting and Reinitializing MoM” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“Restarting MoMs” on page 159 in the PBS Professional Installation & Upgrade Guide](#).
5. Delete the resource using `qmgr`:

```
qmgr -c 'delete resource <resource name>'
```

For example:

```
qmgr -c "delete resource foo"
```

### 5.14.2.7 Defining Custom Resources via Hooks

You can use hooks to add new custom host-level resources, and set their values. See [“Adding Custom Host-level Resources” on page 69 in the PBS Professional Hooks Guide](#).

You must make the resource usable by the scheduler: see [section 5.14.2.8, “Allowing Jobs to Use a Resource”, on page 261](#).

To delete a custom resource created in a hook, use `qmgr`. See [section 5.14.2.6.iv, “Deleting Custom Resources”, on page 260](#).

### 5.14.2.8 Allowing Jobs to Use a Resource

After you define your resource, you need to make it usable by jobs:

1. Put the resource in the `"resources:"` line in `<sched_priv directory>/sched_config`. If the resource is a host-level boolean, you do not need to add it here.
2. If the resource is static, set the value via `qmgr`.
3. If the resource is a server-level dynamic resource, add it to the `server_dyn_res` line in the scheduler's configuration file
4. HUP the scheduler(s)

### 5.14.2.9 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 5.14.2.10 Example of Defining Each Type of Custom Resource

In this example, we add four custom resources: a static host-level resource, a static and a dynamic server-level resource, and a static queue-level resource.

1. The resource must be defined, with appropriate flags set:

```
Qmgr: create resource staticserverresource type=long, flag=q
Qmgr: create resource statichostresource type=long, flag=nh
Qmgr: create resource dynamicserverresource type=long
Qmgr: create resource staticqueueresource type=long, flag=q
```

2. The resource must be added to the scheduler's list of resources:

Add resource to "resources:" line in <sched\_priv directory>/sched\_config:

```
resources: "[...], staticserverresource, statichostresource, dynamicserverresource,
staticqueueresource"
```

Host-level Boolean resources do not need to be added to the "resources:" line.

3. HUP the scheduler:

```
kill -HUP <scheduler PID>
```

4. If the resource is static, use qmgr to set it at the host, queue or server level:

```
Qmgr: set node Host1 resources_available.statichostresource=1
Qmgr: set queue Queue1 resources_available.staticqueueresource=1
Qmgr: set server resources_available.staticserverresource=1
```

See [“qmgr” on page 152 of the PBS Professional Reference Guide](#).

5. If the resource is dynamic, add it to the "server\_dyn\_res" line in <sched\_priv directory>/sched\_config:

Linux:

```
server_dyn_res: "dynamicserverresource !path-to-
command"
```

Windows, no spaces in path:

```
server_dyn_res: 'dynamicserverresource !path-to-
command'
```

or:

```
server_dyn_res: "dynamicserverresource !path-to-
command"
```

Windows, spaces in path:

```
server_dyn_res: 'dynamicserverresource !"path-to-
command including spaces"'
```

- c. Make sure that the script meets the requirements in ["Requirements for Scripts that Update Dynamic Resources"](#)

### 5.14.3 Creating Server-level Custom Resources

You can have PBS track the availability of an externally-managed dynamic server-level resource, by running a script or program specified in the `server_dyn_res` line of `<sched_priv_directory>/sched_config`. PBS updates the value for `resources_available.<resource name>` at each scheduling cycle using the value returned by the script. This script is run at the host where the scheduler runs, once per scheduling cycle. Dynamic server-level resources are usually used for site-wide externally-managed floating application licenses.

The scheduler runs the query and waits for it to finish or time out. The default timeout for a server dynamic resource script is 30 seconds. You can specify a timeout for server dynamic resources in each scheduler's `server_dyn_res_alarm` attribute. If the script does not finish before the timeout, the scheduler uses a value of zero for the dynamic server resource. If you set the timeout to zero, the scheduler does not place a time limit on the script.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

#### 5.14.3.1 Creating Server Dynamic Resource Scripts

You create the script or program that PBS uses to query the external source. The external source can be a license manager or a command, as when you use the `df` command to find the amount of available disk space.

The format of a dynamic server-level resource query is a shell escape:

```
server_dyn_res: "<resource name> !<path to command>"
```

where

`<resource name>` is the name of the dynamic resource

`<path to command>` is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added. This usually means querying a license server.

Place the script on the server host. For example, it could be placed in `/usr/local/bin/serverdyn.pl`. Make sure the script meets the requirements in ["Requirements for Scripts that Update Dynamic Resources"](#).

##### 5.14.3.1.i Requirements for Scripts that Update Dynamic Resources

The script:

- Owned and executable by `PBS_DAEMON_SERVICE_USER`
- Has permissions of 0755
- Returns its output via `stdout`, and the output must be in a single line ending with a newline
- The scheduler has access to the script, and can run it
- If you have set up peer scheduling, make sure that the script is available to any scheduler that needs to run it

The directory containing the script:

- Owned by `PBS_DAEMON_SERVICE_USER`
- Accessible only by `PBS_DAEMON_SERVICE_USER` (must not give write permission to *group* or *others*)
- Has permissions 0550

##### 5.14.3.1.ii Caveats and Restrictions for Server Dynamic Resources

- Server dynamic resources are available only to the scheduler.
- Server dynamic resource values have no `resources_available.<resource name>` representation anywhere in PBS.
- A server dynamic resource shows up in the output of `qstat` only if a job has requested it.



### 5.14.3.1.iii Example of Configuring Dynamic Server-level Resource

For a site-wide externally-managed floating application license you will need two resources: one to represent the licenses themselves, and one to mark the vnodes on which the application can be run. The first is a server-level dynamic resource and the second is a host-level Boolean, set on the vnodes to send jobs requiring that license to those vnodes.

These are the steps for configuring a dynamic server-level resource for a site-wide externally-managed floating license. If this license could be used on all vnodes, the Boolean resource would not be necessary.

1. Define the resources, for example floatlicense and CanRun :  

```
Qmgr: create resource floatlicense type=long
Qmgr: create resource CanRun type=boolean, flag=h
```
2. Write a script, for example serverdyn.pl, that returns the available amount of the resource via stdout, and place it on the server host. For example, it could be placed in /usr/local/bin/serverdyn.pl
3. Make sure that the script and the directory containing meet the requirements in ["Requirements for Scripts that Update Dynamic Resources"](#).
4. Configure the scheduler to use the script by adding the resource and the path to the script in the server\_dyn\_res line of <sched\_priv directory>/sched\_config:  

```
server_dyn_res: "floatlicense !/usr/local/bin/serverdyn.pl"
```
5. Optional: give the scheduler a time limit for the script by setting its server\_dyn\_res\_alarm attribute:  

```
Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
```
6. Add the new dynamic resource to the resources: line in <sched\_priv directory>/sched\_config:  

```
resources: "ncpus, mem, arch, [...], floatlicense"
```
7. Restart the scheduler. See ["Restarting and Reinitializing Scheduler or Multisched"](#) on page 150 in the [PBS Professional Installation & Upgrade Guide](#).
8. Set the Boolean resource on the vnodes where the floating licenses can be run. Here we designate vnode1 and vnode2 as the vnodes that can run the application:  

```
Qmgr: active node vnode1,vnode2
Qmgr: set node resources_available.CanRun=True
```

To request this resource, the job's resource request would include:

```
-l floatlicense=<number of licenses or tokens required>
-l select=1:ncpus=N:CanRun=1
```

### 5.14.3.2 Static Server-level Resources

Static server-level resources are used for resources like floating licenses that PBS will manage. PBS keeps track of the number of available licenses instead of querying an external license manager.

### 5.14.3.2.i Example of Configuring Static Server-level Resource

These are the steps for configuring a static server-level resource:

1. Define the resource, for example sitelicense:  
`Qmgr: create resource sitelicense type=long, flag=q`
2. Use the `qmgr` command to set the value of the resource on the server:  
`Qmgr: set server resources_available.sitelicense=<number of licenses>`
3. Add the new resource to the `resources:` line in `<sched_priv directory>/sched_config`.  
`resources: "ncpus, mem, arch, [...], sitelicense"`
4. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

## 5.14.4 Configuring Host-level Custom Resources

Host-level custom resources can be static and consumable, or static and not consumable.

### 5.14.4.1 Dynamic Host-level Resources

To dynamically track how much scratch space is available on machines, use an `exechost_periodic` hook to keep the resource updated. See the *PBS Professional Hooks (Plugins) Guide*.

#### 5.14.4.1.i Example of Configuring Dynamic Host-level Resource

In this example, we configure a custom resource to track host-level scratch space. The resource is called *dynscratch*. These are the steps for configuring a dynamic host-level resource:

1. Define the resource, for example dynscratch:  
`Qmgr: create resource dynscratch type=size, flag=h`
2. Write an `exechost_periodic` hook that updates the value of the available scratch space on each execution host.
3. You may optionally specify any limits on that resource via `qmgr`, such as the maximum amount available, or the maximum that a single user can request.
4. Add the new resource to the `resources:` line in `<sched_priv directory>/sched_config`:  
`resources: "ncpus, mem, arch, [...], dynscratch"`
5. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request this resource, the resource request would include

```
-l select=1:ncpus=N:dynscratch=10MB
```

### 5.14.4.2 Static Host-level Resources

Use static host-level resources for things that are managed by PBS and available at the host level, such as GPUs.

### 5.14.4.2.i Example of Configuring Static Host-level Resource

In this example, we configure a consumable host-level resource to track GPUs. These are the steps for configuring a static host-level resource:

1. Define the resource, for example `ngpus`:  
`Qmgr: create resource ngpus type=long, flag=nh`
2. Use the `qmgr` command to set the value of the resource on the host:  
`Qmgr: set node Host1 ngpus=<number of GPUs>`
3. Add the new resource to the `resources` line in `<sched_priv directory>/sched_config`.  
`resources: "ncpus, mem, arch, [...], ngpus"`
4. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).
5. If the GPU host is a multi-vnode machine, you may want to define which GPUs belong in which vnodes. In this case, do the following:
  - a. Create a vnode definition file. See [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).
  - b. Restart the MoM. For Linux, see [“Restarting and Reinitializing MoM” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“Restarting MoMs” on page 159 in the PBS Professional Installation & Upgrade Guide](#).

See [section 5.14.6.4.iv, “Example of Per-host Node-locked Licensing”, on page 275](#), [section 5.14.6.4.v, “Example of Per-use Node-locked Licensing”, on page 276](#), and [section 5.14.6.4.vi, “Example of Per-CPU Node-locked Licensing”, on page 277](#). These sections give examples of configuring each kind of node-locked license.

### 5.14.4.3 Shared Host-level Resources

Two or more vnodes can share the use of a resource. The resource is managed at one vnode, but available for use at other vnodes. The MoM manages the sharing of the resource, allocating only the available amount to jobs. For example, if you want jobs at two separate vnodes to be able to use the same 4GB of memory, you can make the memory be a shared resource. This way, if a job at one vnode uses all 4GB, no other jobs can use it, but if one job at one vnode uses 2GB, other jobs at either vnode can use up to 2GB.

#### 5.14.4.3.i Shared Resource Glossary

##### **Borrowing vnode**

The vnode where a shared vnode resource is available, but not managed.

##### **Indirect resource**

A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

##### **Managing vnode**

The vnode where a shared vnode resource is defined, and which manages the resource.

##### **Shared resource**

A vnode resource defined at managed at one vnode, but available for use at others.

#### 5.14.4.3.ii Configuring Shared Host-level Resources

The resource to be shared is defined as usual at one vnode. This is the managing vnode for that resource. For example, to make memory be managed at `Vnode1`:

```
Qmgr: set node Vnode1 mem = 4gb
```

At vnodes which will use the same resource, the resource is defined to be indirect. For example, to make memory be shared and borrowed at *Vnode2*:

```
Qmgr: set node Vnode2 mem = @Vnode1
```

### 5.14.4.3.iii Configuring Shared Static Resources

1. If the resource to be shared is a custom resource, you must define the resource before setting its value:

```
Qmgr: create resource <resource name> type=<resource type> [flag = <flags>]
```

2. Set the resource on the managing vnode:

To set a static value via qmgr:

```
Qmgr: s n managing_vnode resources_available.<resource name> =<value>
```

To set a static value, in a Version 2 configuration file:

```
managing_vnode:<resource name>=<value>
```

3. Next, set the resource on the borrowing vnode:

To set a shared resource on a borrowing vnode via qmgr:

```
Qmgr: s n borrowing_vnode resources_available.<resource name>=@managing_vnode
```

To set a shared resource in a Version 2 configuration file:

```
borrowing_vnode:<resource name>=@managing_vnode
```

4. HUP the MoMs involved; for Linux, see [“Restarting and Reinitializing MoM” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“Restarting MoMs” on page 159 in the PBS Professional Installation & Upgrade Guide](#).

Example 5-10: To make a static host-level license `dyna-license` on `hostA` be managed by the parent vnode at `hostA` and indirect at vnodes `hostA0` and `hostA1`:

```
Qmgr: set node hostA resources_available.dyna-license=4
```

```
Qmgr: set node hostA0 resources_available.dyna-license=@hostA
```

```
Qmgr: set node hostA1 resources_available.dyna-license=@hostA
```

### 5.14.4.3.iv Configuring Shared Dynamic Resources

1. If the resource to be shared is a custom resource, you must define the resource before setting its value:

```
Qmgr: create resource <resource name> type=<resource type> [flag = <flags>]
```

2. Set the resource on the managing vnode via an `exechost_periodic` hook

3. Next, set the resource on the borrowing vnode:

To set a shared resource on a borrowing vnode via qmgr:

```
Qmgr: s n borrowing_vnode resources_available.<resource name>=@managing_vnode
```

To set a shared resource in a Version 2 configuration file:

```
borrowing_vnode:<resource name>=@managing_vnode
```

4. HUP the MoMs involved; for Linux, see [“Restarting and Reinitializing MoM” on page 151 in the PBS Professional Installation & Upgrade Guide](#), and for Windows, see [“Restarting MoMs” on page 159 in the PBS Professional Installation & Upgrade Guide](#).

#### 5.14.4.3.v Restrictions on Shared Host-level Resources

- If your vnodes represent physical units such as blades, sharing resources like `ncpus` across vnodes may not make sense.
- If you want to make a resource shared across vnodes, remember that you do not want to schedule jobs on the parent vnode. To avoid this, the following resources should not be explicitly set on the parent vnode:

```
ncpus
mem
vmem
```

#### 5.14.4.3.vi Defining Shared and Non-shared Resources for Multi-vnode Machines

On a multi-vnode machine, you can manage the resources at each vnode. For dynamic host-level resources, the resource is shared across all the vnodes on the machine, and MoM manages the sharing. For static host-level resources, you can either define the resource as shared or not. Shared resources are usually set on the parent vnode and then made indirect at any child vnodes on which you want the resource available. For resources that are not shared, you can set the value at each vnode.

Example 5-11: To set the resource `string_res` to *round* on the parent vnode of `host03` and make it indirect at `host03[0]` and `host03[1]`:

```
Qmgr: set node host03 resources_available.string_res=round
Qmgr: s n host03[0] resources_available.string_res=@host03
Qmgr: s n host03[1] resources_available.string_res=@host03
pbsnodes -va
host03
...
string_res=round
...
host03[0]
...
string_res=@host03
...
host03[1]
...
string_res=@host03
...
```

If you had set the resource `string_res` individually on `host03[0]` and `host03[1]`:

```
Qmgr: s n host03[0] resources_available.string_res=round
Qmgr: s n host03[1] resources_available.string_res=square
pbsnodes -va
host03
...
 <-----string_res not set on parent vnode
...
host03[0]
...
 string_res=round
...
host03[1]
...
 string_res=square
...
```

#### 5.14.4.3.vii Shared Resource Restrictions for Multi-vnode Machines

- On the parent vnode, all values for `resources_available.<resource name>` should be *zero (0)*, unless the resource is being shared among other vnodes via indirection.

### 5.14.5 Using Scratch Space

#### 5.14.5.1 Dynamic Server-level (Shared) Scratch Space

If you have scratch space set up so that it's available to all execution hosts, you can use a server-level custom dynamic resource to track it. The following are the steps for configuring a dynamic server-level resource called `globalscratch` to track globally available scratch space:

- Define the resource:
 

```
Qmgr: create resource globalscratch type=long
```
- Write a script, for example `serverdynscratch.pl`, that returns the available amount of the resource via `stdout`, and place it on the server host. For example, it could be placed in `/usr/local/bin/serverdynscratch.pl`
- Configure the scheduler to use the script by adding the resource and the path to the script in the `server_dyn_res` line of `<sched_priv directory>/sched_config`:
 

```
server_dyn_res: "globalscratch !/usr/local/bin/serverdynscratch.pl"
```
- Optional: give the scheduler a time limit for the script by setting its `server_dyn_res_alarm` attribute:
 

```
Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
```
- Add the new dynamic resource to the `resources:` line in `<sched_priv directory>/sched_config`:
 

```
resources: "ncpus, mem, arch, [...], globalscratch"
```
- Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request this resource, the job's resource request would include:

```
-l globalscratch=<space required>
```

### 5.14.5.2 Dynamic Host-level Scratch Space

Say you have jobs that require a large amount of scratch disk space during their execution. To ensure that sufficient space is available during job startup, create a custom dynamic resource so that jobs can request scratch space. To create this resource, take the steps outlined in [section 5.14.4.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 265](#).

### 5.14.5.3 Static Server-level Scratch Space

If you want to prevent jobs from stepping on each others' scratch space, you can define additional vnodes that are used only to allocate scratch devices, with one vnode per scratch device. Set the `sharing` attribute on each scratch vnode to `force_excl`, so that only one job can request each scratch device. To set the `sharing` attribute, follow the rules in [section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 50](#). For example, the scratch devices are `/scratch1`, `/scratch2`, `/scratch3`, etc. On each scratch device, set resources as follows:

```
resources_available.ncpus = 0
resources_available.mem = 0
resources_available.scratch = 1
sharing = force_excl
```

Jobs then request one additional chunk to represent the scratch device, for example:

```
-l 16:ncpus=1+1:scratch=1
```

If a job needs to request a specific scratch device, for example `/scratch2`, that can be done by additionally asking for the `scratch` resource:

```
:scratch=1
```

### 5.14.5.4 Static Host-level Scratch Space

If the scratch areas are not mounted on all execution hosts, you can specify which scratch areas are shared among which subsets of vnodes using indirect resources. See [section 5.14.4.3, “Shared Host-level Resources”, on page 266](#).

### 5.14.5.5 Caveats for Scratch Space and Jobs

When more than one job uses scratch space, or when a job is suspended, scratch space usage may not be handled correctly. See [section 5.9.5, “Dynamic Resource Allocation Caveats”, on page 247](#) and [section 5.9.6, “Period When Resource is Used by Job”, on page 247](#).

## 5.14.6 Supplying Application Licenses

### 5.14.6.1 Types of Licenses

Application licenses may be managed by PBS or by an external license manager. Application licenses may be floating or node-locked, and they may be *per-host*, where any number of instances can be running on that host, *per-CPU*, where one license allows one CPU to be used for that application, or *per-run*, where one license allows one instance of the application to be running. Each kind of license needs a different form of custom resource.

#### 5.14.6.1.i Externally-managed Licenses

Whenever an application license is managed by an external license manager, you must create a custom dynamic resource for it. This is because PBS has no control over whether these licenses are checked out, and must query the external license manager for the availability of those licenses. PBS does this by executing the script or program that you specify in the dynamic resource. This script returns the amount via `stdout`, in a single line ending with a newline.



### 5.14.6.1.ii Preventing Oversubscription of Externally-managed Licenses

Some applications delay the actual license checkout until some time after the application begins execution. Licenses could be oversubscribed when the scheduler queries for available licenses, and gets a result including licenses that essentially belong to a job that is already running but has not yet checked them out. To prevent this, you can create a consumable custom static integer resource, assign it the total number of licenses, and make each job that requests licenses request this resource as well. You can use a hook to accomplish this. Alternatively, if you know the maximum number of jobs that can run using these licenses, you can create a consumable custom static integer resource to track the number of jobs using licenses, and make each job request this resource.

If licenses are also checked out by applications outside of the control of PBS, this technique will not work.

### 5.14.6.1.iii PBS-managed Licenses

When an application license is managed by PBS, you can create a custom static resource for it. You set the total number of licenses using `qmgr`, and PBS will internally keep track of the number of licenses available.

Use static host-level resources for node-locked application licenses managed by PBS, where PBS is in full control of the licenses. These resources are *static* because PBS tracks them internally, and *host-level* because they are tracked at the host.

## 5.14.6.2 License Units and Features

Different licenses use different license units to track whether an application is allowed to run. Some licenses track the number of CPUs an application is allowed to run on. Some licenses use tokens, requiring that a certain number of tokens be available in order to run. Some licenses require a certain number of features to run the application.

When using units, after you have defined the license resource called `license_name` to the server, be sure to set `resources_available.license_name` to the correct number of units.

Before starting you should have answers to the following questions:

- How many units of a feature does the application require?
- How many features are required to execute the application?
- How do I query the license manager to obtain the available licenses of particular features?

With these questions answered you can begin configuring PBS Professional to query the license manager servers for the availability of application licenses. Think of a license manager feature as a resource. Therefore, you should associate a resource with each feature.

## 5.14.6.3 Server-level (Floating) Licenses

### 5.14.6.3.i Example of Floating, Externally-managed License

Here is an example of setting up floating licenses that are managed by an external license server.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have one licensed application which uses floating licenses from an external license manager. Furthermore we want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses, the hosts on which the licenses should be used, and a description of the type of license used by the application.

| Application | Licenses | Hosts | DESCRIPTION                                   |
|-------------|----------|-------|-----------------------------------------------|
| AppF        | 4        | 3-6   | Uses licenses from an externally managed pool |

For the floating licenses, we will use three resources. One is a dynamic server resource for the licenses themselves. One is a server-level integer to prevent oversubscription. The last is a Boolean resource used to indicate that the floating license can be used on a given host.

## Server Configuration

1. Define the new resources. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>,flag=<flags>
```

## Host Configuration

2. Set the Boolean resource on the hosts where the floating licenses can be used.

```
Qmgr: active node host3,host4,host5,host6
```

```
Qmgr: set node resources_available.runsAppF = True
```

## Scheduler Configuration

3. Edit the scheduler configuration file:

```
cd $<sched_priv directory>/
```

```
[edit] sched_config
```

4. Append the new resource names to the `resources:` line:

```
resources: "ncpus, mem, arch, host, [...], AppF, AppFcount, runsAppF"
```

5. Edit the `server_dyn_res:` line:

```
server_dyn_res: "AppF !/local/flex_AppF"
```

6. Optional: give the scheduler a time limit for the script by setting its `server_dyn_res_alarm` attribute:

```
Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
```

7. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

You can write a hook that examines the number of AppF licenses requested by each job, and assigns that many AppFcount to the job, or you can ask your users to request AppFcount.

To request a floating license for AppF and a host on which AppF can run:

```
qsub -l AppF=1 -l AppFcount=1
-l select=runsAppF=True
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```
host1
host2
host3
 resources_available.runsAppF = True
host4
 resources_available.runsAppF = True
host5
 resources_available.runsAppF = True
host6
 resources_available.runsAppF = True
```

### 5.14.6.3.ii Example of Floating, Externally-managed License with Features

This is an example of a floating license, managed by an external license manager, where the application requires a certain number of features to run. Floating licenses are treated as server-level dynamic resources. The license server is queried by an administrator-created script. This script returns the value via `stdout` in a single line ending with a newline.

The license script runs on the server host once per scheduling cycle and queries the number of available licenses/tokens for each configured application.

When submitting a job, the user's script, in addition to requesting CPUs, memory, etc., also requests licenses.

When the scheduler looks at all the enqueued jobs, it evaluates the license request alongside the request for physical resources, and if all the resource requirements can be met the job is run. If the job's token requirements cannot be met, then it remains queued.

PBS doesn't actually check out the licenses; the application being run inside the job's session does that. Note that a small number of applications request varying amounts of tokens during a job run.

Our example needs four features to run an application, so we need four custom resources.

1. Write four scripts, one to query the license server for each of your four features. Complexity of the script is entirely site-specific due to the nature of how applications are licensed.
2. Define four non-consumable server-level features. These features are defined with no flags:

```
Qmgr: create resource feature1 type=long
Qmgr: create resource feature3 type=long
Qmgr: create resource feature6 type=long
Qmgr: create resource feature8 type=long
```

3. Add the feature resources to the `resources:` line in `<sched_priv directory>/sched_config`:  
`resources: "ncpus, mem, arch, [...], feature1, feature3, feature6, feature8"`

4. Add each feature's script path to the `server_dyn_res:` line in `PBS_HOME/server_priv/config`:

```
server_dyn_res: "feature1 !/path/to/script [args]"
server_dyn_res: "feature3 !/path/to/script [args]"
server_dyn_res: "feature6 !/path/to/script [args]"
server_dyn_res: "feature8 !/path/to/script [args]"
```

5. Optional: give the scheduler a time limit for the scripts by setting its `server_dyn_res_alarm` attribute:

```
Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
```

6. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

### 5.14.6.3.iii Example of Floating License Managed by PBS

Here is an example of configuring custom resources for a floating license that PBS manages. For this you need a server-level static resource to keep track of the number of available licenses. If the application can run only on certain hosts, then you will need a host-level Boolean resource to direct jobs running the application to the correct hosts.

In this example, we have six hosts numbered 1-6, and the application can run on hosts 3, 4, 5 and 6. The resource that will track the licenses is called *AppM*. The Boolean resource is called *RunsAppM*.

Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>, flag=<flags>
```

Example:

```
Qmgr: create resource AppM type=long, flag=q
Qmgr: create resource runsAppM type=boolean, flag=h
```

2. Set a value for AppM at the server. Here, we're allowing 8 copies of the application to run at once:

```
Qmgr: set server resources_available.AppM=8
```

## Host Configuration

3. Set the value of `runsAppM` on the hosts. Each `qmgr` directive is typed on a single line:

```
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.runsAppM = True
```

## Scheduler Configuration

4. Edit the scheduler configuration file:

```
cd $<sched_priv directory>/
[edit] sched_config
```

5. Append the new resource name to the `resources:` line. Note that it is not necessary to add a host-level Boolean resource to this line.

```
resources: "ncpus, mem, arch, host, [...], AppM, runsAppM"
```

6. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request both the application and a host that can run `AppM`:

```
qsub -l AppM=1
-l select=1:runsAppM=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of *False*, you do not need to explicitly set them to *False* on the other hosts. Unset Boolean resources will not be printed.

```
host1
host2
host3
 resources_available.runsAppM = True
host4
 resources_available.runsAppM = True
host5
 resources_available.runsAppM = True
host5
 resources_available.runsAppM = True
```

### 5.14.6.4 Host-level (Node-locked) Licenses

#### 5.14.6.4.i Per-host Node-locked Licenses

If you are configuring a custom resource for a per-host node-locked license, where the number of jobs using the license does not matter, use a host-level Boolean resource on the appropriate host. This resource is set to *True*. When users request the license, they can use the following requests:

For a two-CPU job on a single vnode:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

Users can also use `"license=True"`, but this way they do not have to change their scripts.

#### 5.14.6.4.ii Per-CPU Node-locked Licenses

If you are configuring a custom resource for a per-CPU node-locked license, use a host-level consumable resource on the appropriate vnode. This resource is set to the maximum number of CPUs you want used on that vnode. Then when users request the license, they will use the following request:

For a two-CPU, two-license job:

```
-l select=1:ncpus=2:license=2
```

#### 5.14.6.4.iii Per-use Node-locked License

If you are configuring a custom resource for a per-use node-locked license, use a host-level consumable resource on the appropriate host. This resource is set to the maximum number of instances of the application allowed on that host. Then when users request the license, they will use:

For a two-CPU job on a single host:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job where each chunk needs two CPUs:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

#### 5.14.6.4.iv Example of Per-host Node-locked Licensing

Here is an example of setting up node-locked licenses where one license is required per host, regardless of the number of jobs on that host.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-host node-locked licenses. We want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

| Application | Licenses | Hosts | DESCRIPTION                            |
|-------------|----------|-------|----------------------------------------|
| AppA        | 1        | 1-4   | uses a node-locked application license |

For the per-host node-locked license, we will use a Boolean host-level resource called `resources_available.runsAppA`. This will be set to *True* on any hosts that should have the license, and will default to *False* on all others. The resource is not consumable so that more than one job can request the license at a time.

##### Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):  

```
create resource <resource name> type=<type>,flag=<flag>
```

Example:

```
Qmgr: create resource runsAppA type=boolean, flag=h
Qmgr: create resource AppA type=long, flag=h
```

##### Host Configuration

2. Set the value of `runsAppA` on the hosts. Each `qmgr` directive is typed on a single line:

```
Qmgr: active node host1,host2,host3,host4
Qmgr: set node resources_available.runsAppA = True
```

## Scheduler Configuration

3. Edit the scheduler configuration file.

```
cd $<sched_priv directory>/
[edit] sched_config
```

4. Append the new resource name to the "resources:" line. Note that it is not necessary to add the host-level Boolean resource to this line.

```
resources: "ncpus, mem, arch, [...], AppA, runsAppA"
```

5. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request a host with a per-host node-locked license for AppA:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of *False*, you do not need to explicitly set them to *False* on the other hosts. Unset Boolean resources will not be printed.

```
host1
 resources_available.runsAppA = True
host2
 resources_available.runsAppA = True
host3
 resources_available.runsAppA = True
host4
 resources_available.runsAppA = True
host5
host6
```

#### 5.14.6.4.v Example of Per-use Node-locked Licensing

Here is an example of setting up per-use node-locked licenses. Here, while a job is using one of the licenses, it is not available to any other job.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-use node-locked licenses. We want to limit use of the application only to specific hosts. The licensed hosts can run two instances each of the application. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

| Application | Licenses | Hosts | DESCRIPTION                            |
|-------------|----------|-------|----------------------------------------|
| AppB        | 2        | 1-2   | Uses a node-locked application license |

For the node-locked license, we will use one static host-level resource called `resources_available.AppB`. This will be set to 2 on any hosts that should have the license, and to 0 on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 2 licenses, only two jobs can use those licenses on that host at a time.

## Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>,flag=<flags>
```

Example:

```
Qmgr: create resource AppB type=long, flag=nh
```

Host Configuration

2. Set the value of AppB on the hosts to the maximum number of instances allowed. Each qmgr directive is typed on a single line:

```
Qmgr: active node host1,host2
Qmgr: set node resources_available.AppB = 2
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.AppB = 0
```

Scheduler Configuration

3. Edit the scheduler configuration file.

```
cd $<sched_priv directory>/
[edit] sched_config
```

4. Append the new resource name to the resources: line:

```
resources: "ncpus, mem, arch, host, [...], AppB"
```

5. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request a host with a node-locked license for AppB, where you'll run one instance of AppB on two CPUs:

```
qsub -l select=1:ncpus=2:AppB=1
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```
host1
 resources_available.AppB = 2
host2
 resources_available.AppB = 2
host3
 resources_available.AppB = 0
host4
 resources_available.AppB = 0
host5
 resources_available.AppB = 0
host6
 resources_available.AppB = 0
```

#### 5.14.6.4.vi Example of Per-CPU Node-locked Licensing

Here is an example of setting up per-CPU node-locked licenses. Each license is for one CPU, so a job that runs this application and needs two CPUs must request two licenses. While that job is using those two licenses, they are unavailable to other jobs.



For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-CPU node-locked licenses. We want to limit use of the application to specific hosts only. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

| Application | Licenses | Hosts | DESCRIPTION                            |
|-------------|----------|-------|----------------------------------------|
| AppC        | 4        | 3-4   | uses a node-locked application license |

For the node-locked license, we will use one static host-level resource called `resources_available.AppC`. We will provide a license for each CPU on hosts 3 and 4, so this will be set to `4` on any hosts that should have the license, and to `0` on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 4 licenses, only four CPUs can be used for that application at a time.

#### Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>, flag=<flags>
```

Example:

```
Qmgr: create resource AppC type=long, flag=nh
```

#### Host Configuration

2. Set the value of AppC on the hosts. Each `qmgr` directive is typed on a single line:

```
Qmgr: active node host3,host4
Qmgr: set node resources_available.AppC = 4
Qmgr: active node host1,host2,host5,host6
Qmgr: set node resources_available.AppC = 0
```

#### Scheduler Configuration

3. Edit the scheduler configuration file:

```
cd $<sched_priv directory>/
[edit] sched_config
```

4. Append the new resource name to the `resources:` line:

```
resources: "ncpus, mem, arch, host, [...], AppC"
```

5. Restart the scheduler. See [“Restarting and Reinitializing Scheduler or Multisched” on page 150 in the PBS Professional Installation & Upgrade Guide](#).

To request a host with a node-locked license for AppC, where you'll run a job using two CPUs:

```
qsub -l select=1:ncpus=2:AppC=2
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```
host1
 resources_available.AppC = 0
host2
 resources_available.AppC = 0
host3
 resources_available.AppC = 4
host4
 resources_available.AppC = 4
host5
 resources_available.AppC = 0
host6
 resources_available.AppC = 0
```

## 5.14.7 Using GPUs

You can configure PBS to manage GPU resources. You only need to use one method. You can use any of the following methods, but we recommend using the cgroups hook:

- [Managing GPUs via Cgroups](#) does the configuration for you, takes advantage of topology, and optionally may be used to deny access to GPU devices not assigned to a job. You can use this method to restrict each job to the GPU(s) assigned to it. We recommend using this method; job submission is much easier, among other advantages.
- [Basic GPU Scheduling](#) works well if you have single-GPU vnodes. The basic method will meet the needs of most job submitters; it allows a job to request the number of GPUs it needs, as long as the job requests exclusive use of each node containing the GPUs.
- [Advanced GPU Scheduling](#) allows jobs to request specific GPUs. The advanced method provides some flexibility for multi-job or multi-GPU vnodes, but does not isolate GPUs. PBS Professional allocates GPUs for jobs, but does not perform the actual binding. The application or the CUDA library binds the application to one or more GPUs.

You cannot combine this with the cgroups hook unless you set `ngpus_ext_managed` in the cgroups hook configuration file; see [section 5.14.7.2, “Managing GPUs Manually While Using Cgroups Hook”, on page 279](#) and [section 6.5.5.6, “Not Using Cgroups to Manage GPUs”, on page 349](#).

### 5.14.7.1 Managing GPUs Via Cgroups Hook

We describe how to manage your GPUs via cgroups in [section 6.5.5.1, “Managing GPUs via Cgroups”, on page 345](#).

### 5.14.7.2 Managing GPUs Manually While Using Cgroups Hook

For any node or nodes, you can use basic or advanced GPU scheduling while also using the cgroups hook. By default, the cgroups hook manages GPUs for you: it sets `resources_available.ngpus` on the node, it won't oversubscribe a GPU, and it assigns specific GPUs to jobs running on that node or nodes. However, if you want to oversubscribe the GPUs on a node, you can tell the cgroups hook not to manage the GPUs on that node; see [section 6.5.5.6, “Not Using Cgroups to Manage GPUs”, on page 349](#). Then you can use basic or advanced GPU scheduling on that node.

### 5.14.7.3 Basic GPU Scheduling

Basic scheduling consists of prioritizing jobs based on partition or site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking use of GPUs in accounting logs.

Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node.

This method allows jobs to request unspecified GPUs. Jobs should request exclusive use of the node to prevent other jobs being scheduled on their GPUs.

### 5.14.7.3.i Configuring PBS for Basic GPU Scheduling

You configure a single custom consumable resource to represent all GPU devices on an execution host. Create a host-level consumable custom resource to represent GPUs. We recommend that the custom GPU resource is named *ngpus*. Set the value for this resource at each vnode to the number of GPUs on the vnode.

The *ngpus* resource is used exactly the way you use the *ncpus* resource.

### 5.14.7.3.ii Example of Configuring PBS for Basic GPU Scheduling

In this example, there are two execution hosts, HostA and HostB, and each execution host has 4 GPU devices.

1. Create the *ngpus* resource:  
`Qmgr: create resource ngpus type=long, flag=nh`
2. Stop the server and scheduler. On the server's host, type:  
`systemctl stop pbs`  
or  
`/etc/init.d/pbs stop`
3. Edit `<sched_priv directory>/sched_config` to add *ngpus* to the list of scheduling resources:  
`resources: "ncpus, mem, arch, host, vnode, ngpus"`
4. Start the server and scheduler. On the server's host, type:  
`systemctl start pbs`  
or  
`/etc/init.d/pbs start`
5. Add the number of GPU devices available to each execution host in the cluster via *qmgr*:  
`Qmgr: set node HostA resources_available.ngpus=4`  
`Qmgr: set node HostB resources_available.ngpus=4`

## 5.14.7.4 Advanced GPU Scheduling

Advanced scheduling allows a job to separately allocate (request and/or identify) each individual GPU on a node.

In this case, both PBS and the applications themselves must support individually allocating the GPUs on a node. Advanced scheduling requires defining a child vnode for each GPU.

This capability is useful for sharing a single multi-GPU node among multiple jobs, where each job requires exclusive use of its GPUs.

### 5.14.7.4.i Configuring PBS for Advanced GPU Scheduling

You configure each GPU device in its own vnode, and each GPU vnode has a resource to contain the device number of its GPU.

Create and set two custom resources:

- Create a host-level consumable resource to represent the GPUs on a vnode. We recommend that this resource is called *ngpus*.

Set `ngpus` on each node to the number of GPUs on that node.

- Create a host-level non-consumable resource containing the GPU device number, which serves to tie the individual GPU to the vnode. We recommend that this resource is called `gpu_id`.

Set `gpu_id` for each GPU to the device number of that GPU.

#### 5.14.7.4.ii Example of Configuring PBS for Advanced GPU Scheduling

In this example, there is one execution host, HostA, that has two child vnodes, HostA[0] and HostA[1], as well as the parent vnode. HostA has 4 CPUs, 2 GPUs, and 16 GB of memory.

1. Create the new custom resources:

```
Qmgr: create resource ngpus type=long, flag=nh
Qmgr: create resource gpu_id type=string, flag=h
```

2. Stop the server and scheduler. On the server's host, type:

```
systemctl stop pbs
or
/etc/init.d/pbs stop
```

3. Edit `<sched_priv directory>/sched_config` to add `ngpus` and `gpu_id` to the list of scheduling resources:

```
resources: "ncpus, mem, arch, host, vnode, ngpus, gpu_id"
```

4. Start the server and scheduler. On the server's host, type:

```
systemctl start pbs
or
/etc/init.d/pbs start
```

5. Create a vnode configuration file for each execution host where GPUs are present. See [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#). The script for HostA is named `hostA_vnodes`, and is shown here:

```
$configversion 2
hostA: resources_available.ncpus = 0
hostA: resources_available.mem = 0
hostA[0]: resources_available.ncpus = 2
hostA[0] : resources_available.mem = 8gb
hostA[0] : resources_available.ngpus = 1
hostA[0] : resources_available.gpu_id = gpu0
hostA[0] : sharing = default_excl
hostA[1] : resources_available.ncpus = 2
hostA[1] : resources_available.mem = 8gb
hostA[1] : resources_available.ngpus = 1
hostA[1] : resources_available.gpu_id = gpu1
hostA[1]: sharing = default_excl
```

6. Create a Version 2 configuration file for each host with GPUs. For example:

```
PBS_EXEC/sbin/pbs_mom -s insert HostA_vnodes HostA_vnodes
```

7. Signal each MoM to re-read its configuration files:

```
kill -HUP <pbs_mom PID>
```

---

### 5.14.8 Using FPGAs

You can configure a custom resource that allows PBS to track the usage of FPGAs. The FPGAs are detected outside of PBS at boot time. There are two basic methods for automatic configuration of the FPGA resource:

- Create a static host-level resource called *nfgas*. Create a boot-up script in *init.d* that detects the presence of the FPGAs, and sets the value of the *nfgas* resource.
- Create a dynamic host-level resource called *nfgas*. This resource calls a script to detect the presence of FPGAs

We recommend the static resource, because FPGAs are static.

### 5.14.9 Defining Host-level Resource for Applications

You may need to tag your vnodes with the software that can run on them. You cannot use the built-in **software** resource for this; it is a server-level resource and cannot be set per host. You can define a custom resource named, for example, "node\_software". It should be a **string\_array**, since a host may be able to run more than one application. You can use **qmgr** to create your resource:

```
Qmgr: create resource node_software type=string_array, flag=h
```

You can use your new custom resource to route jobs: see [section 4.9.39, "Routing Jobs", on page 205](#).

### 5.14.10 Custom Resource Caveats

- Because some custom resources are external to PBS, they are not completely under the control of PBS. Therefore it is possible for PBS to query and find a resource available, schedule a job to run and use that resource, only to have an outside entity take that resource before the job is able to use it. For example, say you had an external resource of "scratch space" and your query script simply checked to see how much disk space was free. It would be possible for a job to be started on a host with the requested space, but for another application to use the free space before the job did.
- If a resource is not put in the scheduler's **resources:** line, when jobs request the resource, that request will be ignored. If the resource is ignored, it cannot be used to accept or reject jobs at submission time. For example, if you create a string resource **String1** on the server, and set it to *foo*, a job requesting "**-l String1=bar**" will be accepted. The only exception is host-level Boolean resources, which are considered when scheduling, whether or not they are in the scheduler's **resources:** line.
- Do not create resources with the same names or prefixes that PBS uses when you create custom resources for specific systems.

## 5.15 Managing Resource Usage

You can manage resource usage from different directions:

- You can manage resource usage by users, groups, and projects, and the number of jobs, at the server and queue level. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).
  - You can manage the total amount of each resource that is used by projects, users or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.
  - You can manage the number of jobs being run by projects, users or groups, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.
- You can specify how much of each resource any job is allowed to use, at the server and queue level. See [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#) and [section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251](#).
- You can set default limits for usage for each resource, at the server or queue level, so that jobs that do not request a given resource inherit that default, and are limited to the inherited amount. For example, you can specify that any job entering queue QueueA not specifying mem is limited to using 4MB of memory. See [section 5.9.3, “Specifying Job Default Resources”, on page 241](#).
- You can set limits on the number of jobs that can be in the queued state at the server and/or queue level. You can apply these limits to users, groups, projects, or everyone. This allows users to submit as many jobs as they want, while allowing the scheduler to consider only the jobs in the execution queues, thereby speeding up the scheduling cycle. See [section 5.15.3, “Limiting the Number of Jobs in Queues”, on page 305](#).

### 5.15.1 Managing Resource Usage By Users, Groups, and Projects, at Server & Queues

You can set separate limits for resource usage by individual users, individual groups, individual projects, generic users, generic groups, generic projects, and the total used overall, for queued jobs, running jobs, and queued and running jobs. You can limit the amount of resources used, and the number of queued jobs, the number of running jobs, and the number of queued and running jobs. These limits can be defined separately for each queue and for the server. You define the limits by setting server and queue limit attributes. For information about projects, see [section 10.4, “Grouping Jobs By Project”, on page 457](#).

There are **two incompatible sets of server and queue limit attributes** used in limiting resource usage. The first set existed in PBS Professional before Version 10.1, and we call them the **old limit attributes**. The old limit attributes are discussed in [section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 298](#). The set introduced in Version 10.1 is called simply the limit attributes, and they are discussed here.

You can use either the limit attributes or the old limit attributes for the server and queues, but not both. See [section 5.15.1.13.v, “Do Not Mix Old And New Limits”, on page 297](#).

The server and queues each have per-job limit attributes which operate independently of the limits discussed in this section. The resources\_min.<resource name> and resources\_max.<resource name> server and queue attributes are limits on what each individual job may use. See [section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 251](#) and [section 5.15.2, “Placing Resource Limits on Jobs”, on page 300](#).

---

### 5.15.1.1 Examples of Managing Resource Usage at Server and Queues

You can limit resource usage and job count for specific projects, users and groups:

- UserA can use no more than 6 CPUs, and UserB can use no more than 4 CPUs, at one time anywhere in the PBS complex.
- The crashtest group can use no more than 16 CPUs at one time anywhere in the PBS complex.
- UserC accidentally submitted 200,000 jobs last week. UserC can now have no more than 25 jobs enqueued at one time.
- All jobs request the server-level custom resource `nodehours`, which is used for allocation. UserA cannot use more than 40 `nodehours` in the PBS complex. Once UserA reaches the `nodehours` limit, then all queued jobs owned by UserA are not eligible for execution.
- You wish to allow UserD to use 12 CPUs but limit all other users to 4 CPUs.
- Jobs belonging to Project A can use no more than 8 CPUs at Queue1.

You can limit the number of jobs a particular project, user or group runs in a particular queue:

- UserE can use no more than 2 CPUs at one time at Queue1, and 6 CPUs at one time at Queue2.
- You wish to limit UserF to 10 running jobs in queue Queue3, but allow all other users unlimited jobs running in the same queue.
- UserG is a member of Group1. You have a complex-wide limit of 5 running jobs for UserG. You have a limit at Queue1 of 10 running jobs for Group1. This way, up to 10 of the running jobs in Queue1 can belong to Group1, and 5 of these can belong to UserG.
- UserH is a member of Group1. You have a complex-wide limit of 5 running jobs for UserH. You have a limit at Queue1 of 10 running jobs for any group in Queue1. This way, no group in Queue1 can run more than 10 jobs total at one time, and 5 of these can belong to UserH.
- UserJ is a member of Group1. You have a complex-wide limit of 10 running jobs for UserJ. You also have a limit at Queue1 of 5 running jobs for Group1. This means that there may be up to 5 running jobs owned by users belonging to Group1 in Queue1, and up to 5 of these can be owned by UserJ. UserJ can also have another 5 running jobs owned by Group1 in any other queue, or owned by a different group in Queue1.
- No more than 12 jobs belonging to Project A can run at Queue1, and all other projects are limited to 8 jobs at Queue1.

You can ensure fairness in the use of resources:

- You have multiple departments which have shared the purchase of a large machine. Each department would like to ensure fairness in the use of the machine, by setting limits on individual users and groups.
- You have multiple departments, each of which purchases its own machines. Each department would like to limit the use of its machines so that all departmental users have specific limits. In addition, each department would like to allow non-departmental users to use its machines when they are under-utilized, while giving its own users priority on its machines. A non-departmental user can run jobs on a departmental machine, as long as no departmental users' jobs are waiting to run.

### 5.15.1.2 Glossary

#### Limit

The maximum amount of a resource that can be consumed at any time by running jobs or allocated to queued jobs, or the maximum number of jobs that can be running, or the maximum number of jobs that can be queued.



---

**Overall limit**

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

**Generic user limit**

Applies separately to users at the server or a queue. The limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

**Generic group limit**

Applies separately to groups at the server or a queue. The limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

**Generic project limit**

Applies separately to projects at the server or a queue. The limit for projects which have no individual limit specified. A limit for generic projects is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

**Individual user limit**

Applies separately to users at the server or a queue. Limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

**Individual group limit**

Applies separately to groups at the server or a queue. Limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

**Individual project limit**

Applies separately to projects at the server or a queue. Limit for a project which has its own individual limit specified. An individual project limit overrides the generic project limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

**User limit**

A limit placed on one or more users, whether generic or individual.

**Group limit**

This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

**Project**

In PBS, a project is a way to group jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's `project` attribute specifies the job's project.

**Project limit**

This is a limit applied to the total used by a project, whether the limit is a generic project limit or an individual project limit.

**Queued jobs**

In a queue, queued jobs are the jobs that are waiting in that queue.

---

### 5.15.1.3 Difference Between PBS\_ALL and PBS\_GENERIC

Note the very important **difference** between the *overall limit* and a *generic* limit. We will describe how this works for users, but this applies to other entities as well. You set PBS\_ALL for an overall limit on the total usage of that resource by all entities, whereas you set PBS\_GENERIC for a limit for any single generic user.

Example 5-12: Difference between overall limit and generic user limit

Given the following:

- The overall server limit for running jobs is *100*
- The server limit for generic users is *10*
- The individual limit for User1 is 12 jobs

This means:

- Generic users (any single user except User1) can run no more than 10 jobs at this server
- User1 can run 12 jobs at this server
- At this server, no more than 100 jobs can be running at any time

### 5.15.1.4 Hard and Soft Limits

Hard limits are limits which cannot be exceeded. Soft limits are limits which mark the point where a project, user or group is using "extra, but acceptable" amounts of a resource. When this happens, the jobs belonging to that project, user or group are eligible for preemption. See [section 4.9.33, "Using Preemption", on page 180](#). Soft limits are discussed in [section 4.9.33.7.i, "The Soft Limits Preemption Level", on page 184](#).

### 5.15.1.5 Scope of Limits at Server and Queues

Each of the limits described above can be set separately at the server and at each queue. Each limit's scope is the PBS object where it is set. The individual and generic project, user and group limits that are set within one scope interact with each other only within that scope. For example, a limit set at one queue has no effect at another queue.

The scope of limits set at the server encompasses queues, so that the minimum, more restrictive limit of the two is applied. For precedence within a server or queue, see [section 5.15.1.7, "Precedence of Limits at Server and Queues", on page 289](#).

---

### 5.15.1.6 Ways To Limit Resource Usage at Server and Queues

You can create a complete set of limits at the server, and you can create another complete set of limits at each queue. You can set hard and soft limits. See [section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 184](#). You can limit resource usage at the server and the queue level for the following:

- Running jobs
  - Number of running jobs
  - Number of running jobs (soft limit)
  - Amount of each resource allocated for running jobs
  - Amount of each resource allocated for running jobs (soft limit)
- Queued jobs (this means jobs that are waiting to run from that queue)
  - Number of queued jobs
  - Amount of each resource allocated for queued jobs
- Queued and running jobs (this means both jobs that are waiting to run and jobs that are running from that queue)
  - Number of queued and running jobs
  - Amount of each resource allocated for queued and running jobs

These limits can be applied to the following:

- The total usage at the server
- The total usage at each queue
- Amount used by a single user
  - Generic users
  - Individual users
- Amount used by a single group
  - Generic groups
  - Individual groups
- Amount used by a single project
  - Generic projects
  - Individual projects

#### 5.15.1.6.i Limits at Queues

You can limit the number of jobs that are queued at a queue, and running at a queue, and that are both queued and running at a queue.

You can limit the resources allocated to jobs that are queued at a queue, and running at a queue, and that are both queued and running at a queue.

Jobs queued at a queue are counted the same whether they were submitted to that queue via the `qsub` command or its equivalent API, moved to that queue via the `qmove` command or its equivalent API, or routed to that queue from another queue.

When PBS requeues a job, it does not take limits into account.

Routing queues do not run jobs, so you cannot set a limit for the number of running jobs, or the amount of resources being used by running jobs, at a routing queue.

### 5.15.1.6.ii Generic and Individual Limits

You can set a generic limit for groups, so that each group must obey the same limit. You can likewise set a generic limit for users and projects. Each generic limit can be set separately at the server and at each queue. For example, if you have two queues, the generic limit for the number of jobs a user can run be 4 at QueueA and 6 at QueueB.

You can set a different individual limit for each user, and you can set individual limits for groups and for projects. Each user, group, and project can have a different individual limit at the server and at each queue.

You can use a combination of generic and individual project, user or group limits, at the server and at each queue. Within the scope of the server or a queue, all projects, users or groups except the ones with the individual limits must obey the generic limit, and the individual limits override the generic limits.

Example 5-13: Generic and individual user limits on running jobs at QueueA and QueueB

At QueueA:

- At QueueA, the generic user limit is 5
- At QueueA, Bob's individual limit is 8
- Tom has no individual limit set at QueueA; the generic limit applies

At QueueB:

- At QueueB, the generic user limit is 2
- At QueueB, Tom's individual limit is 1
- Bob has no individual limit at QueueB; the generic limit applies

This means:

- Bob can run 8 jobs at QueueA
- Bob can run 2 jobs at QueueB
- Tom can run 5 jobs at QueueA
- Tom can run 1 job at QueueB

### 5.15.1.6.iii Overall Limits

The overall limit places a cap on the total amount of the resource that can be used within the scope in question (server or queue), regardless of whether project, user, or group limits have been reached. A project, user, or group at the server or a queue cannot use any more of a resource for which the overall limit has been reached, even if that project, user, or group limit has not been reached.

Example 5-14: Overall limit at server

Given the following:

- Overall server limit on running jobs is 100
- Bob's user limit is 10 running jobs
- 98 jobs are already running
- Bob is running zero jobs

This means:

- Bob can start only 2 jobs

### 5.15.1.7 Precedence of Limits at Server and Queues

#### 5.15.1.7.i Interactions Between Limits Within One Scope

Within the scope of a PBS object (server or queue), there is an order of precedence for limits when more than one applies to a job. The order of precedence for the limits at a queue is the same as the order at the server. The following table shows how limits interact within one scope:

**Table 5-12: Limit Interaction Within One Scope**

|                           | <b>Individual User</b> | <b>Generic User</b> | <b>Individual Group</b> | <b>Generic Group</b> | <b>Individual Project</b> | <b>Generic Project</b> |
|---------------------------|------------------------|---------------------|-------------------------|----------------------|---------------------------|------------------------|
| <b>Individual User</b>    | Individual user        | Individual user     | More restrictive        | More restrictive     | More restrictive          | More restrictive       |
| <b>Generic User</b>       | Individual user        | Generic user        | More restrictive        | More restrictive     | More restrictive          | More restrictive       |
| <b>Individual Group</b>   | More restrictive       | More restrictive    | Individual group        | Individual group     | More restrictive          | More restrictive       |
| <b>Generic Group</b>      | More restrictive       | More restrictive    | Individual group        | Generic group        | More restrictive          | More restrictive       |
| <b>Individual Project</b> | More restrictive       | More restrictive    | More restrictive        | More restrictive     | Individual project        | Individual project     |
| <b>Generic Project</b>    | More restrictive       | More restrictive    | More restrictive        | More restrictive     | Individual project        | Generic project        |

An individual user limit overrides a generic user limit.

Example 5-15: Individual user limit overrides generic user limit

Given the following:

- Bob has a limit of 10 running jobs
- The generic limit is 5

This means:

- Bob can run 10 jobs

An individual group limit overrides a generic group limit in the same manner as for users.

If the limits for a user and the user's group are different, the more restrictive limit applies.

Example 5-16: More restrictive user or group limit applies

Given the following:

- Tom's user limit for running jobs is 8
- Tom's group limit is 7

This means:

- Tom can run only 7 jobs in that group

If a user belongs to more than one group, that user can run jobs up to the lesser of his user limit or the sum of the group limits.

Example 5-17: User can run jobs in more than one group

Given the following:

- Tom's user limit is *10* running jobs
- GroupA has a limit of *2* and GroupB has a limit of *4*
- Tom belongs to GroupA and GroupB

This means:

- Tom can run 6 jobs, 2 in GroupA and 4 in GroupB

An individual project limit overrides a generic project limit, similar to the way user and group limits work.

Project limits are applied independently of user and group limits.

Example 5-18: Project limits are applied without regard to user and group limits

Given the following:

- Project A has a limit of 2 jobs
- Bob has an individual limit of 4 jobs
- Bob's group has a limit of 6 jobs
- Bob is running 2 jobs, both in Project A

This means:

- Bob cannot run any more jobs in Project A

#### **5.15.1.7.ii Interactions Between Queue and Server Limits**

If the limits for a queue and the server are different, the more restrictive limit applies.

Example 5-19: More restrictive queue or server limit applies

Given the following:

- Server limit on running jobs for generic users is *10*
- Queue limit for running jobs from QueueA for generic users is *15*
- Queue limit for running jobs from QueueB for generic users is *5*

This means:

- Generic users at QueueA can run 10 jobs
- Generic users at QueueB can run 5 jobs

Example 5-20: More restrictive queue or server limit applies

Given the following:

- Bob's user limit on running jobs, set on the server, is *7*
- Bob's user limit on running jobs, set on QueueA, is *6*

This means:

- Bob can run 6 jobs from QueueA

#### **5.15.1.8 Resource Usage Limit Attributes for Server and Queues**

Each of the following attributes can be set at the server and each queue:

`max_run`

The maximum number of jobs that can be running.

`max_run_soft`

The soft limit on the maximum number of jobs that can be running.

**max\_run\_res.<resource name>**

The maximum amount of the specified resource that can be allocated to running jobs.

**max\_run\_res\_soft.<resource name>**

The soft limit on the amount of the specified resource that can be allocated to running jobs.

**max\_queued**

The maximum number of jobs that can be queued and running. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

**max\_queued\_res.<resource name>**

The maximum amount of the specified resource that can be allocated to queued and running jobs. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

**queued\_jobs\_threshold**

The maximum number of jobs that can be queued. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

**queued\_jobs\_threshold\_res.<resource name>**

The maximum amount of the specified resource that can be allocated to queued jobs. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

Each attribute above can be used to specify all of the following:

- An overall limit (at the queue or server)
- A limit for generic users
- Individual limits for specific users
- A limit for generic projects
- Individual limits for specific projects
- A limit for generic groups
- Individual limits for specific groups



For example, you can specify the limits for the number of running jobs:

- In the complex:
  - The overall server limit (all usage in the entire complex) is *10,000*
  - The limit for generic users is *5*
  - The limit for Bob is *10*
  - The limit for generic groups is *50*
  - The limit for group GroupA is *75*
  - The limit for generic projects is *25*
  - The limit for Project A is *35*
- At QueueA:
  - The overall queue limit (all usage in QueueA) is *200*
  - The limit for generic users is *2*
  - The limit for Bob is *1*
  - The limit for generic groups is *3*
  - The limit for group GroupA is *7*
  - The limit for generic projects is *10*
  - The limit for Project A is *15*
- At QueueB:
  - The overall queue limit (all usage in QueueB) is *500*
  - The limit for generic users is *6*
  - The limit for Bob is *8*
  - The limit for generic groups is *15*
  - The limit for group GroupA is *11*
  - The limit for generic projects is *20*
  - The limit for Project A is *30*

### 5.15.1.9 How to Set Limits at Server and Queues

You can set, add, and remove limits by using the `qmgr` command to set limit attributes.

#### 5.15.1.9.i Syntax

Format for setting a limit attribute:

```
set server <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

```
set <queue> <queue name> <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

Format for adding a limit to an attribute:

```
set server <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

```
set <queue> <queue name> <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

Format for removing a limit from an attribute; note that the value for `<limit>` need not be specified when removing a limit:

```
set server <limit attribute> -= "[limit-spec], [limit-spec],..."
```

```
set <queue> <queue name> <limit attribute> -= "[limit-spec], [limit-spec],..."
```

Alternate format for removing a limit from an attribute; note that the value of *<limit>* used when removing a limit must match the value of the limit:

```
set server <limit attribute> -= "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

```
set <queue> <queue name> <limit attribute> -= "[limit-spec=<limit>], [limit-spec=<limit>],..."
```

where *limit-spec* specifies a user limit, a group limit, or an overall limit:

**Table 5-13: Specifying Limits**

| Limit                 | limit-spec                    |
|-----------------------|-------------------------------|
| Overall limit         | <i>o:PBS_ALL</i>              |
| Generic users         | <i>u:PBS_GENERIC</i>          |
| An individual user    | <i>u:&lt;username&gt;</i>     |
| Generic groups        | <i>g:PBS_GENERIC</i>          |
| An individual group   | <i>g:&lt;group name&gt;</i>   |
| Generic projects      | <i>p:PBS_GENERIC</i>          |
| An individual project | <i>p:&lt;project name&gt;</i> |

The *limit-spec* can contain spaces anywhere except after the colon (":").

If there are comma-separated *limit-specs*, the entire string must be enclosed in double quotes.

A username, group name, or project name containing spaces must be enclosed in quotes.

If a username, group name, or project name is quoted using double quotes, and the entire string requires quotes, the outer enclosing quotes must be single quotes. Similarly, if the inner quotes are single quotes, the outer quotes must be double quotes.

*PBS\_ALL* is a keyword which indicates that this limit applies to the usage total.

*PBS\_GENERIC* is a keyword which indicates that this limit applies to generic users or groups.

When removing a limit, the limit value does not need to be specified.

*PBS\_ALL* and *PBS\_GENERIC* are case-sensitive.

### 5.15.1.9.ii Examples of Setting Server and Queue Limits

Example 5-21: To set the `max_queued` limit on QueueA to 5 for total usage, and to limit user bill to 3:

```
Qmgr: s q QueueA max_queued = "[o:PBS_ALL=5], [u:bill =3]"
```

Example 5-22: On QueueA, set the maximum number of CPUs and the maximum amount of memory that user bill can request in his queued jobs:

```
Qmgr: s q QueueA max_queued_res.ncpus = "[u:bill=5]", max_queued_res.mem =
 "[u:bill=100mb]"
```

Example 5-23: To set a limit for a username with a space in it, and to set a limit for generic groups:

```
Qmgr: s q QueueA max_queued = '[u:"\PROG\Named User" = 1], [g:PBS_GENERIC=4]'
```

Example 5-24: To set a generic server limit for projects, and an individual server limit for Project A:

```
Qmgr: set server max_queued = '[p:PBS_GENERIC=6], [p:ProjectA=8]'
```

### 5.15.1.9.iii Examples of Adding Server and Queue Limits

Example 5-25: To add an overall limit for the maximum number of jobs that can be queued at QueueA to 10:

```
Qmgr: s q QueueA max_queued += [o:PBS_ALL=10]
```

Example 5-26: To add an individual user limit, an individual group limit, and a generic group limit on queued jobs at QueueA:

```
Qmgr: s q QueueA max_queued += "[u:user1= 5], [g:GroupMath=5], [g:PBS_GENERIC=2] "
```

Example 5-27: To add a limit at QueueA on the number of CPUs allocated to queued jobs for an individual user, and a limit at QueueA on the amount of memory allocated to queued jobs for an individual user:

```
Qmgr: s q QueueA max_queued_res.ncpus += [u:tom=5], max_queued_res.mem += [u:tom=100mb]
```

Example 5-28: To add an individual server limit for Project B:

```
Qmgr: set server max_queued += [p:ProjectB=4]
```

### 5.15.1.9.iv Examples of Removing Server and Queue Limits

It is not necessary to specify the value of the limit when removing a limit, but you can specify the value of the limit.

Example 5-29: To remove the generic user limit at QueueA for queued jobs, use either of the following:

```
Qmgr: set queue QueueA max_queued -= [u:PBS_GENERIC]
Qmgr: set queue QueueA max_queued -= [u:PBS_GENERIC=2]
```

Example 5-30: To remove the limit on queued jobs at QueueA for *Named User*, use either of the following:

```
Qmgr: set queue QueueA max_queued -= [u:"\PROG\Named User"]
Qmgr: set queue QueueA max_queued -= [u:"\PROG\Named User"]=1]
```

Example 5-31: To remove the limit at QueueA on the amount of memory allocated to an individual user, use either of the following:

```
Qmgr: set queue QueueA max_queued_res.mem -= [u:tom]
Qmgr: set queue QueueA max_queued_res.mem -= [u:tom=100mb]
```

To remove the limit on the number of CPUs allocated to queued jobs for user bill, use either of the following:

```
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill]
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill=5]
```

Example 5-32: To remove a generic user limit and an individual user limit, use either of the following:

```
Qmgr: set queue QueueA max_queued - -= "[u:user1], [u:PBS_GENERIC] "
Qmgr: set queue QueueA max_queued - -= "[u:user1=2], [u:PBS_GENERIC=4] "
```

Example 5-33: To remove the individual server limit for Project B, use either of the following:

```
Qmgr: set server max_queued -= [p:ProjectB]
Qmgr: set server max_queued -= [p:ProjectB=4]
```

### 5.15.1.10 Who Can Set Limits at Server and Queues

As with other server and queue attributes, only PBS Managers and Operators can set limit attributes.

### 5.15.1.11 Viewing Server and Queue Limit Attributes

#### 5.15.1.11.i Printing Server and Queue Limit Attributes

You can use the `qmgr` command to print the commands used to set the limit attributes at the server or queue.

Example 5-34: To print all the limit attributes for queue QueueA:

```
Qmgr: p q QueueA max_queued, max_queued_res
#
Create queues and set their attributes.
#
Create and define queue QueueA
#
create queue QueueA
set queue QueueA max_queued = "[o:PBS_ALL=10]"
set queue QueueA max_queued += "[u:PBS_GENERIC=2]"
set queue QueueA max_queued += "[u:bill=3]"
set queue QueueA max_queued += "[u:tom=15]"
set queue QueueA max_queued += "[u:user1=3]"
set queue QueueA max_queued += '[u:"\PROG\Named User"]=1]'
set queue QueueA max_queued += "[g:PBS_GENERIC=2] "
set queue QueueA max_queued += "[g:GroupMath=5]"
set queue QueueA max_queued_res.ncpus = "[u:bill=5]"
set queue QueueA max_queued_res.ncpus += "[u:tom=5]"
set queue QueueA max_queued_res.mem = "[u:bill=100mb]"
set queue QueueA max_queued_res.mem += "[u:tom=100mb]"
```

#### 5.15.1.11.ii Listing Server and Queue Limit Attributes

You can use the `qmgr` command to list the limit attributes for the queue or server.

Example 5-35: To list the `max_queued` and `max_queued_res` attributes for QueueA:

```
Qmgr: l q QueueA max_queued, max_queued_res
Queue: QueueA
 max_queued = [o:PBS_ALL=10]
 max_queued = [g:PBS_GENERIC=2]
 max_queued = [g:GroupMath=5]
 max_queued = [u:PBS_GENERIC=2]
 max_queued = [u:bill=3]
 max_queued = [u:tom=15]
 max_queued = [u:user1=3]
 max_queued = [u:"\PROG\Named User"]=1]
 max_queued_res.ncpus = [u:bill=5]
 max_queued_res.ncpus = [u:tom=5]
 max_queued_res.mem = [u:bill=5]
 max_queued_res.mem = [u:bill=100mb]
 max_queued_res.mem = [u:tom=100mb]
```

### 5.15.1.11.iii Using the `qstat` Command to View Queue Limit Attributes

You can use the `qstat` command to see the limit attribute settings for the queue or server.

Example 5-36: To see the settings for the `max_queued` and `max_queued_res` limit attributes for QueueA using the `qstat` command:

```
qstat -Qf QueueA
Queue: QueueA
...
max_queued = [o:PBS_ALL=10]
max_queued = [g:PBS_GENERIC=2]
max_queued = [g:GroupMath=5]
max_queued = [u:PBS_GENERIC=2]
max_queued = [u:bill=3]
max_queued = [u:tom=3]
max_queued = [u:cs=3]
max_queued = [u:"\PROG\Named User"]=1]
max_queued_res.ncpus = [u:bill=5]
max_queued_res.ncpus = [u:tom=5]
max_queued_res.mem = [u:bill=5]
max_queued_res.mem = [u:bill=100mb]
max_queued_res.mem = [u:tom=100mb]
```

### 5.15.1.12 How Server and Queue Limits Work

*Affected jobs* are jobs submitted by the user or group, or jobs belonging to a project, whose limit has been reached. The following table shows what happens when a given limit is reached:

**Table 5-14: Actions Performed When Limits Are Reached**

| Limit                      | Action                                                                                                                                                                                                                                                                                                                 |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Running jobs               | No more affected jobs are run at this server or queue until the number of affected running jobs drops below the limit.                                                                                                                                                                                                 |
| Queued jobs                | The queue does not accept any more affected jobs until the number of affected queued jobs drops below the limit. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue. If a job is requeued, the limit is ignored. |
| Resources for running jobs | The queue does not run any more affected jobs until the limit would not be exceeded if the next affected job were to start.                                                                                                                                                                                            |
| Resources for queued jobs  | The queue does not accept any more affected jobs until the limit would not be exceeded if the next affected job were to start. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.                               |

### 5.15.1.13 Caveats and Advice for Server and Queue Limits

#### 5.15.1.13.i Avoiding Overflow

On PBS server platforms for which the native size of a long is less than 64 bits, you should refrain from defining a limit on a resource of type `long` whose cumulative sum over all queued jobs would exceed the storage capacity of the resource variable. For example, if each submitted job were to request 100 hours of the `cpur` resource, overflow would occur on a 32-bit platform when 5965 jobs (which is  $(2^{31} - 1)/360000$  seconds) were queued.

#### 5.15.1.13.ii Ensuring That Limits Are Effective

In order for limits to be effective, each job must specify each limited resource. This can be accomplished using defaults; see [section 5.9.3, “Specifying Job Default Resources”, on page 241](#). You can also use hooks; see the PBS Professional Hooks Guide.

#### 5.15.1.13.iii Array Jobs

An array job with  $N$  subjobs is considered to consume  $N$  times the amount of resources requested when it was submitted. For example, if there is a server limit of 100 queued jobs, no user would be allowed to submit an array job with more than 100 subjobs.

#### 5.15.1.13.iv Avoiding Job Rejection

Jobs are rejected when users, groups, or projects who have reached their limit submit a job in the following circumstances:

- The job is submitted to the execution queue where the limit has been reached
- The job is submitted to the complex, and the server limit has been reached

If you wish to avoid having jobs be rejected, you can set up a routing queue as the default queue. Set the server's `default_queue` attribute to the name of the routing queue. See [section 2.3.6, “Routing Queues”, on page 27](#).

#### 5.15.1.13.v Do Not Mix Old And New Limits

The new limit attributes are incompatible with the old limit attributes. See [section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 298](#). You cannot mix the use of old and new resource usage limit attributes. This means that:

- If any old limit attribute is set, and you try to set a new limit attribute, you will get error 15141.
- If any new limit attribute is set, and you try to set an old limit attribute, you will get error 15141.

You must unset all of one kind in order to set any of the other kind.

#### 5.15.1.13.vi Do Not Limit Running Time

Beware creating limits such as `max_run_res.walltime` or `max_run_res.max_walltime`. The results probably will not be useful. You will be limiting the amount of walltime that can be requested by running jobs for a user, group, or project. For example, if you set a walltime limit of 10 hours for group A, then group A cannot run one job requesting 5 hours and another job requesting 6 hours.

### 5.15.1.14 Errors and Logging for Server and Queue Limits

#### 5.15.1.14.i Error When Setting Limit Attributes

Attempting to set a new limit attribute while an old limit attribute is set:

```
"use new/old qmgr syntax, not both"
```

```
"Attribute name <new> not allowed. Older name <old> already set"
```

Attempting to set an old limit attribute while a new limit attribute is set:

```
"use new/old qmgr syntax, not both"
"Attribute name <old> not allowed: Newer name <new> already set''
```

#### 5.15.1.14.ii Logging Events

Whenever a limit attribute is set or modified, the server logs the event, listing which attribute was modified and who modified it.

Whenever a limit is reached, and would be exceeded by a job, the scheduler logs the event, listing the limit attribute and the reason.

#### 5.15.1.14.iii Queued Limit Error Messages

When a limit for queued jobs or resources allocated to queued jobs is reached, the command involved presents a message. This command can be `qsub`, `qmove` or `qalter`.

#### 5.15.1.14.iv Run Limit Error Messages

See [“Run Limit Error Messages” on page 385 of the PBS Professional Reference Guide](#) for a list of run limit error messages.

### 5.15.1.15 Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1

The old server and queue limit attributes discussed here existed in PBS Professional before Version 10.1. The old limit attributes continue to function as they did in PBS Professional 10.0. These attributes are incompatible with the limit attributes introduced in Version 10.1. See [section 5.15.1.13.v, “Do Not Mix Old And New Limits”, on page 297](#) and [section 5.15.1.14.i, “Error When Setting Limit Attributes”, on page 297](#).

The following table shows how the old limit attributes are used:

**Table 5-15: Resource Usage Limits Existing Before Version 10.1**

| Limit                                                                       | Overall Limit | Generic Users     | Generic Groups     | Individual Users | Individual Group |
|-----------------------------------------------------------------------------|---------------|-------------------|--------------------|------------------|------------------|
| Maximum number of running jobs                                              | max_running   | max_user_run      | max_group_run      | N/A              | N/A              |
| Maximum number of running jobs (soft limit)                                 | N/A           | max_user_run_soft | max_group_run_soft | N/A              | N/A              |
| Maximum amount of specified resource allocated to running jobs              | N/A           | max_user_res      | max_group_res      | N/A              | N/A              |
| Maximum amount of specified resource allocated to running jobs (soft limit) | N/A           | max_user_res_soft | max_group_res_soft | N/A              | N/A              |
| Maximum number of queued jobs                                               | max_queuable  | N/A               | N/A                | N/A              | N/A              |
| Maximum amount of specified resource allocated to queued jobs               | N/A           | N/A               | N/A                | N/A              | N/A              |



### 5.15.1.15.i Precedence of Old Limits

If an old limit is defined at both the server and queue, the more restrictive limit applies.

### 5.15.1.15.ii Old Server Limits

For details of these limits, see [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

`max_running`

The maximum number of jobs allowed to be selected for execution at any given time.

`max_group_res,`

`max_group_res_soft`

The maximum amount of the specified resource that all members of the same Linux group may consume simultaneously.

`max_group_run,`

`max_group_run_soft`

The maximum number of jobs owned by a Linux group that are allowed to be running from this server at one time.

`max_user_res,`

`max_user_res_soft`

The maximum amount of the specified resource that any single user may consume.

`max_user_run,`

`max_user_run_soft`

The maximum number of jobs owned by a single user that are allowed to be running at one time.

### 5.15.1.15.iii Old Queue Limits

For details of these limits, see [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#).

`max_group_res,`

`max_group_res_soft`

The maximum amount of the specified resource that all members of the same Linux group may consume simultaneously, in the specified queue.

`max_group_run,`

`max_group_run_soft`

The maximum number of jobs owned by a Linux group that are allowed to be running from this queue at one time

`max_queueable`

The maximum number of jobs allowed to reside in the queue at any given time. Once this limit is reached, no new jobs will be accepted into the queue.

`max_user_res,`

`max_user_res_soft`

The maximum amount of the specified resource that any single user may consume in submitting to this queue.

`max_user_run,`

`max_user_run_soft`

The maximum number of jobs owned by a single user that are allowed to be running at one time from this queue.

## 5.15.2 Placing Resource Limits on Jobs

Jobs are assigned limits on the amount of resources they can use. Each limit is set at the amount requested or allocated by default. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources. For information on default resources, see [section 5.9.3, “Specifying Job Default Resources”, on page 241](#).

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

The consumable resources requested for chunks in the select specification are summed, and this sum makes a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Job resource limits set a limit for per-job resource usage. Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

### 5.15.2.1 How Limits Are Derived

Job resource limits are derived in this order from the following:

1. Explicitly requested job-wide resources (e.g. `-l resource=value`)
2. The following built-in chunk-level resources in the job's select specification (e.g. `-l select =...`)
  - `mem`
  - `mpiprocs`
  - `ncpus`
  - `nodect`
  - `vmem`
3. The server's `default_qsub_arguments` attribute
4. The queue's `resources_default.<resource name>`
5. The server's `resources_default.<resource name>`
6. The queue's `resources_max.<resource name>`
7. The server's `resources_max.<resource name>`

The server's `default_chunk.<resource name>` does **not** affect job-wide limits.

You can use a hook to set a per-chunk limit, using any hook that operates on jobs, such as a job submission hook, a modify job hook, etc.

### 5.15.2.2 Configuring Per-job Limits at Server and Queue

You can set per-job limits on the amount of each resource that any one job can use. You can set these limits at the server and at each queue. For example, you can specify the following limits:

- Jobs at the server can use no more than 48 hours of CPU time
- Jobs at QueueA can use no more than 12 hours of CPU time
- Jobs at QueueA must request more than 2 hours of CPU time

To set these limits, specify values for the server's `resources_max.<resource name>` attribute and each queue's `resources_max.<resource name>` and `resources_min.<resource name>` attributes. The server does not have a `resources_min.<resource name>` attribute. To set the maximum at the server, the format is:

*Qmgr: set server resources\_max.<resource name> = value*

To set the maximum and minimum at the queue, the format is:

*Qmgr: set queue <queue name> resources\_max.<resource name> = value*

*Qmgr: set queue <queue name> resources\_min.<resource name> = value*

For example, to set the 48 hour CPU time limit:

```
Qmgr: set server resources_max.cput = 48:00:00
```

### 5.15.2.2.i Running Time Limits at Server and Queues

For non-shrink-to-fit jobs, you can set limits on `walltime` at the server or queue. To set a `walltime` limit for non-shrink-to-fit jobs at the server or a queue, use `resources_max.walltime` and `resources_min.walltime`.

For shrink-to-fit jobs, running time limits are applied to `max_walltime` and `min_walltime`, not `walltime`. To set a running time limit for shrink-to-fit jobs, you cannot use `resources_max` or `resources_min` for `max_walltime` or `min_walltime`. Instead, use `resources_max.walltime` and `resources_min.walltime`. See [section 4.9.42.6, “Shrink-to-fit Jobs and Resource Limits”](#), on page 213.

### 5.15.2.3 Configuring Per-job Resource Limit Enforcement at Vnodes

For a job, enforcement of resource limits is per-MoM, not per-vnode. So if a job requests 3 chunks, each of which has 1MB of memory, and all chunks are placed on one host, the limit for that job for memory for that MoM is 3MB. Therefore one chunk can be using 2 MB and the other two using 0.5MB and the job can continue to run.

Job resource limits can be enforced for single-vnode jobs, or for multi-vnode jobs that are using a PBS-aware MPI. See the following table for an overview. Memory limits are handled differently depending on the operating system. See [“Job Memory Limit Enforcement on Linux” on page 302](#). The `ncpus` limit can be adjusted in several ways. See [“Job ncpus Limit Enforcement” on page 303](#) for a discussion. The following table summarizes how resource limits are enforced at vnodes:

**Table 5-16: Resource Limit Enforcement at Vnodes**

| Limit     | What Determines When Limit Is Enforced                                                                                                                             | Scope of Limit | Enforcement Method       |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|--------------------------|
| file size | automatically                                                                                                                                                      | per-process    | <code>setrlimit()</code> |
| vmem      | If job requests or inherits vmem                                                                                                                                   | job-wide       | MoM poll                 |
| pvmem     | If job requests or inherits pvmem                                                                                                                                  | per-process    | <code>setrlimit()</code> |
| pmem      | If job requests or inherits pmem                                                                                                                                   | per-process    | <code>setrlimit()</code> |
| pcput     | If job requests or inherits pcput                                                                                                                                  | per-process    | <code>setrlimit()</code> |
| cput      | If job requests or inherits cput                                                                                                                                   | job-wide       | MoM poll                 |
| walltime  | If job requests or inherits walltime                                                                                                                               | job-wide       | MoM poll                 |
| mem       | if <code>\$enforce mem</code> in MoM's config                                                                                                                      | job-wide       | MoM poll                 |
| ncpus     | if <code>\$enforce cpuaverage</code> , <code>\$enforce cpuburst</code> , or both, in MoM's config. See <a href="#">“Job ncpus Limit Enforcement” on page 303</a> . | job-wide       | MoM poll                 |

### 5.15.2.4 Job Memory Limit Enforcement

You may wish to prevent jobs from swapping memory. To prevent this, you can set limits on the amount of memory a job can use. Then the job must request an amount of memory equal to or smaller than the amount of physical memory available.

PBS measures and enforces memory limits in two ways:

- On each host, by setting OS-level limits, using the limit system calls
- By periodically summing the usage recorded in the `/proc` entries.

Enforcement of `mem` is dependent on the following:

- Adding `$enforce mem` to the MoM's `config` file
- The job requesting or inheriting a default value for `mem`

You can configure default `qsub` parameters in the `default_qsub_arguments` server attribute, or set memory defaults at the server or queue. See [section 5.9.3, “Specifying Job Default Resources”, on page 241](#).

#### 5.15.2.4.i Job Memory Limit Enforcement on Linux

By default, memory limits are not enforced. To enforce `mem` resource usage, put `$enforce mem` into MoM's `config` file, and set defaults for `mem` so that each job inherits a value if it does not request it.

The `mem` resource can be enforced at both the job level and the `vnode` level. The job-wide limit is the smaller of a job-wide resource request and the sum of that for all chunks. The `vnode`-level limit is the sum for all chunks on that host.

Job-wide limits are enforced by MoM polling the working set size of all processes in the job's session. Jobs that exceed their specified amount of physical memory are killed. A job may exceed its limit for the period between two polling cycles. See [section 3.1.2, “Configuring MoM Polling Cycle”, on page 38](#).

Per-process limits are enforced by the operating system kernel. PBS calls the kernel call `setrlimit()` to set the limit for the top process (the shell), and any process started by the shell inherits those limits. PBS does not know whether the kernel kills a process for exceeding the limit.

If a user submits a job with a job limit, but not per-process limits (`qsub -l cput=10:00`) then PBS sets the per-process limit to the same value. If a user submits a job with both job and per-process limits, then the per-process limit is set to the lesser of the two values.

Example: a job is submitted with `qsub -lcput=10:00`

- There are two CPU-intensive processes which use 5:01 each. The job will be killed by PBS for exceeding the `cput` limit. 5:01 + 5:01 is greater than 10:00.
- There is one CPU-intensive process which uses 10:01. It is very likely that the kernel will detect it first.
- There is one process that uses 0:02 and another that uses 10:00. PBS may or may not catch it before the kernel does depending on exactly when the polling takes place.

If a job is submitted with a `pmem` limit, or without `pmem` but with a `mem` limit, PBS uses the `setrlimit(2)` call to set the limit. For most operating systems, `setrlimit()` is called with `RLIMIT_RSS` which limits the Resident Set (working set size). This is not a hard limit, but advice to the kernel. This process becomes a prime candidate to have memory pages reclaimed.

If `vmem` is specified and no single process exceeds that limit, but the total usage by all the processes in the job does, then PBS enforces the `vmem` limit, but not the `pvmem` limit, and logs a message. PBS uses MoM polling to enforce `vmem`.

The limit for `pmem` is enforced if the job specifies, or inherits a default value for, `pmem`. When `pmem` is enforced, the limit is set to the smaller of `mem` and `pmem`. Enforcement is done by the kernel, and applies to any single process in the job.

The limit for `pvmem` is enforced if the job specifies, or inherits a default value for, `pvmem`. When `pvmem` is enforced, the limit is set to the smaller of `vmem` and `pvmem`. Enforcement is done by the kernel, and applies to any single process in the job.

The following table shows which OS resource limits can be used by each operating system.

**Table 5-17: RLIMIT Usage in PBS Professional**

| OS    | file         | mem/pmem   | vmem/pvmem | cput/pcput |
|-------|--------------|------------|------------|------------|
| Linux | RLIMIT_FSIZE | RLIMIT_RSS | RLIMIT_AS  | RLIMIT_CPU |

Note that `RLIMIT_RSS`, `RLIMIT_UMEM`, and `RLIMIT_VMEM` are not standardized (i.e. do not appear in the Open Group Base Specifications Issue 6).

#### 5.15.2.4.ii Memory Enforcement on cpusets

There should be no need to do so: either the vnode containing the memory in question has been allocated exclusively (in which case no other job will also be allocated this vnode, hence this memory) or the vnode is shareable (in which case using `mem_exclusive` would prevent two CPU sets from sharing the memory). Essentially, PBS enforces the equivalent of `mem_exclusive` by itself.

### 5.15.2.5 Job ncpus Limit Enforcement

Enforcement of the `ncpus` limit (number of CPUs used) is available on all platforms. The `ncpus` limit can be enforced using average CPU usage, burst CPU usage, or both. By default, enforcement of the `ncpus` limit is off. See [“\\$enforce <limit>” on page 245 of the PBS Professional Reference Guide](#).

#### 5.15.2.5.i Average CPU Usage Enforcement

Each MoM enforces `cpuaverage` independently, per MoM, not per vnode. To enforce average CPU usage, put `$enforce cpuaverage` in MoM's `config` file. You can set the values of three variables to control how the average is enforced. These are shown in the following table.

**Table 5-18: Variables Used in Average CPU Usage**

| Variable                          | Type    | Description                                                                                                                                                      | Default            |
|-----------------------------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <code>cpuaverage</code>           | Boolean | If present ( <code>=True</code> ), MoM enforces <code>ncpus</code> when the average CPU usage over the job's lifetime usage is greater than the specified limit. | <code>False</code> |
| <code>average_trialperiod</code>  | integer | Modifies <code>cpuaverage</code> . Minimum job <code>walltime</code> before enforcement begins. Seconds.                                                         | 120                |
| <code>average_percent_over</code> | integer | Modifies <code>cpuaverage</code> . Percentage by which the job may exceed <code>ncpus</code> limit.                                                              | 50                 |
| <code>average_cpufactor</code>    | float   | Modifies <code>cpuaverage</code> . <code>ncpus</code> limit is multiplied by this factor to produce actual limit.                                                | 1.025              |

Enforcement of `cpuaverage` is based on the polled sum of CPU time for all processes in the job. The limit is checked each poll period. Enforcement begins after the job has had `average_trialperiod` seconds of `walltime`. Then, the job is killed if the following is true:

$$(\text{cput} / \text{walltime}) > (\text{ncpus} * \text{average\_cpufactor} + \text{average\_percent\_over} / 100)$$

### 5.15.2.5.ii CPU Burst Usage Enforcement

To enforce burst CPU usage, put `$enforce cpuburst` in MoM's `config` file. You can set the values of four variables to control how the burst usage is enforced. These are shown in the following table.

**Table 5-19: Variables Used in CPU Burst**

| Variable                        | Type    | Description                                                                                                      | Default      |
|---------------------------------|---------|------------------------------------------------------------------------------------------------------------------|--------------|
| <code>cpuburst</code>           | Boolean | If present ( <code>=True</code> ), MoM enforces <code>ncpus</code> when CPU burst usage exceeds specified limit. | <i>False</i> |
| <code>delta_percent_over</code> | integer | Modifies <code>cpuburst</code> . Percentage over limit to be allowed.                                            | <i>50</i>    |
| <code>delta_cpufactor</code>    | float   | Modifies <code>cpuburst</code> . <code>ncpus</code> limit is multiplied by this factor to produce actual limit.  | <i>1.5</i>   |
| <code>delta_weightup</code>     | float   | Modifies <code>cpuburst</code> . Weighting factor for smoothing burst usage when average is increasing.          | <i>0.4</i>   |
| <code>delta_weightdown</code>   | float   | Modifies <code>cpuburst</code> . Weighting factor for smoothing burst usage when average is decreasing.          | <i>0.1</i>   |

MoM calculates an integer value called `cpupercent` each polling cycle. This is a moving weighted average of CPU usage for the cycle, given as the average percentage usage of one CPU. For example, a value of 50 means that during a certain period, the job used 50 percent of one CPU. A value of 300 means that during the period, the job used an average of three CPUs.

$$new\_percent = change\_in\_cpu\_time * 100 / change\_in\_walltime$$

$$weight = delta\_weight[up|down] * walltime / max\_poll\_period$$

$$new\_cpupercent = (new\_percent * weight) + (old\_cpupercent * (1 - weight))$$

`delta_weight_up` is used if `new_percent` is higher than the old `cpupercent` value. `delta_weight_down` is used if `new_percent` is lower than the old `cpupercent` value. `delta_weight_[up|down]` controls the speed with which `cpupercent` changes. If `delta_weight_[up|down]` is 0.0, the value for `cpupercent` does not change over time. If it is 1.0, `cpupercent` will take the value of `new_percent` for the poll period. In this case `cpupercent` changes quickly.

However, `cpupercent` is controlled so that it stays at the greater of the average over the entire run or `ncpus*100`.

`max_poll_period` is the maximum time between samples, set in MoM's `config` file by `$max_check_poll`, with a default of 120 seconds.

The job is killed if the following is true:

$$new\_cpupercent > ((ncpus * 100 * delta\_cpufactor) + delta\_percent\_over)$$

The following entries in MoM's `config` file turn on enforcement of both average and burst with the default values:

```
$enforce cpuaverage
$enforce cpuburst
$enforce delta_percent_over 50
$enforce delta_cpufactor 1.05
$enforce delta_weightup 0.4
$enforce delta_weightdown 0.1
$enforce average_percent_over 50
$enforce average_cpufactor 1.025
$enforce average_trialperiod 120
```

The `cpuburst` and `cpuaverage` information show up in MoM's log file, whether or not they have been configured in `mom_priv/config`. This is so a site can test different parameters for `cpuburst`/`cpuaverage` before enabling enforcement. You can see the effect of any change to the parameters on your job mix before "going live".

Note that if the job creates a child process whose usage is not tracked by MoM during its lifetime, CPU usage can appear to jump dramatically when the child process exits. This is because the CPU time for the child process is assigned to its parent when the child process exits. MoM may see a big jump in `cpupercent`, and kill the job.

### 5.15.2.5.iii Job Memory Limit Restrictions

Enforcement of `mem` resource usage is available on all Linux platforms, but not Windows.

## 5.15.2.6 Changing Job Limits

The `qalter` command is used to change job limits, with these restrictions:

- A non-privileged user may only lower the limits for job resources
- A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as `pcput` and `pmem`, because these are set when the process starts, and enforced by the kernel.
- When you lengthen the `walltime` of a running job, make sure that the new `walltime` will not interfere with any existing reservations etc.

See [“qalter” on page 130 of the PBS Professional Reference Guide](#).

## 5.15.3 Limiting the Number of Jobs in Queues

If you limit the number of jobs in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the `queued_jobs_threshold` queue or server attribute. See [section 5.15.1.9, “How to Set Limits at Server and Queues”, on page 292](#).

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See [section 4.9.39, “Routing Jobs”, on page 205](#).

## 5.16 Where Resource Information Is Kept

Definitions and values for PBS resources are kept in the following files, attributes, and parameters. Attributes specifying resource limits are not listed here. They are listed in [section 5.15.1.8, “Resource Usage Limit Attributes for Server and Queues”, on page 290](#) and [section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 298](#).

### 5.16.1 Files

`<sched_priv directory>/sched_config`

`resources:` line

In order for scheduler to be able to schedule using a resource, the resource must be listed in the `resources:` line. Format:



*resources: "<resource name>, [<resource name>, ...]"*

Example:

```
resources: "ncpus, mem, arch, [...], FloatLicense, SharedScratch"
```

The only exception is host-level Boolean resources, which do not need to appear in the `resources:` line.

`server_dyn_res:` line

Each dynamic server resource must be listed in its own `server_dyn_res:` line. Format:

*server\_dyn\_res: "<resource name> !<path to script/command>"*

Example:

```
server_dyn_res: "SharedScratch !/usr/local/bin/serverdynscratch.pl"
```

### Version 2 Configuration Files

Contain vnode information. See [section 3.4.3, “Version 2 Vnode Configuration Files”, on page 46](#).

## 5.16.2 MoM Configuration Parameters

`$cputmult <factor>`

This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MoM's system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

```
$cputmult 1.5
```

If MoM's system is slower, set factor to a value between 1.0 and 0.0. For example:

```
$cputmult 0.75
```

`$wallmult <factor>`

Each job's `walltime` usage is multiplied by this factor. For example:

```
$wallmult 1.5
```

## 5.16.3 Attributes

Resources are tracked in the following attributes:

**Table 5-20: Attributes Where Resources Are Tracked**

| Resource Being Tracked                                                                             | Attribute Name                                         |                                                        |                                                   |             |
|----------------------------------------------------------------------------------------------------|--------------------------------------------------------|--------------------------------------------------------|---------------------------------------------------|-------------|
|                                                                                                    | Server and Queue                                       | Vnode                                                  | Job                                               | Reservation |
| Amount of each resource available for use at the object (server, queue, vnode)                     | <code>resources_available.&lt;resource name&gt;</code> | <code>resources_available.&lt;resource name&gt;</code> |                                                   |             |
| Amount of each resource allocated to jobs running and exiting at the object (server, queue, vnode) | <code>resources_assigned.&lt;resource name&gt;</code>  | <code>resources_assigned.&lt;resource name&gt;</code>  |                                                   |             |
| Amount of each resource used by the job                                                            |                                                        |                                                        | <code>resources_used.&lt;resource name&gt;</code> |             |

Table 5-20: Attributes Where Resources Are Tracked

| Resource Being Tracked                                                                                                                                               | Attribute Name                    |       |                               |                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|-------|-------------------------------|-------------------------------|
|                                                                                                                                                                      | Server and Queue                  | Vnode | Job                           | Reservation                   |
| Amount of each job-wide resource that is assigned to any job that does not explicitly request the resource                                                           | resources_default.<resource name> |       |                               |                               |
| Amount of each host-level resource that is assigned to each chunk of any job where that does not explicitly request the resource                                     | default_chunk.<resource name>     |       |                               |                               |
| List of resources requested by the object (job or reservation)                                                                                                       |                                   |       | Resource_List.<resource name> | Resource_List.<resource name> |
| List of chunks for the job. Each chunk shows the name of the vnode from which it is taken along with the host-level, consumable resources allocated from that vnode. |                                   |       | exec_vnode                    |                               |
| List of vnodes and resources allocated to them to satisfy the chunks requested for this reservation or occurrence                                                    |                                   |       |                               | resv_nodes                    |

## 5.17 Viewing Resource Information

You can see attribute values of resources for the server, queues, and vnodes using the `qmgr` or `pbsnodes` commands. The value in the server, queue, or vnode `resources_assigned` attribute is the amount explicitly requested by running and exiting jobs and, at the server and vnodes, started reservations.

You can see job attribute values using the `qstat` command. The value in the job's `Resource_List` attribute is the amount explicitly requested by the job. See ["Resources Requested by Job" on page 241 in the PBS Professional Administrator's Guide](#).

The following table summarizes how to find resource information:

**Table 5-21: How to View Resource Information**

| Location       | Item to View                                                                                      | Command                                                                 |
|----------------|---------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| server         | default_chunk, default_qsub_arguments, resources_available, resources_assigned, resources_default | <a href="#">qmgr</a> , <a href="#">qstat</a> , <a href="#">pbsnodes</a> |
| scheduler      | sched_config file                                                                                 | Favorite editor or viewer                                               |
| queues         | default_chunk, resources_available, resources_assigned, resources_default                         | <a href="#">qmgr</a> , <a href="#">qstat</a>                            |
| MoM and vnodes | resources_available, sharing, pcpus, resources_assigned                                           | <a href="#">qmgr</a> , <a href="#">pbsnodes</a>                         |
|                | mom_config file                                                                                   | Favorite editor or viewer                                               |
| job            | Resource_List                                                                                     | <a href="#">qstat</a>                                                   |
| reservation    | Resource_List                                                                                     | <a href="#">pbs_rstat -f</a>                                            |
| accounting     | resources_assigned entry in accounting log                                                        | Favorite editor or viewer                                               |

Every consumable resource, for example mem, can appear in four PBS attributes. These attributes are used in the following elements of PBS:

**Table 5-22: Values Associated with Consumable Resources**

| Attribute           | Vnode | Queue | Server | Accounting Log | Job | Scheduler |
|---------------------|-------|-------|--------|----------------|-----|-----------|
| resources_available | Yes   | Yes   | Yes    |                |     | Yes       |
| resources_assigned  | Yes   | Yes   | Yes    | Yes            |     |           |
| resources_used      |       |       |        | Yes            | Yes | Yes       |
| Resource_List       |       |       |        |                | Yes | Yes       |

### 5.17.1 Resource Information in Accounting Logs

For a complete description of the resource information in the PBS accounting logs, see [Chapter 12, "Accounting", on page 529](#).

### 5.17.2 Resource Information in Daemon Logs

At the end of each job, the server logs the values in the job's resources\_used attribute, at event class 0x0010.

Upon startup, MoM logs the number of CPUs reported by the OS, at event class 0x0002.

At the end of each job, the MoM logs cput and mem used by each job, and cput used by each job task, at event class 0x0100.

### 5.17.3 Finding Current Value

You can find the current value of a resource by subtracting the amount being used from the amount that is defined.

Use the `qstat -Bf` command, and `grep` for `resources_available.<resource name>` and `resources_used.<resource name>`. To find the current amount not being used, subtract `resources_used.<resource name>` from `resources_available.<resource name>`.

### 5.17.4 Restrictions on Viewing Resources

- Dynamic resources shown in `qstat` do not display the current value, they display the most recent retrieval. Dynamic resources have no `resources_available.<resource name>` representation anywhere in PBS.

## 5.18 Resource Recommendations and Caveats

- It is not recommended to set the value for `resources_available.ncpus`. The exception is when you want to oversubscribe CPUs. See [section 8.6.6.1.iii, “How To Share CPUs”, on page 416](#).
- It is not recommended to change the value of `ncpus` at `vnodes` on a multi-`vnode`d machine.
- If you want to limit how many jobs are run, or how much of each resource is used, use the new limits. See [section 5.15, “Managing Resource Usage”, on page 283](#).
- Do not attempt to set values for `resources_available.<resource name>` for dynamic resources.
- Externally-managed application licenses may not be available when PBS thinks they are. PBS doesn't actually check out externally-managed licenses; the application being run inside the job's session does that. Between the time that the scheduler queries for licenses, and the time the application checks them out, another application may take the licenses. In addition, some applications request varying amounts of tokens during a job run.
- Jobs may be placed on different `vnodes` from those where they would have run in earlier versions of PBS. This is because a job's resource request will no longer match the same resources on the server, queues and `vnodes`.
- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue `resources_default.<resource name>` attribute.
- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue `resources_default` or `default_chunk` as an input argument.
- Resources assigned from the `default_qsub_arguments` server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from `default_qsub_arguments`. Be aware that creating custom resources with permission flags and then using these in the `default_qsub_arguments` server attribute can cause jobs to be rejected. See [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).
- Numeric server dynamic resources cannot have the `q` or `n` flags set. This would cause these resources to be underused. These resources are tracked automatically by scheduler.
- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

`pbsnodes`

Users cannot view restricted host-level custom resources.

`pbs_rstat`

Users cannot view restricted reservation resources.

`pbs_rsub`

Users cannot request restricted custom resources for reservations.

**qalter**

Users cannot alter a restricted resource.

**qmgr**

Users cannot print or list a restricted resource.

**qselect**

Users cannot specify restricted resources via `-l Resource_List`.

**qsub**

Users cannot request a restricted resource.

**qstat**

Users cannot view a restricted resource.

- Do not set values for any resources, except those such as shared scratch space or floating application licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set `resources_available.walltime` at the server to `10:00:00`, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
- If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has `resources_max.<resource name>` set, the job inherits that maximum value. If the queue has `resources_max.<resource name>` set, the job inherits the queue value, and if not, the job inherits the server value.
- When setting static vnode resources on multi-vnode machines, follow the rules in [section 3.4.5, “Configuring Vnode Resources”, on page 51](#).
- Do not create custom resources with the same names or prefixes that PBS uses when you create custom resources for specific systems.
- Do not set `resources_available.place` for a vnode.
- On the parent vnode, all values for `resources_available.<resource name>` should be **zero (0)**, unless the resource is being shared among child vnodes via indirection.
- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a **select** and a **place**. This means that if you specify a default placement, such as `excl`, with `-lplace=excl`, and the user specifies an arrangement, such as `pack`, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for `select`; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

# Configuring and Using PBS with Cgroups

## 6.1 Chapter Contents

|       |                                                                        |        |
|-------|------------------------------------------------------------------------|--------|
| 6.1   | Chapter Contents . . . . .                                             | AG-311 |
| 6.2   | Introduction to Cgroups . . . . .                                      | AG-311 |
| 6.3   | Why Use Cgroups? . . . . .                                             | AG-312 |
| 6.3.1 | What PBS Can Do With Cgroups . . . . .                                 | AG-313 |
| 6.3.2 | Examples of Using Cgroups . . . . .                                    | AG-313 |
| 6.4   | How PBS Uses Cgroups . . . . .                                         | AG-313 |
| 6.4.1 | Vnode Creation via Cgroups Hook . . . . .                              | AG-313 |
| 6.4.2 | Job Life Cycle with Cgroups . . . . .                                  | AG-314 |
| 6.4.3 | Cgroup Subsystems . . . . .                                            | AG-314 |
| 6.5   | Configuring Cgroups . . . . .                                          | AG-316 |
| 6.5.1 | Prerequisites for Cgroups Hook . . . . .                               | AG-316 |
| 6.5.2 | Enabling and Tuning Hook According to Host and/or Vnode Type . . . . . | AG-317 |
| 6.5.3 | Cgroups Hook Configuration Parameters . . . . .                        | AG-319 |
| 6.5.4 | Finish Up . . . . .                                                    | AG-344 |
| 6.5.5 | Managing GPUs or Xeon Phi via Cgroups . . . . .                        | AG-344 |
| 6.6   | Configuring MPI for Cgroups . . . . .                                  | AG-350 |
| 6.6.1 | Steps to Integrate MPI with PBS via ssh . . . . .                      | AG-351 |
| 6.7   | Managing Jobs with Cgroups . . . . .                                   | AG-352 |
| 6.7.1 | Requesting Memory . . . . .                                            | AG-352 |
| 6.7.2 | Limit Enforcement . . . . .                                            | AG-352 |
| 6.7.3 | Examples of Requesting Cores and Hyperthreads . . . . .                | AG-352 |
| 6.7.4 | Spawning Job Processes . . . . .                                       | AG-352 |
| 6.8   | Managing Oom Killer for Cgroups . . . . .                              | AG-353 |
| 6.9   | Caveats and Errors . . . . .                                           | AG-353 |
| 6.9.1 | Interactions Between Suspend/resume and the cpuset Subsystem . . . . . | AG-353 |
| 6.9.2 | Caveats for Shrinking a Job on a Host . . . . .                        | AG-354 |
| 6.9.3 | Caveats for Using CUDA . . . . .                                       | AG-354 |
| 6.9.4 | Do Not Change ncpus When cpuset Subsystem is Enabled . . . . .         | AG-354 |
| 6.9.5 | Cgroups Hook Prevents Epilogue from Running . . . . .                  | AG-354 |
| 6.9.6 | Errors . . . . .                                                       | AG-354 |

## 6.2 Introduction to Cgroups

The term cgroup (pronounced see-group, short for control groups) refers to a Linux kernel feature that was introduced in version 2.6.24.

A cgroup may be used to manage access to system resources and to account for resource usage. The root cgroup is the ancestor of all cgroups and provides access to all system resources. When a cgroup is created, it inherits the configuration of its parent. When a process assigned to a cgroup creates a child process, the child is automatically assigned to its parent's cgroup.

Once created, a cgroup may be configured to restrict access to a subset of its parent's resources. These restrictions may include such things as memory, NUMA nodes, and devices. These different resource classes are grouped into categories referred to as *cgroup subsystems*. When processes are assigned to a cgroup, the kernel enforces all configured restrictions.

In Cgroups v1, supported by this cgroups hook, the kernel supports a number of cgroup subsystems. A cgroup subsystem is a kernel component that modifies the behavior of the processes in a cgroup. Subsystems are sometimes also known as *cgroup resource controllers* or *cgroup controllers*. We describe PBS support for cgroup subsystems in this chapter.

PBS provides a hook that allows you to take advantage of cgroups. When the cgroups hook is enabled, it runs on every node assigned to the job. When a job is started, the hook creates a set of directories for the configured subsystems based on the resource requirements of the job and then places the job process within the cgroup.

While the job is running, the kernel, not PBS, enforces resource restrictions, based on the cgroup settings written by the hook earlier. The cgroups hook can be configured to periodically poll the job's cgroup and update resource usage. When the job finishes, the hook writes final resource usage to the job's `resources_used` attribute, and removes the cgroup directories it created to house the job.

## 6.3 Why Use Cgroups?

Without cgroups, Linux can define sets of processes as related, but cannot define a set of loosely coupled processes as a single entity. Without cgroups, PBS uses Linux sessions to track job processes, but less accurately than with cgroups. Linux sessions impose the following limitations:

- Restrictions on how processes have to be related: sessions must encompass a parent process and nothing else but its progeny; it is impossible to merge a job session and another session created by for example `sshd`. However, PBS can manage more than one session per host.
- Inability to set resource usage restrictions for the entire set of processes belonging to a job; while you can set per-process limits on memory or CPU time, you cannot limit a session or a group of sessions belonging to a job.
- Inability to make job sessions be inescapable containers: the `setsid` call or command will make processes leave the current session and create their own unrelated session; users and applications can use it without restriction.

With cgroups, all of these issues can be avoided:

- Since cgroups offer a well-established and general way of grouping processes together, cgroup controllers can implement precise resource usage accounting, resource usage limits, and process control for the entire cgroup. The cgroup also persists until it is explicitly destroyed, allowing some resource usage counters in a cgroup to outlive the processes that were members of the cgroup.
- Root can add processes to a cgroup regardless of the relationship between the processes, so a daemon can force processes spawned by OS services to join an existing cgroup. A process or thread spawned by a process appears at first in the cgroup of the parent process, and a non-root process cannot use any library call to escape the cgroup unless another cgroup grants that process permission to change cgroups.



### 6.3.1 What PBS Can Do With Cgroups

- More correctly identify which processes are part of a PBS job, even when libraries mislead PBS by creating processes that move into their own sessions that are not registered with PBS
- Make processes spawned through external `systemd` daemons, including `ssh`, fully join a job
- Prevent job processes from using more resources than specified; for example disallow bursting above limits set according to the resources requested by the job when it was submitted
- Keep job processes within defined memory and CPU boundaries, ensuring there is minimal interference between jobs sharing a host, in order to provide consistent job run times
- More accurately track and account for resource usage, even at the end of the job, when the job processes have exited and can no longer be investigated
- Enable or disable access to devices
- Ensure jobs leave enough resources for OS processes, to avoid disrupting OS services and turning nodes into "black hole nodes" that will no longer correctly run jobs

In addition to leveraging the kernel's cgroups support, the cgroups hook can also optionally discover the hardware configuration of a host and create child vnodes that reflect the hardware configuration, for example a number of CPU sockets with locally attached memory and GPU device.

This allows configuring the server and scheduler for optimal job performance. This functionality is an extension of the functionality formerly supported by the cpuset MoM (since the cpuset controller is now just one of the cgroup controllers in the kernel).

### 6.3.2 Examples of Using Cgroups

- Limit all of a job's processes to 6 CPUs and 6GB of RAM on vnode A
- Ensure that if two MPI jobs share a host, they do not pin processes to the same CPUs
- Limit access to GPU devices to only those assigned to the job by the cgroups hook
- Partition a host into a number of socket-aligned vnodes for optimal placement of jobs
- Pair processor, memory, and coprocessors like Xeon Phi (**deprecated**) and GPUs for optimal job placement

## 6.4 How PBS Uses Cgroups

### 6.4.1 Vnode Creation via Cgroups Hook

The hook creates a child vnode for each NUMA node on a host when all of the following conditions are true:

- The hook is enabled on the host
- At least one subsystem is enabled on the host
- The `vnode_per_numa_node` parameter is set to *true*
- The hook finds a NUMA node on the host

For example, if the host named "myhost" has one NUMA node, you end up with two vnodes to represent the host: the parent vnode, named "myhost", and a vnode to represent the NUMA node, named "myhost[0]".

### 6.4.1.1 Caveats for Vnode Creation

- Make sure that you run the cgroups hook only after you have created the parent vnode.
- If the cgroups hook creates vnodes for a host, do not use any other method to create child vnodes for that host. For example, do not create special vnodes for GPUs.

## 6.4.2 Job Life Cycle with Cgroups

### 6.4.2.1 Running Single-host Jobs with Cgroups

When PBS runs a single-host job, the following happens:

1. On the host assigned to the job, PBS creates a cgroup for each enabled subsystem. PBS assigns resources (CPUs, memory, and optionally co-processors such as GPUs or Xeon Phis (**deprecated**)) to the job on the host. It sets the required limits on resource usage in the cgroups created.
2. PBS places the top job process in each created cgroup. Cgroup semantics then automatically ensure that the progeny of the top job process are also confined to the correct cgroups.
3. The cgroup periodic hook collects resource usage information about the cgroups and hands it over to MoM; MoM uses these values when it reports job resource usage to the server to set the `resources_used` attribute.
4. When the job has finished, the cgroups hook reports CPU and memory usage to MoM and cleans up the cgroups.

### 6.4.2.2 Running Multi-host Jobs with Cgroups

When PBS runs a multi-host job, the following happens:

1. PBS creates a cgroup on each host assigned to the job. PBS assigns resources to the job and sets the required cgroup limits.
2. PBS places the top job process in the cgroup on the first node (the primary execution host). Any child processes that the job spawns on the primary execution host remain in the cgroup, even if they use calls to change Linux session.
3. The job creates processes on the remote hosts. If PBS integration of multihost applications has been done correctly, those processes will either:
  - Be created as children of MoM through a call to the TM API `tm_spawn` call (possibly indirectly through the use of `pbs_tm_rsh`)
  - Be spawned by an external service and then registered as part of the job through a call to `tm_attach` (possibly indirectly through the use of `pbs_attach`)

In both cases, the cgroups hook migrates the processes into the correct cgroups; the kernel then ensures their progeny remain in the cgroups.

4. On each host, the cgroup periodic hook collects usage information about the cgroups and hands it over to MoM; the primary execution host MoM periodically queries the resource usage on sister nodes, sums the contributions of all the nodes, and reports it to the server, which publishes it in the `resources_used` attribute.
5. When the job has finished, on each host, the cgroups hook reports final CPU and memory usage to the local MoM and cleans up the job cgroups. The primary execution host MoM collects resource usage information from all nodes, sums it across nodes, and sends it to the server, which makes a final update to the `resources_used` attribute.

## 6.4.3 Cgroup Subsystems

The cgroups hook can manage the subsystems listed in ["Subsystems Managed by the Cgroups Hook" on page 315](#).

For other subsystems, listed in ["Subsystems Used by Other Software" on page 315](#), the hook can only create and destroy per-job cgroups and move job processes into those cgroups; apart from that, it does not manage any values in those cgroups. Some of these subsystems are used by other software.

### 6.4.3.1 Cgroup Subsystems Managed by the Cgroups Hook

The cgroups hook can manage the following subsystems:

**Table 6-1: Subsystems Managed by the Cgroups Hook**

| Subsystem | Functionality Used by Cgroups Hook                                                                                                                                                                  |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cpu       | Set quota for CPU usage and/or set CPU usage shares for each job, to allow the Linux CFS scheduler to implement fair sharing<br>See <a href="#">section 6.5.3.7, "cpu Subsystem", on page 329</a> . |
| cpuacct   | Monitor CPU resource usage<br>See <a href="#">section 6.5.3.5, "cpuacct Subsystem", on page 326</a> .                                                                                               |
| cpuset    | Allocate and restrict jobs to specific CPU and optionally NUMA memory domains<br>See <a href="#">section 6.5.3.6, "cpuset Subsystem", on page 326</a> .                                             |
| devices   | Restrict job access to only specific character and block devices<br>See <a href="#">section 6.5.3.8, "devices Subsystem", on page 331</a> .                                                         |
| memory    | Set kernel-enforced per-job limits for memory usage; monitor memory usage<br>See <a href="#">section 6.5.3.9, "memory Subsystem", on page 333</a> .                                                 |
| memsw     | Set kernel-enforced per-job limits for swap usage; monitor swap usage<br>See <a href="#">section 6.5.3.10, "memsw Subsystem", on page 337</a> .                                                     |
| hugetlb   | Set kernel-enforced per-job limits for huge pages usage; monitor huge pages usage<br>See <a href="#">section 6.5.3.11, "hugetlb Subsystem", on page 340</a> .                                       |

### 6.4.3.2 Cgroup Subsystems Not Managed by Cgroups Hook

When subsystems not managed by the cgroups hook are enabled, the cgroups hook only creates per-job cgroup directories, ensures job processes are moved into them, and deletes the per-job cgroup when the job ends. Managing any parameters is left to other hooks or software.

If you run another hook, for example the container hook, that expects per-job cgroups to have been created by the cgroups hook for a set of subsystems, make sure you enable these subsystems on the host.

The cgroups hook can create and destroy cgroups for, and move processes into, the following subsystems:

**Table 6-2: Subsystems Used by Other Software**

| Subsystem | Functionality Used by Other Software                                   |
|-----------|------------------------------------------------------------------------|
| freezer   | Can be used to suspend entire job                                      |
| blkio     | Can track and limit usage and bandwidth to specific disk block devices |
| pids      | Can be used to track and limit number of processes in a job            |

Table 6-2: Subsystems Used by Other Software

| Subsystem   | Functionality Used by Other Software                                                                                                    |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| net_cls     | Can be used to tag network packets sent by the job with a job-specific class identifier that can be used in e.g. firewall configuration |
| net_prio    | Can be used to dynamically set the priority of network traffic generated by each job                                                    |
| perf_events | Allow the <code>perf</code> tool to monitor a PBS Professional job as a group                                                           |

## 6.5 Configuring Cgroups

You manage the behavior of the cgroups hook across your complex by setting parameters in the cgroups hook configuration file.

### 6.5.1 Prerequisites for Cgroups Hook

#### 6.5.1.1 Ensure that Cgroups v1 are Available

Many Linux distributions have cgroups v1 available by default; for others you may need to install and enable cgroups. We provide some tips here for making sure that cgroups are available on your system:

- Verify that you have cgroups configured on your system:

```
cat /proc/mounts | grep cgroup
```

You should see `cpuset`, `cpuacct`, `memory`, etc. enabled.

Each subsystem that will be enabled on the host by the configuration file should be listed (except `memsw`; see below).

If you do not have cgroups available, install them.

You may need to set your kernel flags so that they enable cgroup support.

- A cgroup subsystem may be disabled at boot time. To check for this, look for "cgroup\_disable" entries in `/proc/cmdline` and take appropriate actions to remove it from the kernel command line parameters.
- The `memsw` subsystem is not a separate controller but an option of the `memory` controller that is present or absent depending on the kernel options at boot time.

If you plan to enable the `memsw` subsystem, to verify whether swap accounting and limits are available:

```
cat /proc/mounts | grep cgroup | grep memory | awk 'BEGIN {FS=" "} {print $2}' | head -n 1 | xargs
--replace=dir ls dir/memory.memsw.usage_in_bytes
```

If this lists a file, then memory plus swap accounting is turned on. If instead, `ls` reports it cannot access the file, then kernel support for it is disabled; in some kernels it is disabled by default.

If `memsw` support is disabled, either add "swapaccount=1" to the kernel command line parameters to enable it and reboot the node, or ensure that the `memsw` section of the configuration file is disabled for the host.

- Make sure that cgroups will survive a reboot. Test whether cgroups survive by rebooting the host, then look to see whether cgroups are available. If they are not available, refer to the documentation for your Linux distribution.

#### 6.5.1.2 Ensure that PBS Is Already Installed and Started

Make sure that PBS is installed and started.

## 6.5.2 Enabling and Tuning Hook According to Host and/or Vnode Type

You can use just one configuration file across a complex containing hosts with different configurations. You can enable the hook for a specific subset of the hosts in your complex. You can also tune the hook by enabling each subsystem independently according to host. For example if you have some hosts with swap, and some without, and some hosts with GPUs and some without, you can enable the `memsw` subsystem only for the hosts that have swap, and enable the `devices` subsystem only for the hosts that have GPUs.

You can similarly tune the hook for any parameter that takes *true* and *false*. So for example you can tune the `soft_limit` parameter in the `memory` subsystem so that `soft_limit` evaluates to *true* for certain hosts.

### 6.5.2.1 Vnode Types for Cgroups Hook

You can label each host to reflect its characteristics, then use the label when specifying which hosts are included in a subset.

The labeling mechanism is a single string in a file. If `PBS_MOM_HOME` is defined, the string is in a file named `PBS_MOM_HOME/mom_priv/vntype` on MoM's host, and if not, it is in a file on MoM's host in `PBS_HOME/mom_priv/vntype`.

We refer to this string as a "*vnode type*", and the hook stores the value of the string in the variable "*vntype*". You can define any vnode type you need. Write your vntypes using only alphanumerical characters and the delimiters ".", "-", and "\_".

#### 6.5.2.1.i Vnode Type File and vntype Resource

The `resources_available.vntype` vnode resource and the `vntype` file contents are related but different tools. Do not try to set the `vntype` file contents by changing `resources_available.vntype` for the vnode (this will not work). If you want the value of `resources_available.vntype` for the vnode to reflect the contents of the `vntype` file on the execution host, you can propagate the file string to the resource by setting the `propagate_vntype_to_server` parameter in the hook's configuration file to *True*.

### 6.5.2.2 Tuning Where Hook, Subsystems, and Parameters are Enabled

For each of the *true/false* parameters, and for the `swappiness` parameter, you can specify whether a host or vnode type is in, or not in, the list for which the parameter evaluates to *true*. A list is one or more comma-separated host or vnode names, specified using one of these:

"vntype in:"

"vntype not in:"

"host in:"

"host not in:"

Whitespace around the entries is ignored. You can use hostnames or vntypes, or Python `fnmatch` sequences, which allows "\*" or "?" wildcards. Do not use commas inside an entry.

Example 6-1: If you have four vntypes "compute\_swap", "compute\_noswap", "gpu\_swap", "gpu\_noswap", you can set the `swappiness` parameter for the `memory` subsystem using

```
"swappiness" : "vntype in: *_swap"
```

and set the `vnode_per_numa_node` parameter in the main section using:

```
"vnode_per_numa_node" : "vntype in: gpu_*"
```

### 6.5.2.2.i Enabling the Hook and Subsystems

For the hook and each subsystem, the `enabled` parameter can be modified using the `exclude_vntypes`, `exclude_hosts`, `include_hosts`, and `run_only_on_hosts` parameters. In the following hierarchy, each parameter modifies the previous parameters. Always specify these parameters in this order in the configuration file:

1. `enabled`
2. `exclude_hosts`
3. `exclude_vntypes`
4. `include_hosts`
5. `run_only_on_hosts`

#### 6.5.2.2.ii `exclude_vntypes`

Modifies the `enabled` parameter. JSON list of patterns for `vntypes` to exclude from enabled group. For example, to include all hosts except those without `cgroups` (marked with `"no_cgroup"` in their `vntype`):

```
"enabled" : true,
"exclude_vntypes": ["*no_cgroup*"]
```

This is equivalent to:

```
"enabled" : "vntype not in: *no_cgroup*"
```

#### 6.5.2.2.iii `exclude_hosts`

Modifies the `enabled` parameter. List of hosts to exclude from membership group. For example, to enable all hosts with GPUs (marked with `"gpu"` in their `vntype`), except for the hosts marked for testing:

```
"enabled" : "vntype in: gpu"
"exclude_hosts" : ["gpu_test*"]
```

#### 6.5.2.2.iv `include_hosts`

Modifies and overrides the `enabled`, `exclude_vntypes`, and `exclude_hosts` parameters. List of hosts to include in membership group, despite having been among those excluded. For example, to include two `"thin"` hosts in the `cpuset` subsystem list, despite the fact that they did not qualify to be enabled in the `"fat"` group:

```
"cpuset" : {
 "enabled" : "vntype in: fat"
 "include_hosts" : ["test_cpuset_thin01", "test_cpuset_thin02"]
}
```

This is equivalent to:

```
"cpuset" : {
 "enabled" : "vntype not in: thin"
 "include_hosts" : ["test_cpuset_thin01", "test_cpuset_thin02"]
}
```

#### 6.5.2.2.v `run_only_on_hosts`

Modifies the `enabled`, `exclude_vntypes`, `exclude_hosts`, and `include_hosts` parameters. Provides additional restriction on hosts otherwise qualified for inclusion in membership list. Any host included must pass all membership tests. For example, to include only hosts that are both `"willing"` and among the list of `"able01"`, `"able02"`, and `"able03"`:

```
"enabled" : "vntype in: willing"
"run_only_on_hosts" : ["able01", "able02", "able03"]
```

### 6.5.2.2.vi Hook and Subsystem Enablement Tuning Parameters

The following parameters let you tune whether the hook and each subsystem is enabled on any host and/or vntype. You can set each parameter in this table for the entire hook and for each subsystem individually. Here we show the defaults in the configuration file, and in the code, as they apply to whether the hook itself is enabled. We do not list the defaults for each subsystem; they may be different.

**Table 6-3: Cgroups Hook Global and Subsystem Membership Configuration Parameters**

| Parameter Name    | Default Value: Configuration File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                              |
|-------------------|-----------------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled           | <i>true</i>                       | <i>True</i>         | When <i>true</i> , the cgroups hook is enabled on the host.<br>Can be modified using the <code>exclude_vntypes</code> , <code>exclude_hosts</code> , <code>include_hosts</code> , and <code>run_only_on_hosts</code> parameters.<br>See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”, on page 317</a> .     |
| exclude_hosts     | <code>[]</code>                   | <code>[]</code>     | Modifies the <code>enabled</code> parameter.<br>If not empty, specifies a list of hosts where the hook should be disabled.<br>See <a href="#">section 6.5.2.2.iii, “exclude_hosts”, on page 318</a>                                                                                                                                                      |
| exclude_vntypes   | <code>["no_cgroups"]</code>       | <code>[]</code>     | Modifies the <code>enabled</code> parameter.<br>Specifies a list of vnode types for which the cgroups hook should be disabled.<br>See <a href="#">section 6.5.2.2.ii, “exclude_vntypes”, on page 318</a>                                                                                                                                                 |
| include_hosts     |                                   |                     | Modifies the <code>enabled</code> and <code>exclude_hosts</code> parameters.<br>If not empty, specifies a list of hosts where the hook should be enabled despite earlier exclusion.<br>See <a href="#">section 6.5.2.2.iv, “include_hosts”, on page 318</a>                                                                                              |
| run_only_on_hosts | <code>[]</code>                   | <code>[]</code>     | Overrides the <code>enabled</code> , <code>exclude_hosts</code> , <code>exclude_vntypes</code> , and <code>include_hosts</code> parameters.<br>If not empty, specifies a list of hosts limiting the hosts for which the cgroups hook should be enabled to the matching hosts.<br>See <a href="#">section 6.5.2.2.v, “run_only_on_hosts”, on page 318</a> |

## 6.5.3 Cgroups Hook Configuration Parameters

The cgroups hook configuration file contains parameters that the hook uses as guides for its behavior, and parameters that the hook uses when it sets values in the cgroups directories. For any parameter that is unset in the configuration file, the cgroups hook uses defaults built into the hook. This file must conform to JSON syntax.

The cgroups hook configuration file is named *pbs\_cgroups.json* before it is imported as the hook configuration file. After the file is imported as the hook configuration file, PBS names it *pbs\_cgroups.CF*.



### 6.5.3.1 Global Parameters for Cgroups Hook

Here are the cgroups hook configuration parameters, except for the membership tuning parameters described above. Please note that some parameters may be different from those shown here:

**Table 6-4: Cgroups Hook Configuration File Global Parameters**

| Parameter Name        | Default Value: Config File | Default Value: Hook                           | Description                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|----------------------------|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled               | <i>true</i>                | <i>True</i>                                   | When <i>true</i> , the cgroups hook is enabled on the host.<br>Can be modified using the <code>exclude_vntypes</code> , <code>exclude_hosts</code> , <code>include_hosts</code> , and <code>run_only_on_hosts</code> parameters. See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”</a> , on page 317. |
| cgroup_lock_file      |                            | <i>"/var/spool/pbs/mom_priv/cgroups.lock"</i> | This file ensures that only one hook event can manipulate the cgroups at any one time. The filesystem on which this file resides must support file locking.                                                                                                                                                                                       |
| cgroup_prefix         | <i>"pbs_jobs"</i>          | <i>"pbs_jobs"</i>                             | The parent directory under each cgroup subsystem where job cgroups are created. If the memory subsystem is located at <code>/sys/fs/cgroup/memory</code> , the memory cgroup for job 1.foo is found in the <code>/sys/fs/cgroup/memory/pbs_jobs.service/&lt;job ID&gt;/1.foo</code> directory.                                                    |
| kill_timeout          | <i>10</i>                  | <i>10</i>                                     | Maximum number of seconds the cgroups hook spends attempting to kill job processes before destroying cgroups                                                                                                                                                                                                                                      |
| server_timeout        | <i>15</i>                  | <i>15</i>                                     | Maximum number of seconds the cgroups hook spends attempting to fetch node comments from the server                                                                                                                                                                                                                                               |
| nvidia-smi            |                            | <i>"usr/bin/nvidia-smi"</i>                   | The location of the <code>nvidia-smi</code> command on nodes supporting NVIDIA GPU devices.<br>See <a href="#">section 6.5.5, “Managing GPUs or Xeon Phi via Cgroups”</a> , on page 344.                                                                                                                                                          |
| online_offlined_nodes | <i>true</i>                | <i>false</i>                                  | When enabled, if the periodic hook manages to confirm there are no orphan groups, it will online vnodes again if it can confirm they were earlier offlined by the cgroups hook.<br>See <a href="#">section 6.5.3.4, “Automatic Onlining of Fixed Vnodes”</a> , on page 325.                                                                       |
| periodic_resc_update  | <i>true</i>                | <i>false</i>                                  | When set to <i>true</i> , the hook periodically posts updates of the job's resource usage on this host to MoM. When set to <i>false</i> , the usage is sent to MoM only when the job ends                                                                                                                                                         |
| use_hyperthreads      | <i>false</i>               | <i>false</i>                                  | When set to <i>true</i> , all CPU threads are made available to jobs. When <i>false</i> , only the first hyperthread of each core is made visible to jobs.<br>See <a href="#">section 6.5.3.3, “Configuring Hyperthreading Support”</a> , on page 323.                                                                                            |

Table 6-4: Cgroups Hook Configuration File Global Parameters

| Parameter Name             | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------|----------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ncpus_are_cores            | <i>false</i>               | <i>"false"</i>      | <p>When <i>true</i>, <code>resources_available.ncpus</code> of a vnode is the number of cores, and the hook assigns all threads of each core to a job.</p> <p>When <i>false</i>, <code>resources_available.ncpus</code> is the number of CPU threads available to jobs, and the hook assigns individual CPU threads to jobs.</p> <p>See <a href="#">section 6.5.3.3, “Configuring Hyperthreading Support”</a>, on page 323.</p>         |
| vnode_per_numa_node        | <i>false</i>               | <i>"false"</i>      | <p>When set to <i>true</i>, each NUMA node is represented by a separate vnode, and the host is managed by a resourceless parent vnode.</p> <p>When set to <i>false</i>, the entire host is represented by a single vnode.</p> <p>See <a href="#">section 6.5.3.2, “Setting vnode_per_numa_node”</a>, on page 323.</p>                                                                                                                   |
| propagate_vntype_to_server |                            | <i>"true"</i>       | <p>When set to <i>true</i>, the contents of the <code>vntype</code> file on the local host are propagated to the <code>resources_available.vntype</code> vnode resource.</p> <p>When set to <i>false</i>, the <code>vntype</code> file contents are not propagated to the vnode <code>resources_available.vntype</code> resource.</p> <p>See <a href="#">section 6.5.2.1.i, “Vnode Type File and vntype Resource”</a>, on page 317.</p> |

Table 6-4: Cgroups Hook Configuration File Global Parameters

| Parameter Name    | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|----------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| manage_rlimit_as  | <i>true</i>                | <i>"true"</i>       | <p>When set to <i>true</i>, the cgroups hook resets the RLIMIT_AS process limit for task processes to the value of <code>pvmem</code> requested for the job, or to unlimited if <code>pvmem</code> is not requested.</p> <p>Requires a kernel that supports the <code>prlimit</code> system call.</p> <p>When set to <i>false</i>, limits set by MoM are not changed.</p> <p>Set this to <i>true</i> when you enable the <code>memsw</code> subsystem. Otherwise you can set it to <i>false</i>. When it is <i>false</i>, MoM sets RLIMIT_AS to <code>Resource_List.pvmem</code> if specified, otherwise <code>Resource_List.vmem</code>.</p> |
| ngpus_ext_managed | <i>false</i>               | <i>"false"</i>      | <p>Allows you to manage <code>resources_available.ngpus</code> using something other than the cgroups hook.</p> <p>When set to <i>true</i>:</p> <ul style="list-style-type: none"> <li>The <code>discover_gpus</code> parameter is disabled</li> <li>The hook never sets <code>resources_available.ngpus</code> on the vnodes of the host</li> <li>The hook does not attempt to assign individual GPUs to jobs; the hook will allow a job on a node regardless of the <code>ngpus</code> resources requested by the job</li> </ul> <p>See <a href="#">section 6.5.5.6, “Not Using Cgroups to Manage GPUs”</a>, on page 349.</p>               |
| discover_gpus     | <i>true</i>                | <i>"true"</i>       | <p>Enables or disables call to <code>nvidia-smi</code>. When set to <i>true</i>, <code>nvidia-smi</code> is called. When set to <i>false</i>, speeds the process of device discovery when the <code>devices</code> subsystem is enabled on a GPU-less host.</p> <p>Disabled when <code>ngpus_ext_managed</code> parameter set to <i>true</i>.</p>                                                                                                                                                                                                                                                                                             |

We show a sample file in [section 6.5.3.12, “Sample Cgroups Hook Configuration File”](#), on page 342. You can also export and look at the installed PBS cgroups hook configuration file:

```
qmgr -c "export hook pbs_cgroups application/x-config default" >pbs_cgroups.json
```

You can edit this file and change its parameters, then read it back in:

```
qmgr -c "import hook pbs_cgroups application/x-config default pbs_cgroups.json"
```

Note that if your configuration file is incomplete or not present, hook behavior may differ from what is expected.

### 6.5.3.2 Setting `vnode_per_numa_node`

When this is *false*, all resources of the host are presented to the server as a single vnode (the parent vnode). In a large complex, minimizing the number of vnodes makes it faster for the scheduler to select nodes for jobs, and if large parallel jobs span a set of small hosts used exclusively by one job at a time, there is little advantage in making subdivisions of the host visible to the server and scheduler.

However, on clusters where execution hosts run more than one job at a time, you can take advantage of hosts made up of a number of separate NUMA nodes. (A NUMA node is a set of CPUs with uniform access speed and latency to a set of local memory and PCIe resources. Usually a NUMA node maps to a socket in a multi-socket computer, but some processors integrate more than one NUMA node on a single socket. On both AMD and Intel processors, the number of NUMA nodes per socket also depends on BIOS configuration, which allows tuning the size of a NUMA node to the workload. The cgroups hook does not decide how many NUMA nodes there are; it relies on the Linux kernel's view of the NUMA nodes on a host.)

When this parameter is *true*, the scheduler is able to improve application performance in these ways:

- Run small jobs only on single-NUMA-node vnodes
- Give parallel applications smaller than a host exclusive use of their NUMA nodes
- Run jobs that request GPUs on vnodes where the CPUs are on the socket directly connected to the GPU's PCIe bus

When `vnode_per_numa_node` is *true*, the host is presented to the server as a parent vnode that has no resources, plus a number of child vnodes that are aligned to NUMA nodes and hold specific CPU, memory, and coprocessor resources such as GPUs or Intel Xeon Phi (**deprecated**) processors.

The scheduler can still spread a single chunk across several vnodes on the same host. To ensure that a job is placed on only one NUMA node, use `-lplace=group=vnode`. You can also group using custom resources that identify larger sets of well-connected vnodes.

The main drawback to enabling `vnode_per_numa_node` is the increase in the number of vnodes in a cluster, which may slow down the scheduler, and make the output of `pbsnodes` larger and more difficult to interpret. A second drawback is that certain classes of jobs will no longer fit in one vnode, making it more complex to ensure they are still placed in a well-connected set of vnodes.

### 6.5.3.3 Configuring Hyperthreading Support

Hyperthreading can increase the throughput for some applications; it can also allow the operating system to retain access to some idle CPU threads even when PBS jobs use every core. If you disable hyperthreading in the BIOS, the kernel must share the only visible thread in each core with PBS jobs. On the other hand,

- Hyperthreading makes it more complicated to run applications that get no performance benefits from it, especially on clusters where some nodes are hyperthread-enabled and others are not
- On a host that runs more than one job, for parallel applications, hyperthreading that is not tightly managed can have severe negative impacts on performance if the threads of a single core are running processes from unrelated applications

This is why the cgroups hook supports three different models of hyperthreading support:

#### **"No hyperthreads" behavior**

`use_hyperthreads` disabled

In this model PBS makes only the first thread of each core visible to PBS jobs, so if your workload cannot leverage hyperthreading well, you don't need to disable hyperthreading in the BIOS. The other CPU threads are still usable by the operating system, which means throughput is better than if hyperthreading support is disabled in the BIOS.

The value of `resources_available.ncpus` reflects the number of cores associated with a vnode, minus the cores whose threads have been marked as reserved by using `exclude_cpus` in the `cpuset` section of the configuration file.

This model is different from ["Assign whole cores to jobs" behavior](#) only when the `cpuset` subsystem is enabled.

**"Default behavior"**

`use_hyperthreads` enabled and `ncpus_are_cores` disabled

This mode mimics the behavior you would get without the cgroups hook, as well as the behavior of the former cpuset MoM.

The value of `resources_available.ncpus` reflects the number of CPU threads available on a vnode. For applications to request all threads of  $N$  cores, they must request  $2*N$  `ncpus` on 2-way hyperthreaded hosts, or  $N$  `ncpus` on hosts where hyperthreading is disabled. The cgroups hook tries to allocate those threads from the minimum number of cores.

The main use case of this mode is a workload consisting of many single-threaded jobs for which running one job per thread rather than one job per core improves throughput. For example, on a host with a total of 24 2-way hyper-threaded cores, you can run 48 unrelated jobs instead of 24. The 48 jobs will run slower than if you ran only 24, but the total throughput might still be greater than if you had run 24 jobs.

**"Assign whole cores to jobs" behavior**

`use_hyperthreads` enabled and `ncpus_are_cores` enabled

In this model, hyperthreads are exposed to applications within a job, but one CPU of `ncpus` maps to all the threads of a single core. The cgroups hook assigns each CPU core exclusively to one job, but within a job the processes see all CPU threads of the assigned cores and can either choose to ignore hyperthreading or leverage it.

This is the easiest model to use when you do not need to map more than one job on a single core to increase throughput.

The value of `resources_available.ncpus` reflects the number of cores associated with a vnode, minus the cores whose threads have been marked as reserved by using `exclude_cpus` in the `cpuset` section of the configuration file.

This model is often preferred.

This model is different from ["No hyperthreads" behavior](#) only when the `cpuset` subsystem is enabled.

**6.5.3.3.i Mixing Hyperthreading Models in a Complex**

If you have a mixed workload or complex where you want to run some high-throughput single-threaded jobs, and some that take advantage of hyperthreading:

- You can tune your hook configuration file in order to partition your complex, where:
  - Some hosts are configured with `ncpus_are_cores` disabled, to run high-throughput single-threaded workloads
  - Other hosts are configured with `ncpus_are_cores` enabled
- You can allow jobs to request hyperthreaded hosts based on whether or not `ncpus_are_cores` is enabled

You can include hyperthreading information in each host's vnode type string in its `vntype` file, so that the cgroups hook parameters `use_hyperthreads` and `ncpus_are_cores` evaluate correctly to *true* or *false* for each host. You can set a resource on each host to indicate how hyperthreads are handled on that host, so that jobs can request the hyperthreading they want. You can also use this resource to associate hosts with queues, so that job submitters can specify a queue instead of requesting a resource. The string in the `vntype` file is probably going to be used for multiple characteristics such as GPUs, swap, etc., so you probably don't want to propagate it to `resources_available.vntype`.

Example 6-2: You use the custom resource `ht` to indicate whether and what kind of hyperthreads are available, and you use `"ht"` or `"nohyper"` in the `vntype` file to indicate whether hyperthreads exist. You can associate queues with hosts according to host configuration. If you have some hosts with hyperthreading and some without, some hosts with all threads of a core assigned to one job and some hosts without, and some hosts with GPUs and some without, you might end up with the following:

**Table 6-5: Example of Mixing Hyperthreading Models**

| Host  | use_hyperthreads | ncpus_are_cores | vntype file     | resources_available.ht | queue name  |
|-------|------------------|-----------------|-----------------|------------------------|-------------|
| hosta | <i>true</i>      | <i>true</i>     | gpu_ht          | ht_by_core             | ht_core_q   |
| hostb | <i>true</i>      | <i>true</i>     | compute_ht      | ht_by_core             | ht_core_q   |
| hoste | <i>true</i>      | <i>false</i>    | gpu_ht          | ht_by_thread           | ht_thread_q |
| hostf | <i>true</i>      | <i>false</i>    | compute_ht      | ht_by_thread           | ht_thread_q |
| hostc | <i>false</i>     | <i>true</i>     | gpu_nohyper     | core                   | core_q      |
| hostd | <i>false</i>     | <i>true</i>     | compute_nohyper | core                   | core_q      |
| hostg | <i>false</i>     | <i>false</i>    | gpu_nohyper     | core                   | core_q      |
| hosth | <i>false</i>     | <i>false</i>    | compute_nohyper | core                   | core_q      |

In our example, none of the job submitters care about whether their jobs run on hosts with GPUs; the GPU machines are here just to illustrate how you might use the `vntype` file string for more than one aspect of a host.

Jobs that want hyperthreads and want all of the threads for each core can request `"ht=ht_by_core"` in the select statement, or they can request or be routed to the queue named `"ht_core_q"`.

Single-threaded jobs that are I/O bound and don't mind sharing a core can request `"ht=ht_by_thread"`, or they can request or be routed to the queue named `"ht_thread_q"`.

Single-threaded jobs that want non-hyperthreaded cores can request `"ht=core"`, or they can request or be routed to the queue named `"core_q"`.

### 6.5.3.4 Automatic Onlining of Fixed Vnodes

When cleaning up a job, if the cgroups hook fails to kill all processes within a cgroup, it cannot destroy the job's cgroups. If that happens, it offlines the vnodes on the host to prevent the scheduler from sending new jobs to the vnodes; since `resources_assigned` is no longer accurate for the vnodes, the cgroups hook might reject the jobs. The cgroups hook then periodically attempts to clean up these orphaned cgroups.

When the `online_offlined_nodes` parameter is enabled, the hook automatically online the vnodes once no more orphaned cgroups exist, if the node comment confirms that the cgroups hook offlined the vnodes. When this parameter is disabled, it leaves the node offline; you can manually online vnodes again later after confirming the host is healthy.

### 6.5.3.5 **cpuacct Subsystem**

The **cpuacct** subsystem enables per-job CPU time accounting, through per-cgroup usage counters provided by the kernel. Advantages:

- The kernel maintains per-cgroup usage counters, so MoM doesn't have to sum the usage of each job process
- MPI libraries or applications that use `setsid` to detach processes from existing sessions do not cause inaccurate resource usage accounting, since the processes are still seen as part of the job
- A session spawning a child session that registers itself to PBS (not necessary with cgroups) does not cause some CPU time to be counted twice incorrectly at the end of the job
- Usage accounting does not rely on MoM polling, and usage accumulated after the last poll cycle is still counted. (When the **cpuacct** subsystem is not used, CPU time for `tm_attached` processes is counted only until the last MoM poll cycle)
- Short MoM polling cycles are not required for accurate accounting at the end of the job

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 319](#).

### 6.5.3.6 **cpuset Subsystem**

The **cpuset** subsystem restricts jobs to specific CPUs and optionally memory sockets allocated to them by the hook.

Advantages:

- Libraries know which set of CPUs are available for process pinning, so when a vnode is shared between multiple jobs, libraries don't try to pin more than one job to the same CPUs. For example, if you have two 4-CPU jobs and an 8-CPU vnode, they won't both try to pin themselves on CPUs 0, 1, 2, and 3.
- Strict job isolation is enforced; it is impossible for a job to steal CPU resources from another job on the same host, since the Linux scheduler will only run processes on the designated CPUs. If a job requests N CPUs and then creates N\*2 processes, the job's processes will compete with each other for CPU resources, instead of disturbing other jobs.
- Since job processes are restricted to specific CPUs and not left to wander over the entire host, the default "first touch node local" memory allocation policy can minimize memory latency; for jobs that do not span NUMA nodes, pinning processes to CPUs to avoid non-local memory accesses is no longer even necessary.

Disadvantages: if you want to overcommit CPU resources, for example by setting `resources_available.ncpus` to 128 on a host that has 64 CPU threads, you cannot use the **cpuset** subsystem.

Note that there can be a problem when using the **cpuset** subsystem and preemption via suspend and resume. See [section 6.9.1, “Interactions Between Suspend/resume and the cpuset Subsystem”, on page 353](#).

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 319](#).

#### 6.5.3.6.i **Using Memory Fences for Job Memory Requests**

You can set memory fences around jobs by setting the `mem_fences` parameter to *true*. When *true*, the cgroups hook sets the `cpuset.mems` to only the NUMA nodes assigned to the job, preventing jobs from using memory from other NUMA nodes. When *false*, `cpuset.mems` encompasses all NUMA nodes present on the host. This parameter is *false* by default.

Using job memory fences maximizes application performance, and mimics the behavior of the former **cpuset** MoM most closely.

Problems can arise when `mem_fences` is set to *true* but some jobs can still straddle nodes and grab memory on a node where a fenced job runs. In this case the straddling job can cause the fenced job not to have enough memory available, and the fenced job can die.



Recommendations when using memory fences for job requests:

- Do not enable `mem_fences` when `vnode_per_numa_node` is disabled, unless all jobs request less memory per `ncpus` requested than the average available per `ncpus`. When `vnode_per_numa_node` is disabled, the scheduler will not track to which sockets memory is assigned for existing jobs, and cgroup memory limits also do not allow discerning to what vnodes memory was assigned for existing jobs. If fences are enabled, that can lead to jobs getting killed when jobs fenced on one NUMA node require more than is available there.
- Precisely match job chunk specifications to vnodes, and/or run only a combination of jobs that use `-lplace=group=vnode` for small jobs and `-lplace=excl` for larger jobs. Do not allow the scheduler to split chunks across several vnodes. If a job requests `-lncpus=1:mem=2GB` and the `ncpus` are allocated on vnode `node[0]` but the memory is partially allocated on vnode `node[1]`, memory fences are not going to enforce node local memory allocations. Memory fences can cause jobs to fail if some jobs straddle vnodes and share vnodes with jobs entirely confined to one vnode. For example, if job B is allocated 1GB of memory on node `node[0]` and 63GB of memory on `node[1]`, the kernel may let the job allocate 64GB on `node[0]` instead, causing memory allocations for jobs confined to `node[0]` to fail through no fault of their own.
- When erecting memory fences and using huge pages, install utilities to ensure the correct amount of huge pages are present on NUMA nodes allocated to jobs; if the huge pages still available are on the wrong nodes, jobs may fail.

Without memory fences, processes in a cpuset still have a strong preference to allocate memory on the NUMA node of the first process to access the memory, unless memory on that NUMA node is depleted, when the kernel will satisfy requests using memory on other NUMA nodes.

Disabling memory fences is the safest option to ensure that jobs do not fail because they cannot request enough memory. This can happen when another job, possibly one using more than one vnode, grabs the memory first.

The drawback of not having memory fences is that when unanticipated off-node allocations do happen, a job will not fail but will silently use remote memory and run slower; you may prefer these jobs to fail, so that you can address the root cause rather than just run jobs too slowly, especially for large parallel jobs that should never cause such off-node allocations.

If you wish to rely on first touch local node memory placement to work most of the time and would rather see jobs still run rather than fail when remote memory allocations become inevitable, disable memory fences.

Recommendations when not using memory fences for job requests:

- If you disable memory fences, you must appropriately set kernel tunables governing how the OS uses memory for caching files; see [section 6.5.3.6.iii, “Memory Spreading for OS File Caching”, on page 328](#).
- When using large memory pages (see [section 6.5.3.11, “hugetlb Subsystem”, on page 340](#)) you must ensure that enough large pages are always available on the NUMA nodes where they are needed to ensure best performance.

### 6.5.3.6.ii Using Memory Fences for OS File Caching

Pages allocated by the kernel to cache files accessed by processes have their own fences, controlled by the `mem_hardwall` parameter. In theory, placing these on remote NUMA nodes has less deleterious effects, both because memory latency to these pages is less important, and also because a job on foreign NUMA nodes can in theory reuse these pages fairly easily, since the kernel can discard the cached contents of the file to mark the memory free again (but possibly only after writing out dirty cache to disk, which may take a while).

By default `mem_hardwall` is set to *false*, which allows the operating system the freedom to use memory on all nodes to cache files for any process. If you want to isolate jobs to get more repeatable performance, you can enable the `mem_hardwall` parameter, so that a cgroup can cache files using pages on only the NUMA nodes assigned to it.

### 6.5.3.6.iii Memory Spreading for OS File Caching

By default `memory_spread_page` is disabled for a `cpuset`, which means that file cache allocated for a process in the `cgroup` will preferentially be allocated on the NUMA node where the process that causes the file to be cached is currently running. This minimizes I/O latency to the buffer cache, but may create hotspots on certain NUMA nodes where file cache is concentrated, reducing the free memory immediately available for application memory allocations. If the kernel cannot reuse the file cache rapidly enough, the concentration of file cache may cause memory to be allocated off-node, which can slow application performance.

You can enable the `memory_spread_page` parameter to reduce file cache hotspots by spreading file cache allocation across the NUMA nodes that can be used by the `cgroup`. This may cause a slight increase in latency for accessing cached files.

To control whether pages are spread only on the NUMA nodes assigned to the job, or on the entire host, use the `mem_hardwall` parameter. Set this to *true* to spread pages only on the NUMA nodes assigned to the job.

### 6.5.3.6.iv Allowing Zero CPU Jobs

Some job submitters may want to run "weightless" jobs that consume few CPU resources; these jobs are assigned zero CPUs. The `cpuset` subsystem allows these jobs to run in the parent `cpuset` that has access to all CPUs and memories. However, these processes break down the barriers that otherwise ensure jobs are isolated from other jobs, so you may want to disable support for these by setting the `allow_zero_cpus` parameter in the `cpuset` subsystem to *false*.

If you want to allow zero-CPU jobs while ensuring that they don't take up too much of your resources, enable the `cpu` subsystem.

### 6.5.3.6.v Excluding CPUs

You can exclude CPUs so that they are not used by jobs by listing them in the `exclude_cpus` parameter. If the `cpuset` subsystem is enabled, the CPUs you specify in `exclude_cpus` are not assigned to jobs. Note, however, that if the `cpuset` subsystem is disabled, CPUs are still excluded from jobs, but only by reducing the count of CPUs available; you cannot control which CPUs are excluded.

6.5.3.6.vi **cpuset Subsystem Configuration Parameters****Table 6-6: cpuset Subsystem Configuration Parameters**

| Parameter Name     | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled            | <i>true</i>                | <i>false</i>        | <p>When set to <i>true</i>, the hook creates a cpuset for each job, taking into account the number of <code>nproc</code> and <code>mem</code> resources assigned by the scheduler to this host's vnodes.</p> <p>When set to <i>false</i>, the kernel is free to schedule processes and allocate memory based on the system configured policies.</p> <p>See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”, on page 317</a>.</p> |
| exclude_cpus       | <code>[]</code>            | <code>[]</code>     | <p>Specifies CPU thread IDs not to be assigned to jobs</p> <p>Format: JSON list</p> <p>Default: Empty list, no CPU thread IDs excluded</p> <p>See <a href="#">section 6.5.3.6.v, “Excluding CPUs”, on page 328</a></p>                                                                                                                                                                                                                                                     |
| mem_fences         | <i>false</i>               | <i>false</i>        | <p>When <i>true</i>, the cgroups hook sets the <code>cpuset.mems</code> to only the NUMA nodes assigned to the job, preventing other NUMA nodes from satisfying memory requests from the job.</p> <p>When <i>false</i>, <code>cpuset.mems</code> encompasses all NUMA nodes present on the host.</p> <p>See <a href="#">section 6.5.3.6.i, “Using Memory Fences for Job Memory Requests”, on page 326</a></p>                                                              |
| mem_hardwall       | <i>false</i>               | <i>false</i>        | <p>Specifies whether kernel allocations for file caching should be restricted to the memory nodes in the cpuset. By default, all NUMA nodes can be used for caching files accessed by job processes. When set to <i>true</i>, the buffer cache for this job will be constrained to the NUMA nodes listed in <code>cpuset.mems</code> for the job.</p> <p>See <a href="#">section 6.5.3.6.ii, “Using Memory Fences for OS File Caching”, on page 327</a></p>                |
| memory_spread_page | <i>false</i>               | <i>false</i>        | <p>Specifies whether file system buffers should be spread evenly across the memory nodes allocated to the cpuset. By default, no attempt is made to spread memory pages for these buffers evenly, and buffers are placed on the same node on which the process that created them is running.</p> <p>See <a href="#">section 6.5.3.6.iii, “Memory Spreading for OS File Caching”, on page 328</a></p>                                                                       |
| allow_zero_cpus    |                            | <i>true</i>         | <p>Specifies whether zero CPU jobs should be allowed to run or refused when the cpuset subsystem is enabled.</p> <p>See <a href="#">section 6.5.3.6.iv, “Allowing Zero CPU Jobs”, on page 328</a></p>                                                                                                                                                                                                                                                                      |

**6.5.3.7 cpu Subsystem**

The `cpu` subsystem is an alternative way to control which processes get access to CPU resources. It cannot isolate jobs

with the precision of the `cpuset` subsystem, but you can use it in some specific circumstances:

- When you use the `cpuset` subsystem while allowing zero CPU jobs, but want to ensure that the Linux scheduler favors the jobs requesting one or more CPUs.
- When one of the features you use in PBS does not interoperate well with `cpusets`, for example when using suspend/resume when both the high-priority workload and the low-priority workload might have jobs that share `vnodes`. See [section 6.9.1, “Interactions Between Suspend/resume and the cpuset Subsystem”](#), on page 353.
- When you want to overcommit CPU resources (impossible if the CPUs are assigned to jobs via the `cpuset` subsystem) but still want jobs to get access to CPU resources according to the `ncpus` requested.

The `cpu` subsystem implements two different mechanisms:

- Linux scheduler fair sharing, where different shares are assigned to `cgroups`, according to requested `ncpus`. If there are CPU resource conflicts, the Linux scheduler favors `cgroups` that have used less than their allotted share
- Hard quotas that can be enforced if too much CPU usage is detected.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”](#), on page 319.

#### 6.5.3.7.i **cpu Subsystem Caveats**

- The `cpu` subsystem controls are often imprecise when hyperthreading is enabled, since the Linux kernel CFS scheduler sees each CPU thread as 100% of a CPU, regardless of how slow or fast it runs (which depends on usage of the other threads of the core). When `ncpus_are_cores` is enabled, quotas are multiplied by the number of threads per core, which can be misleading.
- There is no strict isolation between jobs; the Linux scheduler enforces a quota only on the set of processes, not individual processes. Linux fair sharing is not as efficient as `cpusets` in isolating jobs from rogue jobs. The `cpu` subsystem will slow down the rogue job, but since throttling may be uneven, a rogue process may still interfere with other jobs; if the hapless victims belong to a parallel application, that whole application may be affected, including its processes on CPUs where there is no conflict, since these will be forced to wait for the application process that was slowed down.

## 6.5.3.7.ii cpu Subsystem Configuration Parameters

Table 6-7: cpu Subsystem Configuration Parameters

| Parameter Name            | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------|----------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled                   |                            | <i>false</i>        | When set to <i>true</i> , the hook creates a <code>cpu</code> subsystem directory for each job, and assigns a number of shares and optionally a quota based on <code>ncpus</code> .<br><br>When set to <i>false</i> , no per-job cgroup is created for this subsystem.<br><br><a href="#">See section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”, on page 317.</a> |
| cfs_period_us             |                            | <i>100000</i>       | Time in microseconds between periodic checks by the Linux scheduler to compute usage and check quotas. Lowering this makes computation more precise and throttles rogues faster, but uses more OS scheduler overhead, with less CPU resources left for jobs.                                                                                                                                  |
| cfs_quota_fudge_factor    |                            | <i>1.03</i>         | Sets quota slightly above theoretically valid value.<br><br>Setting a CPU usage quota to 100% of a CPU still throttles applications, due to rounding errors.<br><br>The default is appropriate for the default <code>cfs_period_us</code> , but should be raised if <code>cfs_period_us</code> is lowered.                                                                                    |
| enforce_per_period_quotas |                            | <i>false</i>        | Specifies whether hard quotas are set.<br><br>When <i>false</i> , only shares are set, and a job can use more CPU resources than it requested provided other jobs leave CPU resources idle.<br><br>When <i>true</i> , hard CPU usage quotas are set                                                                                                                                           |
| zero_cpus_shares_fraction |                            | <i>0.002</i>        | Specifies fraction of shares allotted for a CPU to a "zero-CPU" job.<br><br>The default, <i>0.002</i> , is the minimum allowed by the kernel, and ensures that such a job gets CPU resources only if they are left idle by other jobs.                                                                                                                                                        |
| zero_cpus_quota_fraction  |                            | <i>0.2</i>          | Specifies fraction of a CPU allotted to a "zero-CPU" job before it is throttled by its hard quota.<br><br>Default fraction: <i>one-fifth of a CPU</i>                                                                                                                                                                                                                                         |

## 6.5.3.8 devices Subsystem

The `devices` subsystem is used to grant or restrict access to devices on the system, restricting the job to use specific devices, including GPU and Intel Xeon Phi ("MIC") devices (**deprecated**) assigned to the job. If MICs and/or GPUs are available in the complex and the `devices` subsystem is enabled, the PBS cgroups hook creates the `nmics` and/or `ngpus` resources if they are not already present.

Since detecting the GPU and/or MIC resources assigned to existing jobs relies on the `devices` cgroups for these jobs, enable this subsystem if you want the cgroups hook to manage GPU and/or MIC assignments.

For examples of how to use this subsystem, see [section 6.5.5, “Managing GPUs or Xeon Phi via Cgroups”, on page 344](#).

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 319](#).

You may want to discover and manage devices on hosts that do not have GPUs. You can speed that process, avoiding a call to `nvidia-smi`, by having the `discover_gpus` parameter evaluate to *false* for GPU-less hosts in the main section. Devices such as GPUs are discovered only when the `devices` subsystem is enabled.

See [section 6.5.5.6.i, “Caveats and Restrictions for Managing GPUs Externally to Cgroups Hook”, on page 350](#).

### 6.5.3.8.i Allowing Access to Devices

The `allow` parameter specifies how access to devices will be controlled. The list consists of entries in one of the following formats:

- A single string entry, used verbatim. For example:
  - `"b *.* rwm"` allows full access (read, write, and mknod) to all block devices
  - `"c *.* rwm"` allows full access to all character devices
- A list containing two strings. For example:
  - `["mic/scif", "rwm"]` looks for the major and minor number of the mic/scif device and allows full access.
  - If `ls /dev/mic` reported
 

```
"crw-rw-rw- 1 root root 244, 1 Mar 30 14:50 scif"
```

 then the line added to the allow file looks like
 

```
"c 244:1rwm"
```
- A list containing three strings. For example:
  - `["nvidiactl", "rwm", "*"]` looks for the major number of the nvidiactl device and allows full access to all devices with that major number.
  - If `ls /dev/nvidiactl` reported
 

```
"crw-rw-rw- 1 root root 284, 1 Mar 30 14:50 nvidiactl"
```

 then the line added to the allow file looks like
 

```
"c 284:* rwm"
```

**6.5.3.8.ii devices Subsystem Configuration Parameters****Table 6-8: devices Subsystem Configuration Parameters**

| Parameter Name | Default Value: Config File            | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------|---------------------------------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled        | <i>false</i>                          | <i>false</i>        | When set to <i>true</i> , the hook configures the <b>devices</b> subsystem based on the number of <b>nmics</b> and <b>ngpus</b> requested by the job. Refer to the <b>allow</b> parameter for additional information. When set to <i>false</i> , no cgroup is created for the <b>devices</b> subsystem.                                                                                                                                                                 |
| allow          | [<br>"b *:* rwm",<br>"c *:* rwm"<br>] | []                  | Specifies how access to devices will be controlled. The list consists of entries in one of the following formats: <ul style="list-style-type: none"> <li>• A single string entry, used verbatim. Example: "b *:* rwm"</li> <li>• A list containing two strings. Example: ["mic/scif", "rwm"]</li> <li>• A list containing three strings. Example: ["nvidiactl", "rwm", "*"]</li> </ul> See <a href="#">section 6.5.3.8.i, “Allowing Access to Devices”, on page 332</a> |

**6.5.3.9 memory Subsystem**

The **memory** subsystem allows you to monitor and limit the amount of physical memory used by all of the processes of a job on a host.

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 319](#).

Advantages to enabling the **memory** subsystem:

- With cgroups, jobs and operating systems are protected from any attempt by a job to use too much memory. Without cgroups, enforcing memory limits relies on monitoring and after-the-fact interventions to kill processes; MoM may not be able to act in time to protect the health of the host from being compromised through excessive memory usage.
- Accurate monitoring and records of memory usage. Without cgroups, memory usage such as **resources\_used.mem**, which is supposed to capture peak usage, relies instead on periodic polling, which can miss the high-water mark.
- The accuracy of memory usage monitoring is unaffected by processes that leave the Linux sessions registered as part of the job; without cgroups, this behavior breaks memory usage accounting when using some precompiled MPI libraries.
- Because memory usage reporting does not depend on polling, MoM can be configured to poll less often, which reduces the load on the PBS datastore.
- Jobs are prevented from rampantly filling host memory with kernel-allocated file cache. Instead, because kernel-allocated file cache for job file access is considered job memory, when the job hits its memory limit, job memory requests are fulfilled by reclaiming file cache allocated by the job earlier, or even by temporarily moving some of the job to swap until this can be done. If necessary, job processes will hang until memory can be allocated without crossing the memory usage limit.
- Recommended: you can reserve enough memory for the operating system and the file cache required for OS services to run well. If you do this, jobs are prevented from starving operating system services of memory resources.



### 6.5.3.9.i Reserving Memory

If you want to reserve memory so that it is not assigned to jobs, you can set a percent of physical memory using `reserve_percent`, and add a fixed amount to that using `reserve_amount`. The `reserve_amount` parameter sets a specific amount of available physical memory that is not to be assigned to jobs.

Reserving memory decreases the amount of `resources_available.mem` that MoM advertises to the server as being available for each vnode, as well as the amount of memory the cgroups hook is willing to assign to jobs.

For most HPC compute nodes with a minimum of 32GB, we recommend at least 2GB of memory for the operating system. Since there is no minimum amount of memory specified for a MoM host, the defaults are conservative.

### 6.5.3.9.ii Effect of Cgroups Hook on the mem Resource

The cgroups hook changes how much memory must be requested for job I/O and accounted in `resources_used.mem`.

Without the cgroups hook, file content cached in memory need not be included in a job's memory request, and is not accounted for in `resources_used.mem`. With the cgroups hook, reported memory usage reported includes memory for cached files accessed by the job.

Jobs requesting memory can use that amount both for physical memory and for caching. For example, when using the cgroups hook, a job that requests 20GB and uses 16GB but reads a 50GB file can hold only 4GB of the file in cache at a time. So if a job requires 32GB of application memory but also requires 5GB of private file cache to perform adequately, then it needs to request 37GB.

Memory is accounted accurately with cgroups. For jobs with multiple processes all accessing the same memory, without the cgroups hook the amount of memory reported as used is multiplied by the number of processes, but with the cgroups hook, the memory is only counted once. However, memory usage includes file cache placed into memory by job I/O, but not file cache merely accessed by the job but placed into memory earlier by unrelated processes.

It may be necessary to use other tools to determine application usage not involving file cache instead of `resources_used.mem`.

Applications using direct I/O to filesystems, meaning they bypass the buffer cache, are unaffected; these jobs do not see a change in `resources_used.mem` with and without cgroups.

### 6.5.3.9.iii Assigning a Default Amount of Memory to Jobs

The default parameter is the amount of memory assigned to the job if it doesn't request any memory. Because the scheduler does not know about this allocation, do not make this overly large, otherwise the cgroups hook may reject jobs because there isn't enough available memory. Instead, set large defaults for job memory resource requests (`Resource_List.mem`, `default_chunk.mem`, etc.) via a `queuejob` hook or defaults at the server or queue.

This value is not communicated to the server or scheduler; setting this value has no effect on the job's resource request.

### 6.5.3.9.iv Managing Use of Swap by Jobs

If your execution nodes have no swap or if you do not want your jobs to be able to swap to disk, set the `swappiness` parameter to `0` or `false`. In this case you should disable the `memsw` subsystem.

When this is zero or false, if a job cannot have its memory requests satisfied by claiming free physical memory or reclaiming memory from the page cache, then the Out of Memory killer will step in and kill job processes instead of allowing swap usage.

If you want your jobs to swap only when it is necessary to avoid job failure, and not to proactively move infrequently-used job pages to swap, set the `swappiness` parameter to `1` or `true`.

You can use the membership tuning tools described in [section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”](#), on page 317 to specify `swappiness` for hosts or vnodes.

Not recommended for HPC workloads: if you set this parameter to larger values such as *60*, the kernel will move infrequently-accessed user pages in order to free memory for file caching. We recommend using larger values only if you have at least as much swap as there is physical memory not assigned to jobs in the `memsw` subsystem. You can either not enable the `memsw` subsystem, or enable it but ensure there is enough swap and set `reserve_amount` in the `memsw` section to at least the physical memory of the host.

If `swappiness` is set to 1, make sure you enable the `memsw` subsystem unless you have an enormous amount of swap. Unless you explicitly set `resources_available.swap` on a node, the scheduler is unaware of the swap on that node if `memsw` is disabled. If `swappiness` is set to 1 and the `memsw` subsystem is disabled, jobs will swap but without any controls; you can run out of swap, performance can suffer, the OOM killer may kill the wrong process, and you can end up with black hole nodes.

#### 6.5.3.9.v Setting Memory Soft Limits

You can limit whether PBS imposes hard memory limits on job processes.

If you set the `soft_limit` parameter to *false*, PBS uses hard memory limits which prevent the processes from ever exceeding their requested memory usage. If a job accesses more memory than it has requested, some of the job's memory is moved to swap or the job is killed.

If you set this parameter to *true*, PBS uses soft memory limits; the memory requested by the job is set as a hint to the kernel as to what usage is expected, but no hard limit is enforced; when the kernel experiences memory shortage it uses the limits to select the cgroups from which memory is moved to swap.

#### 6.5.3.9.vi Setting Aside Memory for Kernel Drivers

Some kernel drivers (notably, GPFS or Lustre filesystem) may lower the memory available on the host or NUMA nodes by a small number of MB, typically 32MB or 64MB, after MoM has started.

This may reduce the amount of memory actually available at a vnode to less than the amount the scheduler thinks is available, and if jobs requiring the full amount are scheduled on that vnode, the hook will reject those jobs.

To compensate, you can hide some memory from the server. You set `vnode_hidden_mb` to for example "32" or "64".

The amount listed in the `vnode_hidden_mb` parameter lowers the memory advertised to the server when the `exechost_startup` hook is run, but not the memory that the cgroups hook itself is willing to assign to jobs.

#### 6.5.3.9.vii Using Configuration File Defaults for Memory

If you set the `enforce_default` parameter to *true*, jobs that don't explicitly request memory are bound by the defaults in the cgroups hook configuration file. If you set this to *false*, these jobs have access to all memory available to cgroups. By default, this parameter is set to *true*.

#### 6.5.3.9.viii Allowing Whole-host Jobs to Use Available Memory

If you set the `exclhost_ignore_default` parameter to *true*, jobs that request the whole host via `-lplace=exclhost` and do not explicitly request memory are treated as if `enforce_default` is *false*, and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the memory available to cgroups. If `enforce_default` is *false*, `exclhost_ignore_default` has no meaning. By default, this `exclhost_ignore_default` is set to *false*.

**6.5.3.9.ix memory Subsystem Configuration Parameters****Table 6-9: memory Subsystem Configuration Parameters**

| Parameter Name  | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|----------------------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled         | <i>true</i>                | <i>false</i>        | Boolean. When <i>True</i> , enables the memory subsystem.<br>See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”</a> , on page 317.                                                                                                                                                                                                                                                              |
| default         | <i>"256MB"</i>             | <i>"0MB"</i>        | Amount of memory assigned to the job if it doesn't request any memory. Recommendation: keep this small. See <a href="#">section 6.5.3.9.iii, “Assigning a Default Amount of Memory to Jobs”</a> , on page 334.                                                                                                                                                                                                                             |
| swappiness      |                            | <i>1</i>            | Sets the <code>memory.swappiness</code> value for the cgroup.<br>When set to <i>false</i> or <i>0</i> , jobs are not allowed to use swap.<br>When set to <i>true</i> or <i>1</i> , kernel is allowed to only swap if absolutely required.<br>When set to larger values, kernel is allowed to proactively swap. Not recommended for HPC workloads.<br>See <a href="#">section 6.5.3.9.iv, “Managing Use of Swap by Jobs”</a> , on page 334. |
| reserve_amount  | <i>"1GB"</i>               | <i>"0MB"</i>        | A specific amount of available physical memory that is not to be assigned to jobs.<br>The actual amount of memory reserved is <code>reserve_amount</code> plus the amount specified by <code>reserve_percent</code> .<br>See <a href="#">section 6.5.3.9.i, “Reserving Memory”</a> , on page 334.                                                                                                                                          |
| reserve_percent | <i>"0"</i>                 | <i>"0"</i>          | The percentage of available physical memory that is not to be assigned to jobs.<br>The actual amount of memory reserved is <code>reserve_amount</code> plus the amount specified by <code>reserve_percent</code> .<br>See <a href="#">section 6.5.3.9.i, “Reserving Memory”</a> , on page 334.                                                                                                                                             |
| soft_limit      | <i>false</i>               | <i>false</i>        | When set to <i>false</i> , PBS uses hard memory limits which prevent the processes from ever exceeding their requested memory usage.<br>When set to <i>true</i> , PBS uses soft memory limits; the memory requested by the job is set as a hint to the kernel as to what usage is expected, but no hard limit is enforced.<br>See <a href="#">section 6.5.3.9.v, “Setting Memory Soft Limits”</a> , on page 335.                           |
| vnode_hidden_mb | <i>"1"</i>                 | <i>"1"</i>          | Amount of memory per vnode to hide from the server but keep available to the hook, to compensate for memory reduction by kernel drivers.<br>See <a href="#">section 6.5.3.9.vi, “Setting Aside Memory for Kernel Drivers”</a> , on page 335.                                                                                                                                                                                               |

Table 6-9: memory Subsystem Configuration Parameters

| Parameter Name          | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------------|----------------------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enforce_default         | <i>true</i>                | <i>"true"</i>       | If you set the <code>enforce_default</code> parameter to <i>true</i> , jobs that don't explicitly request memory are bound by the defaults in the cgroups hook configuration file. If you set this to <i>false</i> , these jobs have access to all memory available to cgroups. By default, this parameter is set to <i>true</i> .                                                                                                              |
| exclhost_ignore_default | <i>false</i>               | <i>"false"</i>      | If you set the <code>exclhost_ignore_default</code> parameter to <i>true</i> , jobs that request the whole host via <code>-lplace=exclhost</code> and do not explicitly request memory are treated as if <code>enforce_default</code> is <i>false</i> , and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the memory available to cgroups. By default, this parameter is set to <i>false</i> . |
| oom_kill_disable        | <i>false</i>               | <i>"false"</i>      | Disables oom killer for the cgroup.                                                                                                                                                                                                                                                                                                                                                                                                             |
| under_oom_kill_delay    | 5                          | <i>"5"</i>          | Duration for which the <code>execlhost_periodic</code> event monitors jobs to see whether their processes are hung when memory requests cannot be satisfied.                                                                                                                                                                                                                                                                                    |
| oom_cache_percent_kill  | 5                          | <i>"5"</i>          | After <code>under_oom</code> persists for <code>under_oom_kill_delay</code> , if the amount of pagecache is less than <code>oom_cache_percent_kill</code> percent of the job's cgroup memory limit, the oom killer is re-enabled.                                                                                                                                                                                                               |

### 6.5.3.10 memsw Subsystem

The memsw subsystem is part of the memory subsystem. The memsw subsystem lets you specify how you want swap handled within the memory subsystem. The memsw subsystem allows you to monitor and limit swap used by all of the job processes on a host. If a job exceeds the limit, processes associated with that job are killed.

By default this subsystem is disabled. If you want to enable it, check the prerequisites in [section 6.5.1.1, “Ensure that Cgroups v1 are Available”](#), on page 316.

You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”](#), on page 319.

#### 6.5.3.10.i Effect of memsw Subsystem on the vmem Resource

Enabling the memsw subsystem changes the meaning of `vmem` to be memory + swap usage instead of the address space of the job processes. Enabling the memsw subsystem also changes how much `vmem` must be requested and how `vmem` is accounted in `resources_used.vmem`. The value of `resources_available.vmem` at a host or vnode reflects the disk swap that can be assigned to jobs. If there is more than one vnode per NUMA node, swap resources are split equally over the number of NUMA nodes reported.

If this subsystem is enabled, requesting the `vmem` resource sets a limit for the job's memory plus swap usage. For example, a job requesting `-lselect=1:ncpus=16:mem=8GB:vmem=64GB` is allowed to use 8GB in physical memory plus 56GB of memory resident in swap.

The value of `resources_used.vmem` reflects the job's memory plus swap usage across all nodes.

The way physical memory is accounted changes with the cgroups hook; see [section 6.5.3.9.ii, “Effect of Cgroups Hook on the mem Resource”](#), on page 334.

### 6.5.3.10.ii Reserving Swap

If you want to reserve swap so that it is not assigned to jobs, you can set a percent of swap using `reserve_percent`, and add a fixed amount to that using `reserve_amount`. The `reserve_amount` parameter sets a specific amount of swap that is not to be assigned to jobs.

Reserving swap decreases the amount of `resources_available.vmem` that MoM advertises to the server as being available for each vnode, as well as the amount of `vmem` the cgroups hook is willing to assign to jobs.

### 6.5.3.10.iii Computing Requested Swap

When the `manage_cgswap` parameter is *true*, the cgroups hook can compute the amount of swap a job requests. The hook computes the available swap for each vnode where it runs, and sets this value for the vnode in `resources_available.cgswap`. The value of `resources_available.cgswap` is `resources_available.vmem - resources_available.mem`.

The hook creates the `cgswap` resource.

To enable `cgswap` management:

1. Enable the cgroup hook
2. In the `memsw` section of the hook configuration file, set `manage_cgswap` to *true* or to a value that evaluates to *true* on the relevant hosts (e.g. by setting it to "vntype in: ignore\_default\_mem")
3. On the relevant hosts, HUP or restart the MoM (so that the hook computes `resources_available.cgswap`):  

```
killall -HUP pbs_mom
```
4. Set the flags for `cgswap` to "nhm", to make the server manage `resources_available.cgswap`:  

```
qmgr -c "set resource cgswap flag = 'nhm'"
```
5. Add "cgswap" to the "resources:" line in `$PBS_HOME/sched_priv/sched_config`
6. HUP the scheduler. On the host where the server runs:  

```
killall -HUP pbs_sched
```
7. Enable the cgroup hook's `queuejob` and `modifyjob` events:  

```
qmgr -c "set hook pbs_cgroups event +=queuejob"
qmgr -c "set hook pbs_cgroups event +=modifyjob"
```
8. Examine hook order. If there are other `queuejob` or `modifyjob` hooks that set or modify `mem` or `vmem`, you might want the cgroups hook `queuejob` and `modifyjob` events to run after them.

### 6.5.3.10.iv Using Configuration File Defaults for Swap

If you set the `enforce_default` parameter to *true*, jobs that don't explicitly request swap are bound by the defaults in the cgroups hook configuration file. If you set this to *false*, these jobs have access to all swap available to cgroups. By default, this parameter is set to *true*.

### 6.5.3.10.v Allowing Whole-host Jobs to Use Available Swap

If you set the `exclhost_ignore_default` parameter to *true*, jobs that request the whole host via `-lplace=exclhost` and do not explicitly request swap are treated as if `enforce_default` is *false*, and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the swap available to cgroups. If `enforce_default` is *false*, `exclhost_ignore_default` has no meaning. By default, this parameter is set to *false*.

**6.5.3.10.vi memsw Subsystem Configuration Parameters****Table 6-10: memsw Subsystem Configuration Parameters**

| Parameter Name          | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------|----------------------------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enabled                 | <i>false</i>               |                     | Boolean.<br>When <i>true</i> , enables the memsw subsystem.<br>When <i>false</i> , the subsystem is disabled, and jobs reaching their memory limit are allowed to use swap in an unrestrained fashion (unless <code>memory.swappiness</code> was set to 0) until resources are depleted.<br>See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”, on page 317</a> .                                |
| default                 | <i>"0B"</i>                | <i>"0"</i>          | Specifies the amount of swap assigned to the job if it doesn't request any <code>vmem</code> or <code>cgsnap</code> resource.<br>This value is not communicated to the server or scheduler; setting this value has no effect on the job's resource request.                                                                                                                                                                                 |
| reserve_amount          | <i>"64MB"</i>              | <i>"0MB"</i>        | An amount of available swap that is not to be assigned to jobs.<br>The amount reserved is the amount determined by <code>reserve_percent</code> plus <code>reserve_amount</code> .<br>See <a href="#">section 6.5.3.10.ii, “Reserving Swap”, on page 338</a> .                                                                                                                                                                              |
| reserve_percent         | <i>"0"</i>                 | <i>"0"</i>          | Percentage of available swap that is not to be assigned to jobs.<br>The amount reserved is the amount determined by <code>reserve_percent</code> plus <code>reserve_amount</code> .<br>See <a href="#">section 6.5.3.10.ii, “Reserving Swap”, on page 338</a> .                                                                                                                                                                             |
| manage_cgswap           | <i>false</i>               | <i>"false"</i>      | Boolean. When set to <i>true</i> , the memsw computes how much swap is available and advertises it to the scheduler via <code>resources_available.cgswap</code> .                                                                                                                                                                                                                                                                           |
| enforce_default         | <i>true</i>                | <i>"true"</i>       | If you set the <code>enforce_default</code> parameter to <i>true</i> , jobs that don't explicitly request swap are bound by the defaults in the cgroups hook configuration file. If you set this to <i>false</i> , these jobs have access to all swap available to cgroups. By default, this parameter is set to <i>true</i> .                                                                                                              |
| exclhost_ignore_default | <i>false</i>               | <i>"false"</i>      | If you set the <code>exclhost_ignore_default</code> parameter to <i>true</i> , jobs that request the whole host via <code>-lplace=exclhost</code> and do not explicitly request swap are treated as if <code>enforce_default</code> is <i>false</i> , and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the swap available to cgroups. By default, this parameter is set to <i>false</i> . |

**6.5.3.10.vii Scheduling on the vmem Resource**

To allow the scheduler to take `resources_available.vmem` and `resources_assigned.vmem` on nodes and `vmem` requested by jobs into account when deciding where and when to schedule jobs, list "vmem" on the "resources:" line in `$PBS_HOME/sched_priv/sched_config`.



### 6.5.3.10.viii Caveat for Swap Limits

When enabling the memsw subsystem in the cgroup hook, `manage_rlimit_as` should be set to *true*. This way `RLIMIT_AS` is unlimited unless the job requests `pvmem`, in which case `RLIMIT_AS` is set to the value of `pvmem`.

Without the `manage_rlimit_as` parameter set to *true*, memory limits are imposed as described in [section 5.15.2.4.i, “Job Memory Limit Enforcement on Linux”](#), on page 302. This address space limit is usually too low.

### 6.5.3.10.ix Caveat for Jobs that Use Swap

If jobs will use swap, we recommend enabling the memsw subsystem and setting `manage_cgswap` to *true*.

If you set `manage_cgswap` to *true*, but disable the `queuejob` event, make sure that the jobs correctly request `mem`, `vmem`, and `cgswap`.

If `manage_cgswap` is false, jobs can be submitted requesting only `mem` and possibly `vmem`, but you run the risk of running out of swap unless you use the second or third recommendation below.

The cgroups hook prevents a job from using more physical memory than it has requested, which means that a swap shortage cannot always be made up with physical memory.

If the memory and memsw subsystems are enabled, a job can fall into a trap where the scheduler thinks there is enough swap at a host or vnode, but there is not. The reason is that there is no separate resource for swap; `resources_available.vmem` is the sum of physical memory plus swap.

For example, suppose a node has 64GB of physical memory and 2GB of swap. With no memory reservations in the cgroups hook configuration file, `resources_available.mem` is approximately 64GB and `resources_available.vmem` is approximately 66GB.

If you submit a job with `-lselect=1:mem=2GB:vmem=10GB`, the scheduler sees enough available `vmem` and enough available `mem` on the node. But when the job runs, if it does indeed use 10GB of memory, it will fail. The memory cgroup will limit the job memory resident in physical memory to 2GB, but there is only 2GB of swap, so even though there is enough memory plus swap, the job will not be able to use 8GB of swap to make up the remainder of the 10GB.

The hook will try to catch one common case: if the explicitly requested `vmem` is larger than the requested `mem`, then nodes without any swap will refuse to run the job; the cgroups hook will reject the request to run the job. Administrators and job submitters must ensure that such jobs do not land on nodes without swap, for example by using resources to tag nodes accordingly and setting the proper requests based on those tags.

To avoid the problem of running out of swap when enabling both memory and memsw:

- Set `manage_cgswap` to *true*, so that the cgroups hook can manage swap as a separate resource. See [section 6.5.3.10.iii, “Computing Requested Swap”](#), on page 338.
- Set `reserve_amount` in the memsw section to a value that is equal to `resources_available.mem` for the host; this makes any job specification safe, but requires that swap resources are larger than the physical memory on the host.
- Do not use jobs that leave a lot of physical memory on the host unrequested if `Resource_List.vmem` > `Resource_List.mem`: only use jobs like that when the physical memory cannot fit the job, to use swap as extra memory. This will drastically reduce the amount of swap you need to reserve as not visible to jobs in the configuration file.

## 6.5.3.11 hugetlb Subsystem

The hugetlb subsystem lets you manage the amount of huge page memory used in a cgroup.

While this subsystem imposes limits on the huge pages that can be used by a job, it does not create huge pages on the different NUMA nodes. That must be done separately, so that the cgroups hook `exechohost_startup` portion can report huge pages available on the different nodes.

You can control how many huge pages you want on a set of nodes dynamically:

```
numactl -m <node list> echo X >/proc/sys/vm/nr_hugepages_mempolicy.
```



You can see the number of huge pages currently available here:

```
/sys/devices/system/node/node[0-9]*/hugepages/
```

The `free_hugepages` and `surplus_hugepages` pseudofiles are read-only.

Writing to `nr_hugepages` tells the system to adjust the number of persistent huge pages on the NUMA node, but if existing memory that is free is too fragmented, it may have to repurpose buffer cache pages for it (which may take time if the pages are dirty and need to be pushed to the filesystem) or even move existing used memory to swap.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in [section 6.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”](#), on page 319.

### 6.5.3.11.i Reserving Huge Page Memory

If you want to reserve huge page memory (`hpmem`) so that it is not assigned to jobs, you can set a percent of huge page memory using `reserve_percent`, and add a fixed amount to that using `reserve_amount`. The `reserve_amount` parameter sets a specific amount of huge page memory that is not to be assigned to jobs.

Reserving huge page memory decreases the amount that MoM advertises to the server as being available for each node, as well as the amount that the cgroups hook is willing to assign to jobs.

### 6.5.3.11.ii Caveat for hugetlb Subsystem

When a job spans more than one vnode, it may split its allocation of huge page memory across NUMA nodes differently from how PBS assigned the memory. This can lead to other jobs on those NUMA nodes not having enough huge page memory.

You can:

- Use utilities to check whether huge pages are available on the correct NUMA nodes just before you launch applications
- Use a memory-fences-safe workload
- Disable memory fences to allow huge pages to be allocated off-node

### 6.5.3.11.iii Using Configuration File Defaults for Huge Pages

If you set the `enforce_default` parameter to `true`, jobs that don't explicitly request huge pages are bound by the defaults in the cgroups hook configuration file. If you set this to `false`, these jobs have access to all huge pages available to cgroups. By default, this parameter is set to `true`.

### 6.5.3.11.iv Allowing Whole-host Jobs to Use Available Huge Pages

If you set the `exclhost_ignore_default` parameter to `true`, jobs that request the whole host via `-lplace=exclhost` and do not explicitly request huge pages are treated as if `enforce_default` is `false`, and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the huge pages available to cgroups. If `enforce_default` is `false`, `exclhost_ignore_default` has no meaning. By default, this parameter is set to `false`.

**6.5.3.11.v hugetlb Subsystem Configuration Parameters****Table 6-11: hugetlb Subsystem Configuration Parameters**

| Parameter Name          | Default Value: Config File | Default Value: Hook | Description                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------|----------------------------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| default                 | <i>0MB</i>                 |                     | The amount of huge page memory assigned to the cgroup when the job does not request hpmem.                                                                                                                                                                                                                                                                                                                                                              |
| enabled                 | <i>false</i>               |                     | When set to <i>true</i> , the hook registers a limit that restricts the amount of hugepage memory processes may access.<br>When set to <i>false</i> , no limit is registered.<br>See <a href="#">section 6.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”</a> , on page 317.                                                                                                                                                       |
| reserve_amount          | <i>0MB</i>                 |                     | An amount of available huge page memory (hpmem) that is not to be assigned to jobs.<br>The amount reserved is the amount determined by <code>reserve_percent</code> plus <code>reserve_amount</code> .<br>See <a href="#">section 6.5.3.11.i, “Reserving Huge Page Memory”</a> , on page 341.                                                                                                                                                           |
| reserve_percent         | <i>0</i>                   |                     | The percentage of available huge page memory (hpmem) that is not to be assigned to jobs.<br>The amount reserved is the amount determined by <code>reserve_percent</code> plus <code>reserve_amount</code> .<br>See <a href="#">section 6.5.3.11.i, “Reserving Huge Page Memory”</a> , on page 341.                                                                                                                                                      |
| enforce_default         | <i>true</i>                | <i>"true"</i>       | If you set the <code>enforce_default</code> parameter to <i>true</i> , jobs that don't explicitly request huge pages are bound by the defaults in the cgroups hook configuration file. If you set this to <i>false</i> , these jobs have access to all huge pages available to cgroups. By default, this parameter is set to <i>true</i> .                                                                                                              |
| exclhost_ignore_default | <i>false</i>               | <i>"false"</i>      | If you set the <code>exclhost_ignore_default</code> parameter to <i>true</i> , jobs that request the whole host via <code>-lplace=exclhost</code> and do not explicitly request huge pages are treated as if <code>enforce_default</code> is <i>false</i> , and not bound by the defaults in the cgroup hook configuration file, so they have access to all of the huge pages available to cgroups. By default, this parameter is set to <i>false</i> . |

**6.5.3.12 Sample Cgroups Hook Configuration File**

Here we show a sample cgroups hook configuration file similar to the default configuration file:

```
{
 "enabled" : "vntype not in: no_cgroups",
 "cgroup_prefix" : "pbs_jobs",
 "periodic_resc_update" : true,
 "vnode_per_numa_node": : false,
 "online_offlined_nodes" : true,
 "use_hyperthreads" : true,
```

---

```

"ncpus_are_cores" : true,
"manage_rlimit_as" : true,
"ngpus_ext_managed" : false,
"discover_gpus" : true
"cgroup":{
 "cpuacct":{
 "enabled" : true
 },
 "cpuset":{
 "enabled" : true,
 "exclude_cpus" : [0,8],
 "mem_fences" : false,
 "mem_hardwall" : false,
 "memory_spread_page" : true
 },
 "devices":{
 "enabled" : false,
 "allow":[
 "b*:rwm",
 "c*:rwm"
]
 }
}
"memory":{
 "enabled" : true,
 "soft_limit" : false,
 "default" : "256MB",
 "reserve_percent" : "0",
 "reserve_amount" : "1GB",
 "soft_limit" : "false",
 "vnode_hidden_mb" : "1",
 "enforce_default" : "true",
 "exclhost_ignore_default" : "false"
},
"memsw":{
 "enabled" : false,
 "default" : "256MB",
 "reserve_percent" : "0",
 "reserve_amount" : "1GB",
 "manage_cgswap" : "false",
 "enforce_default" : "true",
 "exclhost_ignore_default" : "false"
},
"hugetlb":{
 "enabled" : false,
 "default" : "0MB",
 "reserve_percent" : "0",

```

```

 "reserve_amount" : "0MB",
 "enforce_default" : "true",
 "exclhost_ignore_default" : "false"
 }
}
}

```

## 6.5.4 Finish Up

### 6.5.4.1 Enable cgroups hook

The cgroups hook and its default configuration file are already imported. You must enable the cgroups hook as root:

1. Log in as root
2. Enable the cgroups hook on the server host:

```
Qmgr: set hook pbs_cgroups enabled = true
```

### 6.5.4.2 HUP or Restart MoM

HUP or restart each MoM:

```
kill -HUP <MoM PID>
```

or

```
<path to PBS start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

### 6.5.4.3 Enable Use of Resources by the Scheduler

Modify the resources: line in <sched\_priv directory>/sched\_config:

- The nmics and ngpus resources are automatically created, but you have to add them to the resources: line in <sched\_priv directory>/sched\_config.
- If you have configured huge page memory, and it is enabled in the cgroups hook, PBS creates the hpmem resource, but you need to add it to the resources: line in <sched\_priv directory>/sched\_config.
- If you have configured the memsw subsystem, add "vmem" to the resources: line in <sched\_priv directory>/sched\_config.

Set resource flags:

- If the cgroups hook creates the nmics and ngpus resources, you may need to set their flags. You also need to set the flags for the vmem and hpmem resources. Set the flags for the nmics, ngpus, vmem, and hpmem resources to "nh":

```
Qmgr: set resource nmics,ngpus,vmem,hpmem flag=nh
```

## 6.5.5 Managing GPUs or Xeon Phi via Cgroups

As of 2022.1.0, support for Xeon Phi is **deprecated**.

Integration with Linux cgroups allows PBS to automatically detect and configure GPUs and Xeon Phi processors. However, you may want to avoid having the cgroups hook manage GPUs on particular nodes, especially if you want to over-subscribe those GPUs. See [section 6.5.5.6, “Not Using Cgroups to Manage GPUs”, on page 349](#).

Since some of the details of managing GPUs and Xeon Phi are different, in the following sections we will proceed by describing the case with GPUs.

### 6.5.5.1 Managing GPUs via Cgroups

PBS can restrict jobs to specific allocated GPUs. If you set `vnode_per_numa_node` to *true* in the cgroups hook configuration file, PBS takes advantage of topology and associates GPUs with the closest memory and CPUs in the system.

If GPUs are available in the complex and the `devices` subsystem is enabled, the PBS cgroups hook creates the `ngpus` resource if it is not already present, and discovers and sets values for it.

### 6.5.5.2 Using NVIDIA Multi-Instance GPUs (MIGs)

The NVIDIA Multi-Instance GPU (MIG) feature partitions the GPU into multiple separate GPU instances. PBS recognizes each GPU instance and treats it as a normal GPU. To use the MIG feature, follow the steps in the NVIDIA documentation at <https://docs.nvidia.com/datacenter/tesla/mig-user-guide/>:

- Enable the MIG feature
- Enable the `nv_cap_enable_devfs` NVIDIA kernel module parameter
- Create GPU Instances (GIs)
- Create Compute Instances (CIs)

### 6.5.5.3 Configuration Steps

Here we summarize the configuration steps that allow the cgroups hook to manage your GPUs:

- Modify the `resources:` line in `<sched_priv directory>/sched_config`.

The `ngpus` resource is automatically created, but you have to add it to the resources: line in `<sched_priv directory>/sched_config`.

- Set the flags for the `ngpus` resource to `"nh"`:

```
qmgr: set resource ngpus flag=nh
```

- Export the installed PBS cgroups hook configuration file:

```
qmgr -c "export hook pbs_cgroups application/x-config default" >pbs_cgroups.json
```

- Edit this file and change its parameters:

- If `nvidia-smi` is someplace other than `/usr/bin/nvidia-smi`, add the `nvidia-smi` global parameter with the absolute path to `nvidia-smi`. Be sure to add the comma to the end of the line. For example:

```
"nvidia-smi" : "/usr/bin/nvidia/nvidia-smi",
```

- Add the `"nvidiactl"` value to the `allow` parameter of the `devices` subsystem section. Be sure to add the comma to the end of the previous line:

```
"c : rwm",
["nvidiactl", "rwm", "*"]
```

- Make sure that the `vnode_per_numa_node` global parameter is set to `true`

- Enable the `devices` subsystem (it is disabled by default); see [section 6.5.3.8, “devices Subsystem”, on page 331](#)

```
"enabled" : true,
```

- Enable the `cpuset` subsystem (it is enabled by default); see [section 6.5.3.6, “cpuset Subsystem”, on page 326](#)

```
"enabled" : true,
```

- Read the configuration file back in:

```
qmgr -c "import hook pbs_cgroups application/x-config default pbs_cgroups.json"
```

- Enable the cgroups hook:

```
qmgr -c "set hook pbs_cgroups enabled=True"
```

- HUP or restart each MoM:

```
kill -HUP <MoM PID>
```

or

```
<path to PBS start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

If device isolation is not enabled for GPUs, the hook assigns devices to jobs, and sets the `CUDA_VISIBLE_DEVICES` environment variable for processes created as children of MoM according to the devices assigned to the job on the host on which the process runs.

### 6.5.5.4 Isolating NVIDIA GPUs

For NVIDIA GPU isolation to work, you need to restrict access to only those devices assigned to the job. In the `"allow"` subsection of the `devices` section of the configuration file, do not use the broad `"c *: rwm"`.

Make sure that the `"allow"` section excludes read and write access for the 195 major number (`"c 195:* m"`), which is what all NVIDIA devices use. Preserve `mknod` (`"m"`) access, since other software such as the container hook may need `"m"` access.

You also need to include `["nvidia-uvm", "rwm"]`, since it is part of how the driver determines isolation, and possibly other global NVIDIA devices used by NVIDIA tools. You may also have to add other devices, such as those required for MPI library access to Infiniband devices.

The cgroups hook assigns the correct NVIDIA GPUs when a job requests `ngpus`.

Here is an example of the `devices` section of the cgroups hook configuration file, configured to allow NVIDIA GPU isolation:

```
"devices":{
 "enabled" : true,
 "allow" :
 [
 "b *:* m",
 "b 7:* rwm",
 "c *:* m",
 "c 136:* rwm",
 "c 195:* m",
 ["infiniband/rdma_cm", "rwm"],
 ["fuse", "rwm"],
 ["net/tun", "rwm"],
 ["tty", "rwm"],
 ["ptmx", "rwm"],
 ["console", "rwm"],
 ["null", "rwm"],
 ["zero", "rwm"],
 ["full", "rwm"],
 ["random", "rwm"],
 ["urandom", "rwm"],
 ["cpu/0/cpuid", "rwm", "*"],
 ["nvidia-modeset", "rwm"],
 ["nvidia-uvm", "rwm"],
 ["nvidia-uvm-tools", "rwm"],
 ["nvidiactl", "rwm"]
]
},
```

Here we detail the block devices with major number 7. On this host, block device 7 are the loop devices:

```
root@myhost:~# ls -l loop*
brw-rw----. 1 root disk 7, 0 May 18 08:22 loop0
brw-r-----. 1 root disk 7, 1 May 18 07:43 loop1
brw-r-----. 1 root disk 7, 10 May 18 07:43 loop10
brw-r-----. 1 root disk 7, 100 May 18 07:43 loop100
brw-r-----. 1 root disk 7, 101 May 18 07:43 loop101
brw-r-----. 1 root disk 7, 102 May 18 07:43 loop102
brw-r-----. 1 root disk 7, 103 May 18 07:43 loop103
brw-r-----. 1 root disk 7, 104 May 18 07:43 loop104
brw-r-----. 1 root disk 7, 105 May 18 07:43 loop105
brw-r-----. 1 root disk 7, 106 May 18 07:43 loop106
brw-r-----. 1 root disk 7, 107 May 18 07:43 loop107
```



Here we detail the 136 device:

```
root@myhost:~# ls -l /dev/pts
crw----- 1 altair tty 136, 0 Jul 22 20:38 0
c----- 1 root root 5, 2 Jul 22 19:18 ptmx
```

Here we detail the 195 devices:

```
root@myhost:~# ls -l /dev/nvidia*
crw-rw-rw- 1 root root 195, 0 Jul 22 19:18 /dev/nvidia0
crw-rw-rw- 1 root root 195, 1 Jul 22 19:18 /dev/nvidia1
crw-rw-rw- 1 root root 195, 2 Jul 22 19:18 /dev/nvidia2
crw-rw-rw- 1 root root 195, 3 Jul 22 19:18 /dev/nvidia3
crw-rw-rw- 1 root root 195, 255 Jul 22 19:18 /dev/nvidiactl
crw-rw-rw- 1 root root 195, 254 Jul 22 19:35 /dev/nvidia-modeset
crw-rw-rw- 1 root root 238, 0 Jul 22 19:35 /dev/nvidia-vm
crw-rw-rw- 1 root root 238, 1 Jul 22 19:35 /dev/nvidia-vm-tools
```

### 6.5.5.5 Environment Variables for CUDA and Xeon Phi

As of 2022.1.0, support for Xeon Phi is **deprecated**. When you use Xeon Phi co-processors, PBS sets the `OFFLOAD_DEVICES` environment variable during job initialization, for each process that is a child of MoM.

When you use CUDA devices, PBS sets the `CUDA_VISIBLE_DEVICES` environment variable for each process that is a child of MoM.

#### 6.5.5.5.i Using `CUDA_VISIBLE_DEVICES` with Multihost Jobs

On each host, the correct value for `CUDA_VISIBLE_DEVICES` references unique identifiers for the GPUs on that host, which means that the value of `CUDA_VISIBLE_DEVICES` is different on each host of a multihost job. As a result, you cannot just propagate the value of `CUDA_VISIBLE_DEVICES` visible in the job script to all processes of a multihost job; you need to get the correct value for each host.

PBS sets the correct value for the `CUDA_VISIBLE_DEVICES` environment variable for the job script on the primary execution host of the job.

For job processes on remote hosts, the `CUDA_VISIBLE_DEVICES` environment variable is also correctly set if the processes are spawned as children of MoM through the Task Management API (the TM API). To spawn processes through the TM API, you can use `pbs_tmsh`, or an MPI library that was compiled with TM API support, or an MPI library that was configured to use `pbs_tmsh` as the remote process launcher; see [Chapter 13, "Using MPI with PBS", on page 559](#).

For other jobs that use an MPI library that does not spawn processes through the TM API (for example HPE MPI, for which remote processes are spawned by array services) or that use other mechanisms such as `ssh`, processes on sister hosts are not children of MoM, and MoM is unable to set the `CUDA_VISIBLE_DEVICES` environment variable directly for these processes. Processes can use `pbs_attach` to join the job, but `pbs_attach` cannot set environment variables for existing processes.

For these non-MoM-child processes on sister hosts it is necessary to fetch the correct value for the `CUDA_VISIBLE_DEVICES` environment variable from a file PBS wrote for the job. The location of that file depends on `PBS_HOME` (or `PBS_MOM_HOME` if it is set) and the job ID. In the rest of this section we describe how to find the value for `CUDA_VISIBLE_DEVICES` using `PBS_HOME`. See examples [\\$PBS\\_HOME is different and PBS\\_MOM\\_HOME is not defined](#) and [\\$PBS\\_HOME is different and PBS\\_MOM\\_HOME is defined](#).

If `PBS_HOME`/`PBS_MOM_HOME` is the same on all hosts for the job, the file that contains the `CUDA_VISIBLE_DEVICES` is in the same location on each host, and you can pass the location of the file to processes on sister hosts via an environment variable. The `PBS_NODEFILE` environment variable is set to the path to the file containing the GPU identifiers for `CUDA_VISIBLE_DEVICES`; see the example [\\$PBS\\_HOME is the same everywhere](#).

If PBS\_HOME may be different on the different hosts:

- Pass the job ID to the sister host(s):
  - Transmit the job ID from the primary execution host to the sister(s):  
On the primary execution host, make sure that /etc/ssh/ssh\_config contains "SendEnv PBS\_JOBID".  
On the sister host(s):
    - Make sure that /etc/ssh/sshd\_config contains "AcceptEnv PBS\_JOBID".
    - Restart sshd if necessary.
- On the sister host, look up PBS\_HOME in /etc/pbs.conf (in a shell process, you can source /etc/pbs.conf directly).

Once you know PBS\_HOME and PBS\_JOBID, you know the name of the file that contains the CUDA\_VISIBLE\_DEVICES assignment. The file is located at \$PBS\_HOME/aux/\$PBS\_JOBID.env .

Example 6-3: \$PBS\_HOME is different and PBS\_MOM\_HOME is not defined

Look in or source /etc/pbs.conf. \$PBS\_HOME/aux/\$PBS\_JOBID.env is the file to read.

If \$PBS\_HOME is /var/spool/node2, and \$PBS\_JOBID is 332.tc72:

```
echo $PBS_HOME/aux/$PBS_JOBID.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_HOME/aux/$PBS_JOBID.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

Example 6-4: \$PBS\_HOME is different and PBS\_MOM\_HOME is defined

\$PBS\_MOM\_HOME/aux/\$PBS\_JOBID.env is the file to read.

If \$PBS\_MOM\_HOME is /var/spool/node2, and \$PBS\_JOBID is 332.tc72:

```
echo $PBS_MOM_HOME/aux/$PBS_JOBID.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_MOM_HOME/aux/$PBS_JOBID.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

Example 6-5: \$PBS\_HOME is the same everywhere

\$PBS\_NODEFILE.env is the file to read.

If \$PBS\_HOME is /var/spool/node2, and \$PBS\_JOBID is 332.tc72:

```
echo $PBS_NODEFILE.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_NODEFILE.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

### 6.5.5.6 Not Using Cgroups to Manage GPUs

You may want to oversubscribe the GPUs on some nodes, which means that the cgroups hook does not manage the ngpus resource on some nodes but does manage it on others. For example, you may have test nodes each with a single GPU that is to be shared by more than one job, mainly for testing. You would like to set resources\_available.ngpus to a fixed number larger than one, via either Version 2 configuration files or qmgr, to manage how much to oversubscribe those GPU resources.

To prevent the cgroups hook from setting the value for `resources_available.ngpus` on a host or assigning GPUs to jobs on that host, set the `ngpus_ext_managed` parameter in the main section of the cgroups hook configuration file to *true*; this indicates that `resources_available.ngpus` is managed externally to the hook and that the hook should not assign GPUs to jobs when jobs request the `ngpus` resource on that host.

The default value for the `ngpus_ext_managed` parameter is *false*. When you set the value to *true*:

- The `discover_gpus` parameter is disabled, since the hook does not need to discover GPUs if it will not manage GPU assignment
- The hook never sets `resources_available.ngpus` on the vnodes of the host, allowing you to use Version 2 configuration files or `qmgr` to set it to any value
- The hook does not attempt to assign individual GPUs to jobs; the hook will allow a job on a node regardless of the `ngpus` resources requested by the job, leaving it to the external manager of `resources_available.ngpus` and the scheduler and server (which manages `resources_assigned.ngpus`) to manage the `resources_available.ngpus` vnode resource.

#### 6.5.5.6.i Caveats and Restrictions for Managing GPUs Externally to Cgroups Hook

If you are using something other than the cgroups hook to manage GPUs, and have enabled the `devices` subsection and configured it to implement device isolation, do not forget that GPUs will not be added automatically by the cgroups hook to the devices the job is allowed to use. To prevent problems, you can do any of the following:

- Disable the `devices` controller on the nodes where `ngpus_ext_managed` is enabled
- Do not use device isolation
- Add all GPU devices to the list of allowed devices if using device isolation
- Use another hook to add the required devices

## 6.6 Configuring MPI for Cgroups

In order to capture job processes and put their PIDs in cgroups, PBS needs the MPI to tell it about those processes. An MPI that is integrated with PBS does this. You need to make sure that your MPI is integrated with PBS. If you are already using an MPI that is integrated with PBS, you do not need to perform this step. OpenMPI, MVAPICH2, and MPICH behave well if they have been compiled with support for the TM API and linked with the PBS libraries. Intel MPI also behaves well if it has been integrated with PBS.

However, if your MPI uses `ssh` and is not integrated with PBS, you can use `pbs_attach` to capture processes started with `ssh`. In [section 13.1, “Integration with MPI”, on page 559](#), we describe integrating MPIs with PBS. If your MPI is not integrated with PBS, cgroups cannot help you manage spawned processes.

If your MPI is not integrated with PBS, you might notice that jobs are running significantly slower, or jobs are crashing with errors such as "Unable to set CPU", or "Unable to join process"; the MPI may be trying to pin all processes to CPU 0 or crashing.

Wrapping `ssh` is sufficient for all precompiled MPIs to work.

## 6.6.1 Steps to Integrate MPI with PBS via ssh

The following is a helpful example of integrating MPI with PBS via `ssh`:

- On each host in the PBS complex, edit `/etc/ssh/ssh_config`:
  - Add the following as the last `SendEnv` line, after the other `SendEnv` lines:
 

```
SendEnv PBS_JOBID
```
- On each execution host in the PBS complex, edit `/etc/ssh/sshd_config`:
  - Add the following as the last `AcceptEnv` line, after the other `AcceptEnv` lines:
 

```
AcceptEnv PBS_JOBID
```
- On each host in the PBS complex, restart `sshd`:
 

```
/etc/init.d/sshd restart
```
- On each host in the PBS complex, edit `/etc/ssh/sshrd` to include the following lines:
 

```
#!/bin/sh
if read proto cookie and [-n "$DISPLAY"]; then
 if [`echo $DISPLAY | cut -c1-10` = 'localhost:']; then
 # X11UseLocalhost=yes
 echo add unix:`echo $DISPLAY |
 cut -c11-` $proto $cookie
 else
 # X11UseLocalhost=no
 echo add $DISPLAY $proto $cookie
 fi | xauth -q -
fi
string=$*
Make sure the following points to your $PBS_EXEC/bin
pbs_bin="/opt/pbs/bin"
attach_cmd="pbs_attach"
#echo "PBS_JOBID: $PBS_JOBID"
#echo "$*"
if [-n "$PBS_JOBID"]; then
 # Check to see whether the command is already calling pbs_attach
 if ["${string/$attach_cmd}" = "$string"] ; then
 echo "Attaching $PPID to $PBS_JOBID"
 $pbs_bin/$attach_cmd -j $PBS_JOBID -p $PPID 2> /dev/null
 exit 0
 fi
fi
```
- Test to ensure that it works as expected and PBS can capture PIDs:
  - Make sure cgroups are enabled
  - Submit an interactive PBS job
  - `ssh` into a host belonging to the job and verify that the job process PID was added to the tasks file for the desired subsystem, e.g. `cpuacct/pbspro/<job ID>/tasks`

## 6.7 Managing Jobs with Cgroups

### 6.7.1 Requesting Memory

The default amount of memory assigned by cgroups to jobs that do not request it is 256MB. If this value does not work for your site, either change the default value if you need to only slightly more, or assign a default value using a `queuejob` hook or a server default. See [section 6.5.3.9.iii, “Assigning a Default Amount of Memory to Jobs”, on page 334](#).

### 6.7.2 Limit Enforcement

When a job is killed due to hitting a cgroup limit, you will see something like the following in the job's `stdout`:

```
mpirun noticed that process rank 0 with PID 115249 on node node0042 exited on signal 9 (Killed).
```

The hook will also attempt to find OOM killer messages in the kernel `dmesg` buffer. If it finds them it prints more specific errors in the MoM log, and in the job's `stderr`, if the cgroup limit violation occurs on the first node.

The messages will contain either "Cgroup memory limit exceeded" or "Cgroup memsw limit exceeded" and will attempt to print the corresponding kernel `dmesg` buffer messages (if found), which usually identify the process that was killed.

### 6.7.3 Examples of Requesting Cores and Hyperthreads

Assume we have 2-way hyperthreaded processors.

When hyperthreading is enabled on a system and `ncpus_are_cores` is disabled, each core is associated with two threads. In this case, if a job submitter wants all threads of a hyperthreaded core, they should request `ncpus` in multiples of 2.

When hyperthreading is not enabled, or if it is and `ncpus_are_cores` is enabled, a job submitter should request just the number of cores

For example, a job submitter requests the following on a cluster with 2-way hyperthreaded CPUs on all nodes:

```
-lselect=1:ncpus=2
```

Result:

- If hyperthreading is not enabled, this nets two cores. If the `cpuset` subsystem is enabled, only the first thread of each core is visible in the job `cpuset`.
- If hyperthreading is enabled and `ncpus_are_cores` is enabled, this also nets two cores, with a total of four threads visible in the job's `cpuset`.
- If hyperthreading is enabled and `ncpus_are_cores` is disabled, this nets two threads, with an attempt to assign the two threads from a single core; this may not succeed if other jobs on the vnode have requested odd numbers of `ncpus`.

### 6.7.4 Spawning Job Processes

When a job process is spawned using `tm_spawn`, the `execjob_launch` cgroups hook runs. The `execjob_launch` hook can set environment variables correctly and set per-process limits for the processes.

When a job process is spawned outside of PBS and `pbs_attach` is used to make the process join the job, the `execjob_attach` cgroups hook runs, but it is unable to set the environment for the job or set per-process limits for the process.

## 6.8 Managing Oom Killer for Cgroups

You can disable the OOM killer for cgroups by setting the `oom_kill_disable` memory configuration variable to *True*.

On recent kernels, the standard mechanism to configure the memory controller can cause spurious OOM killer interventions that kill a job.

Often jobs hit the memory limit with a mix of application pages and page cache content (i.e. file cache for files accessed by the application). Whenever newer pages are then requested (either for page cache for new file content to be written, or for the application), the page reclaimer is supposed to convert page cache back into usable memory (dirty page cache needs to be flushed to disk, and clean page cache needs to be discarded, converted into free memory).

On some recent kernels this mechanism breaks down for page cache for files on remote filesystems (e.g. on a Lustre filesystem), and the OOM killer is invoked before the page reclaiming mechanisms have had a chance to reclaim page cache to satisfy memory requests.

You can set the new `oom_kill_disable` configuration variable in the memory section to *True* so that an alternative control algorithm is used: the OOM killer is disabled for the cgroup, and processes hang when memory requests cannot be satisfied rather than be killed by the OOM killer.

To ensure real rogue jobs do not hang forever, the `exechost_periodic` event monitors jobs to see whether they are hung in this manner (which is reported by the OS through `under_oom` in the `memory.oom_control` pseudofile).

A check is made for `under_oom_kill_delay` seconds (default 5) to see whether `under_oom` persists. If `under_oom` remains *True* for the entire interval, the OOM killer is re-enabled if the amount of pagecache is less than `oom_cache_percent_kill` percent of the job's cgroup memory limit (default 5). The `under_oom_kill_delay` parameter is found in the memory section of the configuration file.

## 6.9 Caveats and Errors

### 6.9.1 Interactions Between Suspend/resume and the cpuset Subsystem

The cgroups hook is in general compatible with suspend/resume. But when using preemption via suspend and resume, unless vnodes are allocated exclusively to jobs in the class of preempting jobs, the class of preempted jobs, or both, since the scheduler is unaware of the CPU assignments made by MoM, it is possible for the scheduler to resume a low-priority job even though the job's cpuset still overlaps with that of a running high priority job.

To avoid this, use any of these methods:

- Disable the `cpuset` subsystem, and use only the `cpu` controller to limit excessive CPU resource usage by jobs. The lack of CPU isolation may still cause the jobs to interfere on some workloads.  
Also, not using the `cpuset` subsystem may require you to disable process pinning in your applications to share a vnode between more than one job, to allow the Linux CFS scheduler to move processes to CPU threads that are free.
- Ensure that each job in either the preempting workload or the preempted workload has exclusive access to all vnodes it uses.
- Use one of the recent hook events triggered on job resumption to either reject resumption of a job when a conflict is detected, or to migrate the cpusets to CPUs no longer assigned to active jobs. If migrating the cpuset, take a lock on the cgroups lock file.

## 6.9.2 Caveats for Shrinking a Job on a Host

If the `cpuset` subsystem is enabled, ensure that if the `cgroups` hook helps to shrink a job, no processes are running on the host.

## 6.9.3 Caveats for Using CUDA

To use CUDA version earlier than 7.0, you must allow access to all devices. Otherwise, the NVIDIA commands will fail. With CUDA 7.0 or greater, you do not need to allow access to all devices.

## 6.9.4 Do Not Change `ncpus` When `cpuset` Subsystem is Enabled

Do not change the value of the `ncpus` resource from that reported by MoM if the `cpuset` subsystem is enabled. Otherwise the `cgroups` hook will attempt to use CPUs that don't exist, and jobs will fail.

## 6.9.5 Cgroups Hook Prevents Epilogue from Running

If you are running the `cgroups` hook, any epilogue script will not run. The `cgroups` hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the `cgroups` hook, make your epilogue script into an `execjob_epilogue` hook instead. See [section 10.5.2, “Using Hooks for Prologue and Epilogue”, on page 462](#).

## 6.9.6 Errors

When a job is trying to access a Xeon Phi (**deprecated**) device on a vnode, but the device is not accessible by the job cgroup, you will see the following error in the job error output:

```
Error getting SCIF driver version
```



# Configuring PBS for Containers

## 7.1 Introduction

PBS supplies a built-in hook that runs jobs and applications inside containers. The hook launches separate container(s) for each job, and runs the job with the same submission or job script commands and environment as it would have outside the container(s). The same job environment variables are exported inside the container(s), and file staging and job output and error files are handled the same way as outside a container.

Users can run multi-vnode, multi-host, and interactive jobs in containers, and PBS tracks resource usage for these jobs. The PBS container(s) use cgroups to constrain the resources that the job can use, track resource usage, and pin and isolate resources. The PBS container(s) use the cgroups hook to assign GPUs correctly; see [Chapter 6, "Configuring and Using PBS with Cgroups", on page 311](#).

When the job finishes, PBS removes the container(s).

Jobs are matched to hosts running container daemons via a custom string array resource, which indicates which container engines are available on each host. You create this resource, and tell the container hook which engines are available on each host by setting the `container_resource_name` parameter in the hook's configuration file to the name of the custom string array resource. The default for the `container_resource_name` parameter is "container\_engine", so we recommend using that name when you create the resource.

Job submitters can specify the job container image by requesting it or setting the `CONTAINER_IMAGE` environment variable to the name of the container in which the job should run. A container request in the `container_image` resource overrides the `CONTAINER_IMAGE` environment variable. The PBS container hook looks for a container request and monitors job submissions for this environment variable, launches the appropriate container, and starts the job in the container.

You can configure a list of allowed registries, and set a default registry.

PBS can perform a registry login in order to pull from registries that require a login.

You can whitelist specific additional arguments to the container engine by listing them in the `container_args_allowed` container hook parameter. Job submitters can then specify any of these whitelisted arguments in the `PBS_CONTAINER_ARGS` environment variable.

You can configure the container hook so that it automatically adds job owners to additional groups inside Docker containers. The hook finds the groups on the execution host where the job owner is already a member, and adds the job owner to these groups inside the container. To do this, set the `enable_group_add_arg` container hook parameter to *True*. This feature applies only to Docker; Singularity users are automatically added to all groups inside containers. Note that for security, we recommend that you never whitelist the `--group-add` container argument in the `container_args_allowed` hook configuration parameter.

You can set permissions on files mounted inside containers, for example setting files in a container to be read-only.

## 7.1.1 Container Engines Used by PBS

A PBS server can create Docker and Singularity containers. Each job can specify which container engine to use, but can use only one container engine. You specify the default container engine by setting the `container_resource_default_value` parameter in the container hook's configuration file to either "docker" or "singularity". In addition, a user can always run a single-node job in a single Singularity container by prepending their scripts, executables, or commands with the Singularity binary.

### 7.1.1.1 Using nvidia-docker

PBS can invoke `nvidia-docker` if the `nvidia-docker-cmd` line in the hook's configuration file points to the location of the `nvidia-docker` command, and the job requests `ngpus` inside its select statement.

### 7.1.1.2 Caching Singularity Images

When an image is downloaded from the container hub, it is saved on the execution host in the cache path. You can set the cache path in the `container_cache_path` hook configuration parameter. The default value for this parameter is empty, in which case it is treated as if it is `<user home>/singularity/cache`.

PBS sets the value of the `SINGULARITY_CACHEDIR` Singularity environment variable to the value of `container_cache_path`. See the Singularity documentation for more information on this environment variable.

## 7.1.2 Container Ports

For single-vnode jobs in Docker containers, job submitters can request ports for applications. The container hook maps requested ports to available ports on the host and returns the mapping. You can define which port ranges are available for containers. The job submitter requests ports by listing comma-separated port numbers in the `container_ports` job resource. Lists of port numbers must be enclosed in single quotes. The hook sets the job's `resources_used.container_ports` value to comma-separated `<container port>:<host port>` pairs. For example, a job can request `-l container_ports='2324,8989'`, and the hook sets the job's `resources_used.container_ports` to `2324:8080,8989:32771`.

## 7.1.3 Managing How Files and Directories are Mounted in Containers

### 7.1.3.1 Setting Permissions on Mounted Files

You can set permissions on the files and directories mounted in both Docker and Singularity containers using the `mount_paths` container hook configuration parameter. The default value for the `mount_paths` parameter is `["/etc/passwd", "/etc/group"]`.

You can set multiple mounting paths in the `mount_paths` parameter. Syntax:

`"mount_paths": [<first path spec>, <second path spec>, ... <nth path spec>]`

where `<path spec>` can be any of:

- A single filename, indicating that source and target have the same name
- `["<source name>", "<target name>"]`, indicating no restriction
- `["<source name>", "<target name>", "<restriction>"]`, specifying restriction on file inside container

When you put more than one element in a path spec, for example both a source and a target, enclose the path spec in square brackets.

Example 7-1: Setting `/etc/passwd` to read-only for a Singularity container:

In the hook configuration file:

```
"mount_paths": [["/etc/passwd", "/etc/passwd", "ro"]]
```

PBS mounts `/etc/passwd` in the container with the destination `/etc/passwd`, and sets the permissions to `ro` (read-only) on the file inside container.

Example 7-2: Docker: setting multiple mount paths, with restriction

In the hook configuration file, we have two path specifications:

```
"mount_paths": [["/etc/passwd", "/etc/passwd", "readonly", "bind-propagation=rslave"],
 ["/etc/group"]]
```

The first path spec is `["/etc/passwd", "/etc/passwd", "readonly"]`, and PBS passes the following to Docker:

```
docker --mount type=bind,source=/etc/passwd,target=/etc/passwd,readonly,bind-propagation=rslave
```

The second path spec is `["/etc/group"]`, and PBS passes the following to Docker:

```
docker --mount type=bind,source=/etc/group,target=/etc/group
```

### 7.1.3.2 Allowing or Disallowing Job Work Directory Inside Container

It may be possible to submit a job from a place that is not sensitive on the submission host, but is sensitive on the compute hosts. You can allow or disallow mounting the job's work directory inside Docker or Singularity containers by setting the value of the `mount_jobdir` hook configuration parameter. Setting this to *True* enables mounting the job's work directory in containers. The default value of this parameter is *True*.

## 7.1.4 How PBS Uses Container Registries

You can configure a list of allowed (whitelisted) registries by listing them in the `allowed_registries` hook configuration parameter. PBS checks each job's registry specification against the whitelist. PBS uses only whitelisted registries. If the specified registry is not whitelisted, PBS rejects the job. If the job submitter does not specify a registry, PBS uses the default registry. You specify the default registry by making it the first entry in the `allowed_registries` parameter. Set the default registry according to the default container engine you are using (*docker* or *singularity*). For Singularity, set the default based on the value of `container_image_source` (*docker://* or *library://*).

You can whitelist all registries by including *PBS\_ALL* in the list.

The default value for `allowed_registries` is `["docker.io", "SylabsCloud", "PBS_ALL"]`.

### 7.1.5 Registry Credential File

PBS can pull images using job owner registry login credentials; this allows job owners to use images from registries where a login is required. If a registry allows you to pull without logging in, PBS allows this.

PBS stores login credentials in a JSON file, in a directory you specify. Make sure that the hook, which runs as root, can read the file.

Example 7-3: PBS uses the job owner's credentials to pull a container image:

```
qsub -v CONTAINER_IMAGE= myregistry.local/MyImage
```

PBS checks whether "myregistry.local" is included in the `allowed_registries` parameter.

If "myregistry.local" is included, PBS logs in using the credentials that are listed in `<job owner>/container/tokens.json` for myregistry.local.

If "myregistry.local" is not included, the job is rejected.

### 7.1.5.1 Registry Credential Filename

The credential filename has this format:

`<job owner>/container/tokens.json`

### 7.1.5.2 Registry Credential File Format

The file contents have this format:

```
{
 "registry1 <URL>/<endpoint>": {
 "user_id": "<user ID>", "passwd": "<generated OAUTH token/password>"
 },
 "registry2 <URL>/<endpoint>": {
 "user_id": "<user ID>", "passwd": "<generated OAUTH token/password>"
 }
}
```

### 7.1.5.3 Registry Credential File Default Values

*registry*: default registry (first element in the `allowed_registries` parameter)

*user\_id*: job owner; if this is empty, PBS tries instead with the job owner ID

*passwd*: no password

### 7.1.5.4 Registry Credential File Location

The registry credential file *base path* is the path to where registry credential files are stored, up to but not including `<job owner>/container/tokens.json`. The default base path to registry credential files is `/home`. You can configure the base path to where registry credential files are stored, by setting the value of the `cred_base_path` parameter in the hook configuration file.

Example 7-4: You set `cred_base_path` to `"/container/creds/"`, and your job owner is User1. The full path to the JSON file is:

`/container/creds/User1/container/tokens.json`

### 7.1.5.5 Docker Examples

Example 7-5: Job submission with image specification:

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/pbsuser/test-image
```

PBS looks for "pbsprohub.local" in the `allowed_registries` hook configuration parameter. If the registry is whitelisted, PBS looks for login credentials for the pbsprohub.local registry in the `<job owner>/container/tokens.json` file. PBS logs into the registry and pulls the requested container image.

If the login credentials for the pbsprohub.local registry are not listed in <job owner>/.container/tokens.json and the registry does not require a login, PBS skips logging in and pulls the requested container image.

If "pbsprohub.local" is not listed in the `allowed_registries` configuration parameter, PBS rejects the job.

Example 7-6: Job submission without image specification:

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/centos:7
```

PBS looks for "pbsprohub.local" in the `allowed_registries` hook configuration parameter. If the registry is whitelisted, PBS looks for login credentials for the pbsprohub.local registry in the <job owner>/.container/tokens.json file. PBS logs into the registry and pulls the requested container image.

If "pbsprohub.local" is not listed in `allowed_registries`, PBS uses the default registry. This can lead to possible failure at runtime.

Example 7-7: Job submission without registry specification:

```
qsub -v CONTAINER_IMAGE=pbsuser/test-image
```

Since no registry is specified, PBS uses the default registry. PBS uses the login credentials if they are listed in <job owner>/.container/tokens.json, and pulls <default registry>/pbsuser/test-image.

### 7.1.5.6 Singularity Examples

Example 7-8: Job submission without registry specification:

```
qsub -v CONTAINER_IMAGE=pbsuser/test-image
```

In the hook configuration file:

```
container_image_source = "docker://"
```

Since no registry is specified in the job request, PBS uses the default registry.

PBS uses the container image found at `docker://<default registry>/pbsuser/test-image`, and PBS uses login credentials if they are listed in <job owner>/.container/tokens.json.

Example 7-9: Job submission without defined endpoint:

```
qsub -v CONTAINER_IMAGE=pbspro/default/test-image
```

In the hook configuration file:

```
container_image_source = "library://"
```

If "pbspro" is not a defined endpoint, and is not listed in `allowed_registries`, PBS uses the default registry. Singularity remote endpoints enable secure container sharing. See the Singularity documentation about remote endpoints.

If "pbspro" is a defined endpoint, PBS checks the `allowed_registries` list, then uses the credentials in <job owner>/.container/tokens.json. PBS uses the "pbspro" endpoint, authenticating via the credentials in <job owner>/.container/tokens.json, by calling the command "singularity remote login".

Example 7-10: Path to image is "shub":

In the hook configuration file:

```
container_image_source = "shub://"
```

Because the Singularity hub is a public repository, PBS does not perform any authorization.

## 7.2 The PBS Container Hook

PBS has a built-in container hook named "PBS\_hpc\_container" which does several useful things:

- The `PBS_hpc_container` hook can create Docker and Singularity containers, and it can invoke `nvidia-docker` if it is configured and the job requests `ngpus` inside the select statement.
- The hook runs for the following events, with these actions:
  - At a `queuejob` event, the hook adds the name and desired value of the string array resource listing container engines to the job's select statement, if the job does not already specify it. This allows the scheduler to match the job to a host running the selected container daemon.
  - At a `queuejob` event, the hook checks the `allowed_registries` parameter. If there are whitelisted registries, it then looks for registry login credentials for the job owner.
  - At an `execjob_launch` event, the hook launches the job inside the selected container
- The hook starts a container instance from the requested image, and sets up the job's environment. The image is specified via `-lcontainer_image=<container image>` or in the job's `CONTAINER_IMAGE` environment variable. The hook uses the requested container engine, or if the job does not request a container engine, the hook uses the default set in the `container_resource_default_value` parameter in the hook's configuration file.
  - The name of the container is the job ID.
  - If the job is interactive, the hook runs the job in the container in interactive mode.
  - If the job has multiple chunks that are scheduled to run on a single host:
    - With Docker, the hook runs all of the job's child processes in one container on that host.
    - With Singularity, the hook runs each child process in its own container.
  - If the job runs on multiple hosts, the hook ensures that containers created on sister MoMs are network linked to the container running on the primary execution host.
- The hook updates the resources used by the job, and removes the job's container.
- The hook cleans up any orphaned containers left behind by previous jobs on the host.
- The hook can automatically add the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.

## 7.3 Prerequisites

The required container daemon(s) must be running on all hosts where users will run jobs in containers.

## 7.4 Configuring PBS for Containers

### 7.4.1 Create Container Resources

1. For the container image name, create a custom string resource named "container\_image":  
**Qmgr: create resource container\_image type=string,flag=m**
2. For the container port number, create a custom string array resource named "container\_ports":  
**Qmgr: create resource container\_ports type=string\_array,flag=m**
3. Create the custom string array resource that will list the container engines available on each host. We recommend naming it "container\_engine":  
**qmgr -c "create resource container\_engine type=string\_array,flag=mh"**
4. Set the value of the container\_engine resource on each host to the list of available container engines:  
*qmgr -c "s n node1 resources\_available.container\_engine=<list of container engines>"*  
 For example, if you have Docker on node1, and you have both Docker and Singularity on node2:  
**qmgr -c "s n node1 resources\_available.container\_engine=docker"**  
**qmgr -c "s n node2 resources\_available.container\_engine=docker"**  
**qmgr -c "s n node2 resources\_available.container\_engine += singularity"**
5. Add the container\_engine resource to the resources: line in PBS\_HOME/<sched\_priv directory>/sched\_config.
6. HUP the scheduler:  
**kill -HUP <scheduler PID>**

### 7.4.2 Configure PBS Container Hook

The container hook's configuration file allows parameters that are specific to each container engine.



### 7.4.2.1 Default Configuration File

```
{
 "container_resource_name": "container_engine",
 "container_resource_default_value": "docker",
 "mount_paths": ["/etc/passwd", "/etc/group"],
 "mount_jobdir": true,
 "cred_base_path": "",
 "allowed_registries": ["docker.io", "SylabsCloud", "PBS_ALL"],
 "docker":{
 "container_cmd": "/usr/bin/docker",
 "remove_env_keys": [],
 "port_ranges": [],
 "container_args_allowed": [],
 "enable_group_add_arg": false
 },
 "singularity":{
 "container_cmd": "/usr/local/bin/singularity",
 "container_image_source": "",
 "container_cache_path": "",
 "container_args_allowed": []
 }
}
```

The following table shows the parameters:

**Table 7-1: PBS Container Hook Configuration File Parameters**

| Parameter Name         | Default Value          | Description                                                                                                                                                                                                                       |
|------------------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| allowed_registries     | []                     | Whitelist of registries PBS is allowed to pull from. Put default registry first; PBS uses this when it cannot find the specified registry. Set this to <i>PBS_ALL</i> to allow all registries.                                    |
| container_args_allowed | []                     | Whitelist of arguments that job submitters are allowed to pass to container engine via the PBS_CONTAINER_ARGS environment variable.<br>Do not whitelist <code>--env</code> , <code>--entrypoint</code> , <code>--group-add</code> |
| container_cache_path   | []                     | For Singularity. Path to cache directory on execution host where singularity images are cached.                                                                                                                                   |
| container_cmd          | <i>/usr/bin/docker</i> | Path to container command. Can be <code>/usr/bin/docker</code> or <code>/usr/local/bin/singularity</code>                                                                                                                         |

Table 7-1: PBS Container Hook Configuration File Parameters

| Parameter Name                   | Default Value                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| container_image_source           | <code>[]</code>                            | Singularity only. Can be path to existing container image, or URI of container hub. Optional; job submitters can specify path to image.<br><br>Example: For an image with the path <code>/home/user1/singularity_images/centos_latest.sif</code> , set <code>container_image_source</code> to <code>["/home/user1/singularity_images/"]</code><br><br>Example Singularity hub: <code>["shub://"]</code><br><br>Example Docker hub where Singularity can fetch an image and convert it to SIF: <code>["docker://"]</code> |
| container_resource_default_value | <code>docker</code>                        | Default container engine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| container_resource_name          | <code>container_engine</code>              | Name of resource that lists available container engines on each host.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| cred_base_path                   | <code>[]</code>                            | Base path to user login credential file, where credential file is <code>&lt;user ID&gt;/ .container/tokens.json</code> . Default is empty, in which case PBS uses <code>/home</code> .                                                                                                                                                                                                                                                                                                                                   |
| enable_group_add_arg             | <code>false</code>                         | The hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.<br><br>Applies to Docker only; Singularity automatically adds job owners to all groups.                                                                                                                                                                                                                                                                         |
| mount_jobdir                     | <code>true</code>                          | Boolean for enabling or disabling mounting the job's working directory inside the container                                                                                                                                                                                                                                                                                                                                                                                                                              |
| mount_paths                      | <code>["/etc/passwd", "/etc/group"]</code> | Additional paths to mount into container at creation time, with additional options for security. For example <code>["/opt/mpich"]</code> , or <code>["/etc/passwd", "/etc/passwd", "ro"]</code>                                                                                                                                                                                                                                                                                                                          |
| nvidia_docker_cmd                | <code>None</code>                          | Path to <code>nvidia-docker</code> command                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| port_ranges                      | <code>[]</code>                            | Docker only. Comma-separated ranges of ports, for example <code>["2001-9999", "3500-4500", "7600-9500"]</code>                                                                                                                                                                                                                                                                                                                                                                                                           |
| remove_env_keys                  | <code>[]</code>                            | Docker only. List of environment variables not to export to job container                                                                                                                                                                                                                                                                                                                                                                                                                                                |

To configure your PBS container hook, export the configuration file, edit it, and re-import it.

1. Export the PBS container hook's configuration file:
 

```
#qmgr -c "export pbshook PBS_hpc_container application/x-config default" > container_config.json
```
2. Set global parameters in the PBS container hook configuration file to match your site. The configuration file must conform to JSON syntax.
  - Set the `allowed_registries` parameter to the list of allowed registries. See [section 7.1.4, “How PBS Uses Container Registries”, on page 357](#)
  - Set the `container_cmd` parameter to the path of the container command
  - Set the `container_resource_name` parameter to the name of the resource that lists available container engines, if you used a name other than "container\_engine"
  - Set the `container_resource_default_value` parameter to the default container engine, if you want it to be different from "docker"
  - Optionally create the credential file(s) `<job owner>/container/tokens.json` (job submitters may want to do this step)
  - Optionally set the `cred_base_path` parameter to the credential file path; see [section 7.1.5.4, “Registry Credential File Location”, on page 358](#)
  - If using Singularity, optionally set the `container_cache_path` parameter to the path where Singularity images will be stored. See [section 7.1.1.2, “Caching Singularity Images”, on page 356](#)
  - Optionally set the `container_image_source` parameter
  - Optionally set `mount_paths`; see [section 7.1.3.1, “Setting Permissions on Mounted Files”, on page 356](#)
  - Optionally disallow mounting the job's work directory in containers; see [section 7.1.3.2, “Allowing or Disallowing Job Work Directory Inside Container”, on page 357](#)
  - If using `nvidia-docker`, set `nvidia_docker_cmd`
  - If using Docker, set `port_ranges` to ranges of allowed ports on hosts
  - Optionally set `remove_env_keys`
  - Optionally set `container_args_allowed` to a whitelist of arguments that job submitters can pass to the container engine via the `PBS_CONTAINER_ARGS` environment variable.
    - Do not include "--entrypoint"; entry points are not supported
    - Do not include "--env"; this is not supported
    - Do not include "--group-add"; this poses security risks
  - Optionally set `enable_group_add_arg` to `True` so that the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.
3. For local users and local groups, include `/etc/passwd` and `/etc/group` in mount paths so the image has the host operating system users and groups defined.
4. To configure PBS to use `nvidia-docker`, make sure it is available in the path specified in the `nvidia-docker-cmd` line in the hook configuration file.

We show a sample configuration file here:

```
{
 "container_resource_name": "container_engine",
 "container_resource_default_value": "docker",
 "mount_paths": ["/etc/passwd", "/etc/group"],
 "mount_jobdir": true,
 "cred_base_path": "",
 "allowed_registries": ["docker.io", "SylabsCloud", "PBS_ALL"],
 "docker": {
 "container_cmd": "/usr/bin/docker",
 "remove_env_keys": [],
 "port_ranges": [],
 "container_args_allowed": [],
 "enable_group_add_arg": false
 },
 "singularity": {
 "container_cmd": "/usr/local/bin/singularity",
 "container_image_source": "",
 "container_cache_path": "",
 "container_args_allowed": []
 }
}
```

5. Re-import the PBS container hook's configuration file:

```
#qmgr -c "import pbshook PBS_hpc_container application/x-config default container_config.json"
```

6. Enable the PBS container hook:

```
#qmgr -c "set pbshook PBS_hpc_container enabled=True"
```

### 7.4.3 Install and Start Container Engines

Install and start Docker and/or Singularity on all hosts where you want to use them. Make sure that users are not part of any Docker groups. Consult documentation for your OS, Docker, and Singularity.

### 7.4.4 Configure Security Enhancement for Docker

We see a security shortcoming in Docker in the case where multiple users are on the same host, where users can see into each others' containers. We have implemented a security enhancement for this. We allow the job to run inside the container, but we don't add job submitters to the Docker group, and we don't allow job submitters to connect to the Docker container.

Our security enhancement for Docker integration allows jobs to run inside the container, but prevents job submitters from connecting to the Docker container. We use `pbs_container` to accomplish this.

Make `PBS_EXEC/sbin/pbs_container` a part of the Docker group. Set its SGID permissions:

```
chgrp docker PBS_EXEC/sbin/pbs_container
chmod 2755 PBS_EXEC/sbin/pbs_container
```

## 7.5 Caveats and Restrictions

- To run a shell in a container using anything besides the user's default, the job submitter must specify the shell using the `-S` option to `qsub`.
- Job submitters cannot use old-style resource requests such as `-lncpus` with containers.
- Any entry point in a container is disabled. If job submitters want to run an entry point command, they must include the complete command with its arguments on the command line.
- Make sure that when you are configuring the container hook, if you whitelist any container arguments in the `container_args_allowed` hook configuration parameter, do not whitelist `--group-add`. This would allow job submitters to add themselves to any groups inside the container. Instead, set the `enable_group_add_arg` hook parameter to `True` so the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.

## 7.6 Errors and Logging

Container creation errors are logged in the MoM log files. You can use `tracejob` to display these errors.

# Making Your Site More Robust

This chapter describes how to configure PBS to make your site more robust.

## 8.1 Robustness

PBS provides the following mechanisms that support site robustness and flexibility:

### Failover

The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.

### Checkpoint and Restart

Allows jobs to be checkpointed and restarted. Uses OS-provided or third-party checkpoint/restart facility.

### Reservation Fault Tolerance

PBS attempts to ensure that reservations run by finding usable vnodes when reservation vnodes become unavailable.

### Vnode Fault Tolerance for Job Start and Run

PBS lets you allocate extra vnodes at job startup or for the life of the job, to compensate for vnode failure and allow the job to successfully start or run on the required number of vnodes.

### Preventing Communication and Timing Problems

PBS allows setting parameters to prevent problems in communication, timing, and load on vnodes.

### Preventing File System Problems

PBS gives you tools to prevent file system problems.

### OOM Killer Protection

PBS is installed so that daemons are protected from an OOM killer.

## 8.2 Failover

### 8.2.1 Glossary

#### **Primary Server**

The PBS Professional server daemon which is running during normal operation.

#### **Secondary Server**

The PBS Professional server daemon which takes over when the primary server fails.

#### **Primary Scheduler**

The PBS Professional scheduler daemon which is running during normal operation.

#### **Secondary Scheduler**

The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

**Active**

A server daemon is active when it is managing user requests and communicating with the scheduler and MoMs.

**Idle**

A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

## 8.2.2 How Failover Works

During normal operation, the primary server is active and the secondary server is idle. If the primary server fails for any reason, the secondary server becomes active and takes over server functions for the complex. No work is lost during the transition between servers. PBS functions the same during failover as it does during normal operation. The PBS data service is considered to be part of the PBS server; if it fails, this triggers failover.

### 8.2.2.1 Primary and Secondary Schedulers

Each server is paired with and uses its own scheduler. If the secondary server becomes active, it starts its own scheduler.

### 8.2.2.2 Primary and Secondary Data Services

Each server is paired with and uses its own data service. If the secondary server becomes active, it starts its own data service.

### 8.2.2.3 Normal Post-configuration Behavior

After you have configured PBS for failover, and started both servers, the secondary server periodically attempts to connect to the primary server until it succeeds and registers itself with the primary server. The secondary server must be registered in order to take over upon failure of the primary server.



### 8.2.2.4 Behavior During Failover

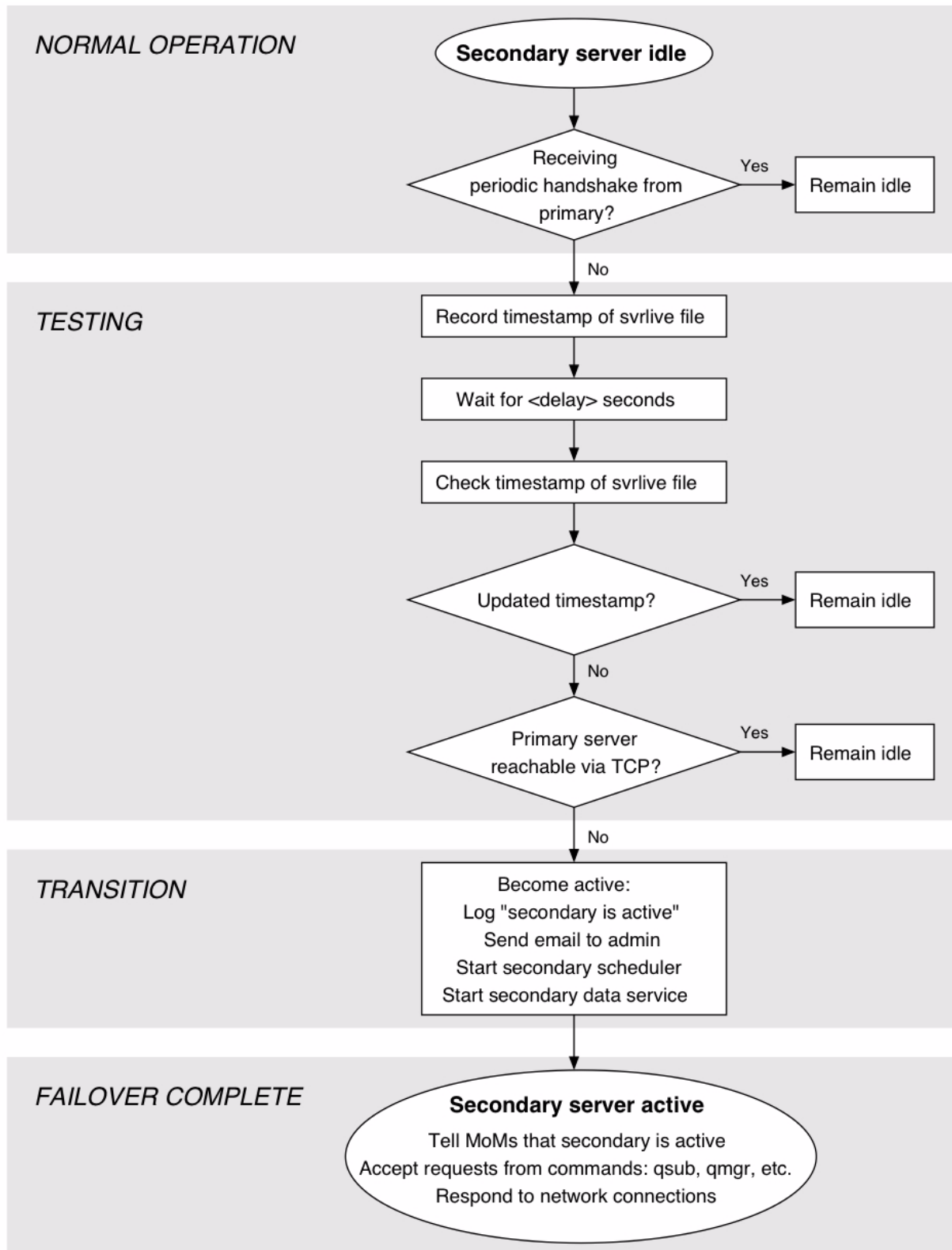


Figure 8-1: Behavior During Failover

When both server daemons are running, the primary server sends periodic handshake messages to the secondary. The primary server also periodically updates the timestamp of the `PBS_HOME/server_priv/svrlive` file. If the secondary server stops receiving handshake messages from the primary server, the following happens:

- The secondary server waits for a specified delay period before taking over. This delay is specified using the `pbs_server -F` option. The default period is 30 seconds.
  - The secondary server reads the timestamp of the `PBS_HOME/server_priv/svrlive` file and stores it in memory
  - The secondary waits for the specified delay, then checks the time stamp again, and compares it to the timestamp it stored in memory
  - If the timestamp has changed, the secondary server remains idle
  - If the timestamp has not changed, the secondary attempts to open a new TCP connection to the primary
  - If the secondary server cannot open a TCP connection to the primary, the secondary becomes active
- The secondary server logs a message saying that failover has occurred.
- An email is sent to and from the account defined in the server's `mail_from` attribute, saying that failover has occurred.
- The secondary server starts the secondary scheduler on the secondary server host.
- The secondary server starts the secondary data service on the secondary server host.
- The secondary server notifies all of the MoMs that it is the active server.
- The secondary server begins responding to network connections and accepting requests from client commands such as `qstat` and `qsub`.

### 8.2.2.5 Delay During Failover Transition

The default delay between when the primary becomes unavailable and the secondary takes over is about 5 minutes. You can change this using `pbs_server -F <seconds>`. If you use `pbs_server -F -1`, the secondary makes only one attempt to contact the primary, then takes over. We include these instructions in the configuration steps.

### 8.2.2.6 Behavior When Primary Resumes Control

When the primary server starts back up, it takes control from the secondary server, becoming the active server. The secondary server becomes idle and resumes listening for the regular handshake messages from the primary server.

The primary server may have been stopped for any of several reasons. The restart method will vary accordingly. If the host was stopped, the PBS server is restarted automatically when the host is started. If the host is still up but the server was stopped, restart the server. See [“Starting Servers With Failover” on page 148 in the PBS Professional Installation & Upgrade Guide](#).

The primary server uses only its own scheduler and data service. When the primary server resumes control, it starts a scheduler and data service, and stops the secondary scheduler and data service. No data is lost in the transition.

When the primary has taken control, the secondary logs a message saying so:

```
received takeover message from primary, going inactive
```

### 8.2.2.7 Server Name and Job IDs During Failover

The server name and job IDs do not change when the secondary server is active. For example, the primary server is on a host named *PrimaryHost.example.com*, and the secondary server is on a host named *SecondaryHost.example.com*. When the primary server is active, the server name is *PrimaryHost*, jobs are given job IDs of the form *NNNN.Primary-Host*, and the value of the `server_host` server attribute is *PrimaryHost.example.com*. When the secondary server is active, the server name is still *PrimaryHost*, jobs are still given job IDs of the form *NNNN.PrimaryHost*, but the value of `server_host` is *SecondaryHost.example.com*.

The table below summarizes the server name, value of `server_host` and the IDs given to jobs, when either the primary or secondary server is active.

**Table 8-1: Server Name, Job ID and Value of `server_host` Depending on Which Server is Active**

|                                   | Active Server                  |                                  |
|-----------------------------------|--------------------------------|----------------------------------|
|                                   | Primary                        | Secondary                        |
| Hostname                          | <i>PrimaryHost.example.com</i> | <i>SecondaryHost.example.com</i> |
| Server Name                       | <i>PrimaryHost</i>             | <i>PrimaryHost</i>               |
| Value of <code>server_host</code> | <i>PrimaryHost.example.com</i> | <i>SecondaryHost.example.com</i> |
| Job Name                          | <i>NNNN.PrimaryHost</i>        | <i>NNNN.PrimaryHost</i>          |

### 8.2.2.8 Information Used by Primary and Secondary Servers

The primary and secondary servers share a single source for attribute information, so anything set via the `qmgr` command need only be set once. `PBS_HOME` is in a shared location. License information is shared and needs to be set at only one server.

Each server, execution and client host uses its own `pbs.conf` file, so these must be set for each host in the complex.

### 8.2.2.9 Impact on Users

Users may not notice when a failover occurs. When a user uses a PBS command such as `qstat`, the command tries to connect to the primary server first. If that fails, the command tries the secondary server. There may be up to a two-minute delay in server commands while failover is taking place.

If the secondary server responds to the command, the command creates a local file so that this process is not repeated for every PBS command.

The file is named:

```
/tmp/.pbsrc.UID
```

where *UID* is the user ID.

When this file exists, commands try the secondary server first, eliminating the delay in attempting to connect to the down server. If a command cannot connect to the secondary server, and can connect to the primary server, the command removes the file.

The file is removed when the primary server takes over.

### 8.2.2.10 Determining Which Server Is Active

The server attribute `server_host` contains the name of the host on which the active server is running. Use the `qstat -Bf` command to see the value of `server_host`.

### 8.2.2.11 Delay Between Primary Failure and Secondary Becoming Active

The default delay time from detection of possible primary server failure until the secondary server takes over is 30 seconds. A secondary server on a very reliable network can use a shorter delay. A secondary server on an unreliable network may need to use a longer delay. The delay is specified via the `-F` option to the `pbs_server` command.

### 8.2.2.12 Communication

If PBS is configured for failover, each server host runs a `pbs_comm`. Note that communication traffic is handled independently of failover behavior. During normal operation, the comm on the primary server host handles communication traffic, but if that comm becomes unavailable, the comm on the secondary automatically takes over the communication traffic. You do not need to perform any configuration to get this behavior; the communication daemons are automatically configured for you. See [“Failover and Communication Daemons” on page 53 in the PBS Professional Installation & Upgrade Guide](#).

#### 8.2.2.12.i Communication with MoMs

- If a MoM will see different server addresses, add a `$clienthost` entry to MoM's configuration file for each possible server address.
- The secondary server is automatically added to the list of hosts allowed to connect to MoMs, in the `$clienthost` MoM configuration parameter.

### 8.2.3 Windows Locations

PBS is installed on Windows systems in `\Program Files (x86)\PBS\`.

### 8.2.4 Prerequisites for Failover

#### 8.2.4.1 Checklist of Prerequisites for Failover

The following table contains a checklist of the prerequisites for failover. Each entry has a link to more detailed information about the entry.

**Table 8-2: Prerequisites for Failover**

| Prerequisite                                                                                                                                           | Explanation                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Identical server hosts                                                                                                                                 | See <a href="#">section 8.2.4.2, “Server Host Requirements”, on page 373</a>                    |
| MoMs on server hosts don't share a <code>mom_priv</code> directory                                                                                     | See <a href="#">section 8.2.4.3, “Requirements for MoMs on Server Hosts”, on page 373</a>       |
| All hosts must be able to communicate over the network                                                                                                 | See <a href="#">section 8.2.4.4, “Ensuring Communication Between Hosts”, on page 374</a>        |
| All hosts must be able resolve hostnames of other hosts in complex                                                                                     | See <a href="#">section 8.2.4.5, “Hostname Resolution”, on page 374</a>                         |
| Filesystem must be shared, on a separate host from either server host, and provide features required for failover; no root squash on shared filesystem | See <a href="#">section 8.2.4.6, “Shared Filesystem”, on page 374</a>                           |
| On systems using <code>systemd</code> , monitoring and automatic restart of PBS daemons must be disabled.                                              | See <a href="#">section 8.2.4.7, “Prevent Automatic Daemon Restart by systemd”, on page 376</a> |
| Administrator must have access to filesystem from both server hosts                                                                                    | See <a href="#">section 8.2.4.8, “Permission Requirements”, on page 376</a>                     |
| Same version of PBS for all components                                                                                                                 | See <a href="#">section 8.2.4.9, “Same PBS Versions Everywhere”, on page 376</a>                |

Table 8-2: Prerequisites for Failover

| Prerequisite                                                                                                             | Explanation                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| Primary server's scheduler must be able to run when primary server runs                                                  | See <a href="#">section 8.2.4.10, “Requirement for Scheduler”, on page 376</a>                      |
| Data service user account must be the same on both primary and secondary server hosts                                    | See <a href="#">section 8.2.4.11, “Same Data Service Account on Both Server Hosts”, on page 376</a> |
| Data service host must be default                                                                                        | See <a href="#">section 8.2.4.12, “Data Service Host Configuration Requirement”, on page 376</a>    |
| usernames must be consistent across primary & secondary servers hosts                                                    | See <a href="#">section 8.2.4.13, “Consistent Usernames”, on page 377</a>                           |
| The <code>mail_from</code> server attribute specifies an email address that is monitored. Not required, but recommended. | See <a href="#">section 8.2.4.14, “Monitor Server Mail”, on page 377</a>                            |

### 8.2.4.2 Server Host Requirements

The primary and secondary servers must run on two separate host machines. Both host machines must have the same architecture. They must be binary compatible, including word length, byte order, and padding within structures. There must be exactly one primary and one secondary server.

On an HPE 8600, use two different service nodes to run the primary and secondary servers.

### 8.2.4.3 Requirements for MoMs on Server Hosts

You can run a MoM on both the primary and secondary server hosts, but this is **not** recommended.

If a MoM is to run on both server hosts, the two MoMs must not share the same `PBS_HOME/mom_priv` directory. In addition, it is strongly recommended that the following be true:

- The `mom_priv` directory structure be replicated on a local, non-shared, filesystem. On Windows, MoM already has a local directory on each server host. On Linux, you must create these.

Replicate the `mom_priv` and `mom_logs` directory structures on the primary server host if they don't exist there already. You must put these in the same location. Do the following on the primary server host:

```
scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
```

Replicate the `mom_priv` and `mom_logs` directory structures on the secondary server host if they don't exist there already. You must put these in the same location. Do the following on the secondary server host:

```
scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
```

- Each MoM use its own, local, `mom_priv` directory structure

The `PBS_MOM_HOME` entry in `pbs.conf` specifies the location that contains the `mom_priv` and `mom_logs` directories. If `PBS_MOM_HOME` is specified in `pbs.conf`, `pbs_mom` uses that location instead of `PBS_HOME`.

---

To prevent the MoMs from automatically using the same directory, do one of the following:

- Recommended: Specify the separate, local PBS\_MOM\_HOME entry in each server host's `pbs.conf` file (each `pbs_mom` will use the location for `mom_priv` specified in its `PBS_MOM_HOME`). Give the location of the local `PBS_HOME/mom_priv` that you replicated on each host. You can perform this step now, or later, when editing `pbs.conf` on each server host, in [section 8.2.5.3, “Host Configuration for Failover on Linux”, on page 380](#), or [section 8.2.5.4, “Host Configuration for Failover on Windows”, on page 384](#).
- Use the `-d` option when starting at least one `pbs_mom` to specify that they use the local, non-default locations for `mom_priv`

### 8.2.4.4 Ensuring Communication Between Hosts

Both the primary and secondary server hosts must be able to communicate over the network with each other and all execution hosts.

Beware of dependencies on remote file systems: The `$PBS_CONF_FILE` environment variable must point to `pbs.conf`. PBS depends on the paths in `pbs.conf` being available when its start/stop script is executed. PBS will hang if a remote file access hangs, and normal privileges don't necessarily carry over for access to remote file systems. For example, a FAT filesystem mounted via NFS won't support permissions.

### 8.2.4.5 Hostname Resolution

Hostname resolution must work between each host in the PBS complex. **Make sure that all hosts in the complex** (the primary and secondary server hosts, the file server host, and all execution and client hosts) **are set up so that they can resolve the names of all other hosts in the complex**. If you are not sure whether hostname resolution is working, run the `pbs_hostn` command at each host, testing the hostnames of the other hosts. The `pbs_hostn` command will return the canonical hostname of the specified host.

### 8.2.4.6 Shared Filesystem

The filesystem you use for the machines managed by PBS should be highly reliable. We recommend, in this order, the following filesystems:

- HA DAS
- DAS, such as `xfs` or `gfs`
- HA NFS
- NFS

---

PBS\_HOME is the top directory used by the PBS server. The primary and secondary servers share the same PBS\_HOME directory. The PBS\_HOME directory must conform to the following:

- The PBS\_HOME directory must be available under the same name to both the primary and secondary server hosts.
- The PBS\_HOME directory must be on a file system which meets the following requirements:
  - It should reside on a different machine from either of the server hosts.
  - It must be shared by the primary and secondary server hosts.
  - It must be reliable. The file system must be always available to both the primary and secondary servers. A failure of the file system will stop PBS from working.
  - The file system protocol must provide file locking support.
  - The file locking daemons must be running.
  - For Linux, the filesystem must support POSIX (Open Group) file semantics.
  - It must support concurrent read and write access from two hosts.
  - It must support multiple export/mounting.
  - No root squash on the shared filesystem.

If your filesystem does not conform to the specifications above, follow the steps in the next sections.

#### 8.2.4.6.i Using NFS Filesystems

When using NFS for PBS\_EXEC, NFS must be configured to allow root access and to allow `setuid-root` programs to execute from it.

If possible, mount NFS file systems synchronously (without caching) to avoid reliability problems.

NFS filesystems should be hard mounted.

#### 8.2.4.6.ii Setting Up the Shared Filesystem

You can use NFS or another filesystem protocol to set up the shared filesystem on which PBS\_HOME resides. Examples are Lustre, IBM GPFS, and Red Hat GFS. Make sure your protocol supports:

- Multiple export/mounting
- Simultaneous read/write from two hosts
- File locking support

To set up your file system:

1. Choose a machine for the file server host. This machine must not be either of the server hosts.
2. Make sure the file system is mounted by both the primary and secondary server hosts. For NFS, make sure the file system is hard mounted by both hosts.
3. Make sure the file system can provide file locking. For NFS, the lock daemon, `lockd`, must be running.
4. Make sure that PBS\_HOME is available under the same name to both the primary and secondary server hosts.



---

### 8.2.4.7 Prevent Automatic Daemon Restart by systemd

On systems running recent versions of `systemd`, make sure that you disable monitoring and automatic daemon restart by `systemd`. Otherwise, when the secondary takes over and kills the primary, `systemd` can detect the daemon's disappearance from the primary and restart it there, at the worst possible moment (while the secondary is recovering nodes, queues, and jobs from the datastore and is thus unresponsive to the primary). This can lead to split brain situations in which the datastore is corrupted, when the primary times out waiting for the secondary and starts even though the secondary is in the process of spinning up. To prevent the problem, use one of the following methods to restart the server:

- Prevent the secondary from starting when you restart the primary:
  - a. Stop the secondary server
  - b. Issue `systemctl restart pbs`
  - c. After the primary is responsive (`qstat -Bf` returns output), restart the secondary server
- Use failover:
  - a. Stop but do not restart the primary, and let failover happen
  - b. After the secondary has taken over (`qstat -Bf` returns output) you can leave the server running there, or take back the services on the primary by starting it again; note that this causes an extra interruption of services

Never start the primary unless you are sure that the secondary is either not running or responds to commands (i.e. `qstat -Bf` returns).

### 8.2.4.8 Permission Requirements

The `PBS_HOME` directory must meet the security requirements of PBS. Each parent directory above `PBS_HOME` must be owned by root and writable by root only.

The `PBS_HOME` directory must be readable and writable from both server hosts by the [PBS Administrator](#).

### 8.2.4.9 Same PBS Versions Everywhere

Both server hosts, all the execution hosts, and all the client hosts must run the same version of PBS Professional.

### 8.2.4.10 Requirement for Scheduler

The primary scheduler must be able to run whenever the primary server is running, and the secondary scheduler must be able to run when the secondary server is running. If a server becomes active but cannot use its own scheduler, PBS will not be able to schedule jobs.

### 8.2.4.11 Same Data Service Account on Both Server Hosts

The PBS [Data service management account](#) must be the same on both server hosts. The UID of the PBS data service management account must be identical on both the primary and secondary server hosts. We recommend that the PBS data service management account is called `pbsdata`.

If you change either data service management account, both must be changed at the same time and both servers must be restarted. The name of the [Data service account](#) must be the same as the data service management account.

### 8.2.4.12 Data Service Host Configuration Requirement

The `DATA_SERVICE_HOST` parameter must not be set in `pbs.conf`. If this parameter is set, failover cannot take place.

### 8.2.4.13 Consistent Usernames

Username must be consistent across the primary and secondary server hosts. If usernames are not consistent, jobs are killed.

### 8.2.4.14 Monitor Server Mail

Use the `qmgr` command to set the `mail_from` server attribute to an address that is monitored regularly:

```
Qmgr: s server mail_from=<address>
```

See [section 2.2.2, “Configuring Server Mail Address”, on page 22](#).

## 8.2.5 Configuring Failover

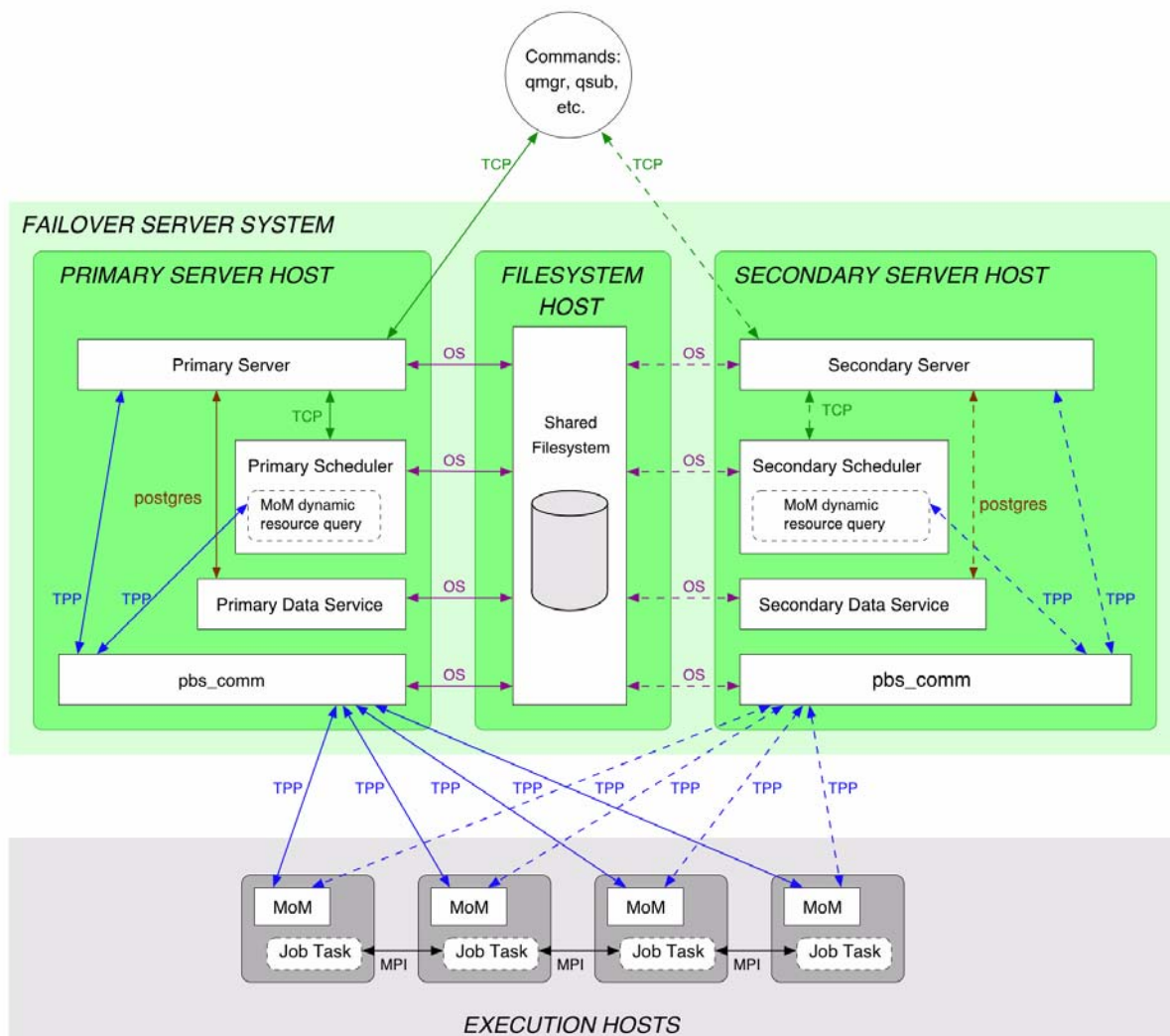


Figure 8-2: Failover Configuration

### 8.2.5.1 Overview of Configuring Failover

If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

Please make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).

Make a copy of your PBS configuration. Follow the instructions in [“Back Everything Up to Transfer Location” on page 99 in the PBS Professional Installation & Upgrade Guide](#).

The following table contains a guide to the steps in configuring PBS for failover. The table contains a link to the description of each step.

**Table 8-3: Overview of Configuring Failover**

| Step                                                             | Linux                                                                                                               | Windows                                                                                                              |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| Configure <code>/etc/pbs.conf</code> on each host in the complex | See <a href="#">section 8.2.5.2, “Configuring the pbs.conf File for Failover”, on page 378</a>                      |                                                                                                                      |
| Configure the primary server                                     | See <a href="#">section 8.2.5.3.i, “Configuring Failover For the Primary Server on Linux”, on page 380</a>          |                                                                                                                      |
| Configure the secondary server                                   | See <a href="#">section 8.2.5.3.ii, “Configuring Failover For the Secondary Server on Linux”, on page 382</a>       |                                                                                                                      |
| Recommended: configure STO-NITH script                           | See <a href="#">section 8.2.5.3.iii, “Configuring STO-NITH Script for Use by Secondary Server”, on page 382</a>     |                                                                                                                      |
| Configure execution and client hosts                             | See <a href="#">section 8.2.5.3.iv, “Configuring Failover For Execution and Client Hosts on Linux”, on page 383</a> | See <a href="#">section 8.2.5.4.i, “Configuring Failover for Execution and Client Hosts on Windows”, on page 384</a> |
| Configure failover with peer scheduling                          | See <a href="#">section 8.2.6.2, “Configuring Failover to Work With Peer Scheduling”, on page 384</a>               |                                                                                                                      |
| Configure failover with routing queues                           | See <a href="#">section 8.2.6.1, “Configuring Failover to Work with Routing Queues”, on page 384</a>                |                                                                                                                      |
| Configure failover with access control                           | See <a href="#">section 8.2.6.3, “Configuring Failover to Work With Access Controls”, on page 385</a>               |                                                                                                                      |

### 8.2.5.2 Configuring the `pbs.conf` File for Failover

The `$PBS_CONF_FILE` environment variable contains the path to the `pbs.conf` file. Each host in the complex must have a properly configured `/etc/pbs.conf` file. This file specifies the hostnames of the primary and secondary servers, the location of `PBS_HOME` and `PBS_MOM_HOME`, and whether to start a server, a scheduler, or a MoM on this host.

The name used for the server in the `PBS_SERVER` variable in the `pbs.conf` file must not be longer than 255 characters. If the short name for the server resolves to the correct host, you can use this in `pbs.conf` as the value of `PBS_SERVER`. However, if the fully-qualified domain name is required in order to resolve to the correct host, then the this must be the value of the `PBS_SERVER` variable.

**Table 8-4: Parameters in `pbs.conf` for Failover**

| Parameters                                  | Value                | Meaning                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------------------------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>PBS_CLIENT_CONN_RETRY_INTERVAL</code> | Seconds              | Interval in seconds at which <code>qsub</code> attempts to connect to the server before timing out.<br>Default: <i>30</i>                                                                                                                                                                                                                                                                     |
| <code>PBS_CLIENT_CONN_RETRY_TIMEOUT</code>  | Seconds              | Maximum time in seconds that <code>qsub</code> will try to connect to the server.<br>When this parameter is $> 0$ , the parameter specifies the timeout.<br>When this parameter is 0, this parameter is disabled, and the timeout reverts to the default of 600.<br>Default: <i>0</i> (disabled)                                                                                              |
| <code>PBS_EXEC</code>                       | Path                 | Location of PBS <code>bin</code> and <code>sbin</code> directories                                                                                                                                                                                                                                                                                                                            |
| <code>PBS_HOME</code>                       | Path                 | Location of PBS working directories in shared filesystem; use specific path on that host                                                                                                                                                                                                                                                                                                      |
| <code>PBS_MOM_HOME</code>                   | Path                 | Location of <code>mom_priv</code> on each host; overrides <code>PBS_HOME</code> for <code>mom_priv</code>                                                                                                                                                                                                                                                                                     |
| <code>PBS_PRIMARY</code>                    | FQDN of hostname     | Hostname of primary server host.<br>If you set <code>PBS_LEAF_NAME</code> on the primary server host, make sure that <code>PBS_PRIMARY</code> matches <code>PBS_LEAF_NAME</code> on the corresponding host. If you do not set <code>PBS_LEAF_NAME</code> on the server host, make sure that <code>PBS_PRIMARY</code> matches the hostname of the server host.                                 |
| <code>PBS_SECONDARY</code>                  | FQDN of hostname     | Hostname of secondary server host.<br>If you set <code>PBS_LEAF_NAME</code> on the secondary server host, make sure that <code>PBS_SECONDARY</code> matches <code>PBS_LEAF_NAME</code> on the corresponding host. If you do not set <code>PBS_LEAF_NAME</code> on the server host, make sure that <code>PBS_SECONDARY</code> matches the hostname of the server host.                         |
| <code>PBS_SERVER</code>                     | Hostname             | Name of primary server host. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the <code>PBS_SERVER</code> entry in <code>pbs.conf</code> . If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of <code>PBS_SERVER</code> . |
| <code>PBS_START_COMM</code>                 | <i>0</i> or <i>1</i> | Specifies whether a comm is to run on this host                                                                                                                                                                                                                                                                                                                                               |
| <code>PBS_START_MOM</code>                  | <i>0</i> or <i>1</i> | Specifies whether a MoM is to run on this host                                                                                                                                                                                                                                                                                                                                                |
| <code>PBS_START_SCHED</code>                | <i>0</i> or <i>1</i> | Specifies whether scheduler is to run on this host                                                                                                                                                                                                                                                                                                                                            |
| <code>PBS_START_SERVER</code>               | <i>0</i> or <i>1</i> | Specifies whether server is to run on this host                                                                                                                                                                                                                                                                                                                                               |

### 8.2.5.2.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 8.2.5.3 Host Configuration for Failover on Linux

- Make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).
- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution hosts. The client commands should already be installed on the client hosts.
- Make root a Manager on both server hosts:  

```
qmgr -c "set server managers =root@<primary server host>"
qmgr -c "set server managers +=root@<secondary server host>"
```
- If the primary server and scheduler are running, shut them down. See [“qterm” on page 236 of the PBS Professional Reference Guide](#).

#### 8.2.5.3.i Configuring Failover For the Primary Server on Linux

1. Make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).

2. Stop PBS on both the primary and secondary server hosts:

On the primary server host:

```
systemctl stop pbs
```

or

```
<path to init.d>/init.d/pbs stop
```

On the secondary server host:

```
systemctl stop pbs
```

or

```
<path to init.d>/init.d/pbs stop
```

3. On the primary server host, edit the `/etc/pbs.conf` file so that it DOES NOT include failover settings. It should look like this:

```
PBS_SERVER=<short name for primary host>
```

```
PBS_HOME=<shared location of PBS_HOME>
```

```
PBS_START_SCHED=1
```

We recommend not running a MoM on any server host. The following setting in `pbs.conf` will prevent a MoM from running:

```
PBS_START_MOM=0
```

If you will run a MoM on the server hosts, specify this:

```
PBS_START_MOM=1
```

If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in [section 8.2.4.3, “Requirements for MoMs on Server Hosts”, on page 373](#):

```
PBS_MOM_HOME=<location of local, replicated mom_priv>
```

4. On the primary server host, start the primary PBS server and scheduler daemons:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

5. Stop the PBS server on the primary server host:

```
systemctl stop pbs
```

or

```
<path to init.d>/init.d/pbs stop
```

6. On the primary server host, edit the `/etc/pbs.conf` file to include the failover settings for `PBS_PRIMARY` and `PBS_SECONDARY`. It should look like this:

```
PBS_PRIMARY=<primary_host>
```

```
PBS_SECONDARY=<secondary_host>
```

```
PBS_SERVER=<short name for primary host>
```

```
PBS_HOME=<shared location of PBS_HOME>
```

The primary scheduler will start automatically:

```
PBS_START_SCHED=1
```

We recommend not running a MoM on any server host. The following setting in `pbs.conf` will prevent a MoM from running:

```
PBS_START_MOM=0
```

If you will run a MoM on the server hosts, specify this:

```
PBS_START_MOM=1
```

If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in [section 8.2.4.3, “Requirements for MoMs on Server Hosts”, on page 373](#):

```
PBS_MOM_HOME=<location of local, replicated mom_priv>
```

If you set `PBS_LEAF_NAME` on the primary server host, make sure that `PBS_PRIMARY` matches `PBS_LEAF_NAME` on the corresponding host. If you do not set `PBS_LEAF_NAME` on the server host, make sure that `PBS_PRIMARY` matches the hostname of the server host.

If you set `PBS_LEAF_NAME` on the secondary server host, make sure that `PBS_SECONDARY` matches `PBS_LEAF_NAME` on the corresponding host. If you do not set `PBS_LEAF_NAME` on the server host, make sure that `PBS_SECONDARY` matches the hostname of the server host.

7. Run a comm on the primary server host. Set the following in `pbs.conf` on the primary server host:

```
PBS_START_COMM = 1
```

8. On the primary server host, start the primary PBS server, scheduler, and comm daemons:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

### 8.2.5.3.ii Configuring Failover For the Secondary Server on Linux

1. Make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).
2. On the secondary server host, edit the `/etc/pbs.conf` file to include the following settings:

```
PBS_PRIMARY=<primary_host>
PBS_SECONDARY=<secondary_host>
PBS_SERVER=<short name for primary host>
PBS_HOME=<shared location of PBS_HOME>
```

The secondary server will start its own scheduler if it needs to; a scheduler should not automatically start on the secondary server host. Include the following so that a scheduler does not automatically start on this host:

```
PBS_START_SCHED=0
```

We recommend not running a MoM on any server host. The following setting in `pbs.conf` will prevent a MoM from running:

```
PBS_START_MOM=0
```

If you will run a MoM on the server hosts, specify this:

```
PBS_START_MOM=1
```

If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in [section 8.2.4.3, “Requirements for MoMs on Server Hosts”, on page 373](#):

```
PBS_MOM_HOME=<location of local, replicated mom_priv>
```

If you set `PBS_LEAF_NAME` on the primary server host, make sure that `PBS_PRIMARY` matches `PBS_LEAF_NAME` on the corresponding host. If you do not set `PBS_LEAF_NAME` on the server host, make sure that `PBS_PRIMARY` matches the hostname of the server host.

If you set `PBS_LEAF_NAME` on the secondary server host, make sure that `PBS_SECONDARY` matches `PBS_LEAF_NAME` on the corresponding host. If you do not set `PBS_LEAF_NAME` on the server host, make sure that `PBS_SECONDARY` matches the hostname of the server host.

3. On the secondary server host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use the `-F <delay>` option on the secondary server's command line in the PBS start script on the secondary server host. Edit the `init.d/pbs` script so that the server is invoked with the `-F <delay>` option:

```
pbs_server -F <delay>
```

See [“pbs\\_server” on page 107 of the PBS Professional Reference Guide](#).

4. Run a comm on the secondary server host. Set the following in `pbs.conf` on the secondary server host:

```
PBS_START_COMM = 1
```

5. On the secondary server host, start the secondary PBS server and comm daemons:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

### 8.2.5.3.iii Configuring STONITH Script for Use by Secondary Server

We strongly recommend that before the secondary server becomes active, it prevents a race condition between the primary and secondary data services by calling a script which shuts down the primary server host. This script is called STONITH, for "shoot the other node in the head". If the script returns failure, the secondary server waits for 10 seconds, then calls the script again. The secondary server does not become active until the script returns success.



Requirements for STONITH:

- You must write the STONITH script, and put it in `$PBS_HOME/server_priv/stonith`.
- Permissions for the script should be `0755`.
- The STONITH script takes one argument, which is the hostname of the primary server. This hostname is the same as what is listed for `PBS_PRIMARY` in `pbs.conf`.
- The STONITH script returns zero for success, and non-zero for failure.

Note that you must supply the command used to power down the primary server host.

Example 8-1: Sample STONITH Script

```
#!/bin/bash
This script powers down the primary server host.
This script runs only on the secondary server host.
PBS_PRIMARY=$1
SECONDARY=`hostname`
POWERDOWN_CMD="<command to power down the primary server host>"

echo "INFO: Secondary starting Stonith script. Secondary server host is ${SECONDARY}."
echo "INFO: This Stonith script will power down the primary server host."
echo "INFO: Primary server host is ${PBS_PRIMARY}."

Power down the primary server host
You can also include a timeout, and check the value of the result.
Example: timeout_result=$({ timeout 10 ${POWERDOWN_CMD} ${PBS_PRIMARY} ; } 2>&1)
${POWERDOWN_CMD} ${PBS_PRIMARY}

if [$? -eq 0] ; then
 echo "INFO: Stonith script succeeded in powering down primary server host ${PBS_PRIMARY}."
 exit 0
else
 echo "ERROR: Stonith script failed to power down primary server host ${PBS_PRIMARY}."
 exit 1
fi
```

### 8.2.5.3.iv Configuring Failover For Execution and Client Hosts on Linux

1. Make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).
2. On each execution or client host, configure the `/etc/pbs.conf` file to include the following parameters:

```
PBS_PRIMARY=<primary_host>
PBS_SECONDARY=<secondary_host>
PBS_SERVER=<short name for primary host>
PBS_HOME=<location of PBS_HOME>
```

The `pbs.conf` files on execution hosts are already configured to start the MoM daemon only. Similarly, the `pbs.conf` files on client hosts are already configured to start no daemons.

3. On each execution host, restart the MoM:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

## 8.2.5.4 Host Configuration for Failover on Windows

### 8.2.5.4.i Configuring Failover for Execution and Client Hosts on Windows

1. Make sure that you have satisfied all of the prerequisites under [section 8.2.4, “Prerequisites for Failover”, on page 372](#).

2. On each execution or client host, specify the location of PBS\_HOME for the primary server:

```
pbs-config-add "PBS_HOME=\\<shared filesystem host>\pbs_home"
```

3. On each execution or client host, specify the primary and secondary server names in the pbs.conf file by running the following commands:

```
pbs-config-add "PBS_SERVER=<short name of primary server host>"
```

```
pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
```

```
pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"
```

4. If this is an execution host, restart the MoM:

```
net start pbs_mom
```

## 8.2.6 Configuring Failover with Other PBS Features

### 8.2.6.1 Configuring Failover to Work with Routing Queues

You must configure failover to work with routing queues which have destinations in another complex. No additional configuration is required for routing queues which have destinations in the same complex.

For a routing queue in one complex which points to a queue *Q1* in another PBS complex that is set up for failover, it is a good idea to specify both *Q1@primary.example.com* and *Q1@secondary.example.com* as destinations.

For example, if a routing queue has a destination queue at another complex's primary server:

```
Qmgr: set queue r66 route_destinations=workq@primary.example.com
```

you need to add the same queue at the other complex's secondary server:

```
Qmgr: set queue r66 route_destinations+=workq@secondary.example.com
```

See [section 2.3.6, “Routing Queues”, on page 27](#).

### 8.2.6.2 Configuring Failover to Work With Peer Scheduling

For peer queueing where the furnishing complex is set up for failover:

- You must list the furnishing queue at both primary and secondary servers. If the furnishing queue is *Q1*, the `peer_queue` line in the pulling complex's `sched_config` file must list *Q1@primary.example.com* and *Q1@secondary.example.com*

For peer queueing where the pulling complex is set up for failover:

- You must add `<manager>@primary.example.com` and `<manager>@secondary.example.com` to the list of managers at the furnishing server.

See [section 4.9.31, “Peer Scheduling”, on page 164](#).

### 8.2.6.3 Configuring Failover to Work With Access Controls

If you are using access control on the server (the `acl_host_enable` server attribute is set to *True* and the `acl_hosts` server attribute is specified), add the secondary server to the host list in `acl_hosts`:

```
Qmgr: s server acl_hosts+=<secondary server host>
```

See [section 11.3.4, “ACLs”, on page 493](#).

## 8.2.7 Using PBS with Failover Configured

### 8.2.7.1 Stopping Servers

To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

```
qterm -f
```

To stop the primary server and leave the secondary server idle:

```
qterm -i
```

To stop the secondary server only:

```
qterm -F
```

### 8.2.7.2 Starting Servers

After configuring the servers, you can start them in any order.

If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it informs the secondary that the secondary can become idle.

However, if there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

```
qterm -F
pbs_server
```

To restart the secondary server while it is the active server:

```
pbs_server -F -l
```

The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.

See [“pbs\\_server” on page 107 of the PBS Professional Reference Guide](#) and [“qterm” on page 236 of the PBS Professional Reference Guide](#).

## 8.2.8 Recommendations and Caveats

### 8.2.8.1 Starting and Stopping PBS

- **Do not** start or stop the data service using anything except the `pbs_dataservice` command. Start or stop the data service using only the `pbs_dataservice` command.
- If you do not wish for the secondary server to take over, use the `-i` option to the `qterm` command when stopping the primary server.
- When the primary server is active, and the secondary server is running and idle, the `pbs start/stop` script stops the active server, but leaves the idle server running. This means that the idle server becomes the active server.

---

### 8.2.8.2 Hosts

- `PBS_HOME` should not be on either server host
- Neither PBS server should be the NFS fileserver

### 8.2.8.3 Connecting to the Server

- If the "*take over*" delay time specified with the `pbs_server -F` option is too long, there may be a period, up to that amount of time, when clients cannot connect to either server.
- If the "*take over*" delay time specified with the `pbs_server -F` option is too short and there are transient network failures, then the secondary server may attempt to take over while the primary server is still active.
- While the primary server is active and the secondary server is inactive, the secondary server will not respond to any network connection attempts. Therefore, you cannot status the secondary server to determine whether it is running.

### 8.2.8.4 Allowing qsub to Withstand Failover

- Consider setting the timeout allowed for `qsub` to connect to the server to a value that can withstand failover. See [section 8.6.4, "Configuring Timeout for qsub Connection", on page 414](#).

### 8.2.8.5 Warnings

- If the secondary server is running, and the primary server cannot contact the secondary server when the primary server is restarted, the primary assumes the secondary is not running and takes over. This can result in two servers running at once.
- Each scheduler and data service must be able to run when its server is started, otherwise no jobs will be scheduled; each server can use only its own scheduler and data service.
- Just because servers are redundant, that doesn't mean that your complex is. Look for single points of failure.

## 8.2.9 Troubleshooting Failover

### 8.2.9.1 PBS Does Not Start

- If you see the following error:  

```
"Failover is configured. Temporarily disable failover before running pbs_ds_password"
```

This means that PBS was started for the first time with failover configured. PBS cannot be started for the first time with failover configured. Remove definitions for `PBS_PRIMARY` and `PBS_SECONDARY` from `pbs.conf` on the primary server host, start PBS, stop PBS, replace the definitions, and start PBS again.

### 8.2.9.2 Primary and Secondary Servers Both Running

If both servers are running, this may be because the primary server was stopped and then restarted, and while the primary was stopped, the secondary began to take over. While the secondary server was coming up, it was not able to receive the message from the primary server indicating that it should go idle, or it couldn't register with the primary.

To avoid this problem, use the `-i` option to the `qterm` command, which tells the secondary server to remain idle.

### 8.2.9.3 Primary or Secondary Server Fails to Start

It does not matter in which order the primary and secondary servers are started.

If the primary or secondary server fails to start with the error:

another server running

then check for the following conditions:

1. There may be lock files left in `PBS_HOME/server_priv` that need to be removed.

The primary and secondary servers use different lock files:

- primary: `server.lock`
- secondary: `server.lock.secondary`

2. On Linux, the RPC `lockd` daemon may not be running. You can manually start this daemon by running as root:

`<path to daemon>/rpc.lockd`

Check that all daemons required by your NFS are running.

### 8.2.9.4 Primary Server Periodically Restarting

If the primary server keeps restarting, an unknown secondary server may be contacting it. This can happen when `PBS_PRIMARY` and `PBS_SECONDARY` are missing from `pbs.conf`, but a secondary server has been started.

### 8.2.9.5 Cannot Connect to Host

If you see an error message about not being able to connect to server, check the permissions of [pbs\\_iff](#) on the secondary server. The `setuid` bit may be wrong (permissions should be `-rsxr-xr-x`), or it may be on a shared filesystem that disallows `setuid` programs from running.

## 8.3 Checkpoint and Restart

PBS Professional allows you to configure MoM to checkpoint jobs using your scripts and checkpoint tools. In addition, users may manage their own checkpointing from within their application.

### 8.3.1 Glossary

#### Application Checkpoint

The application performs its own checkpointing when it receives the appropriate signal etc.

#### Checkpoint and Abort, `checkpoint_abort`

The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job uses the restart file when it resumes execution.

#### Restart

A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

#### Restart File

The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

#### Restart Script

The script that MoM runs to restart a job. This script is common to all jobs, and so must use the information in a job's restart file to restart the job.

### Snapshot Checkpoint

The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes based on this restart file if the system experiences a problem during the job's subsequent execution.

## 8.3.2 How Checkpointing Works

When a job is checkpointed, MoM executes a checkpoint script. The checkpoint script saves all of the information necessary to checkpoint the job. If the checkpoint is for a snapshot, the job continues to run. If the job is checkpointed and aborted, PBS kills and requeues the job after checkpointing it.

When a job is restarted, MoM executes a restart script. The restart script uses the saved information to restore the job. The restart script also reads the `$PBS_NODEFILE`. The manner of restarting the job depends on how it was checkpointed:

- If the job was checkpointed during shutdown, the job becomes eligible to run when PBS is restarted, and will start from where it was checkpointed.
- If the job was checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.
- If the job was checkpointed and held via the `qhold` command, the hold must be released via the `qr1s` command for the job to be eligible to run. Then when the scheduler next runs the job, the restart script is executed, and the job runs from where it was checkpointed.

You can configure PBS to requeue jobs that were snapshot checkpointed while they ran, if the epilogue exits with a special value. These jobs are then restarted from the restart file. However, if you are running the `cgroups` hook, any epilogue script will not run. The `cgroups` hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the `cgroups` hook, make your epilogue script into an `execjob_epilogue` hook instead.

You can provide checkpointing for jobs using any combination of scripts that you write and third-party checkpointing tools such as Meiosys Checkpoint and BLCR (Berkeley Lab Checkpoint/Restart). You can configure PBS to trigger the scripts or tools, so that the scripts and/or tools create a job's restart file.

You can configure one behavior for snapshots, and another behavior for checkpoint and abort.

Some applications provide their own checkpointing, which is triggered, for example, when the application receives a signal or detects a change in a file.

### 8.3.2.1 Types of Checkpointing

#### 8.3.2.1.i Checkpoint and Abort

Checkpoint and abort is used when a job is checkpointed before being killed. When the job is checkpointed, the following takes place:

- MoM runs the `checkpoint_abort` script; the checkpoint script or tool writes a restart file specific to that job
- The `checkpoint_abort` script terminates the job
- PBS requeues the job
- If the job was held via the `qhold` command, PBS applies a hold to the job (puts it in the *Held* state)

The job resumes execution based on the information in the restart file.

Checkpoint and abort is applied when:

- The `qhold` command is used on a job
- The server is shut down via `qterm -t immediate` or `qterm -t delay`
- The scheduler preempts a job using the `checkpoint` method

### 8.3.2.1.ii Snapshot Checkpoint

Snapshot checkpointing is used for checkpointing a job at regular intervals. The job continues to run. When the job is checkpointed, the following takes place:

- MoM runs the snapshot checkpoint script; the checkpoint script or tool writes a restart file specific to that job
- The job continues to execute

The job resumes execution based on this restart file if the system crashes or if the epilogue returns -2. See [section 8.3.7.3, “Requeueing via Epilogue”, on page 398](#).

The interval can be specified by the user via `qsub -c <checkpoint spec>`. You can specify a default interval, in the `checkpoint_min` queue attribute, or in the Checkpoint job attribute. See [“qsub” on page 216 of the PBS Professional Reference Guide](#) and [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).

### 8.3.2.1.iii Application Checkpoint

Application checkpointing is when an application checkpoints itself. PBS can be used to trigger application checkpointing, but does not manage the checkpoint files or process. Application checkpointing can be triggered when the application receives a signal or detects a change in a file.

## 8.3.2.2 Events That Trigger Checkpointing

The following table lists the events that can trigger checkpointing, and the kind of checkpointing that is used.

**Table 8-5: Events Triggering Checkpointing**

| Event                                                                                                                                                | Type of Checkpointing Used          | Description                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------------------------------------------------------------------------------|
| The <code>qhold</code> command is used on a job                                                                                                      | <code>checkpoint_abort</code>       | See <a href="#">section 8.3.7.6, “Holding a Job”, on page 399</a>                   |
| Server shut down via <code>qterm -t immediate</code> or <code>qterm -t delay</code>                                                                  | <code>checkpoint_abort</code>       | See <a href="#">section 8.3.7.2, “Checkpointing During Shutdown”, on page 398</a>   |
| Scheduler preempts a job using the checkpoint method                                                                                                 | <code>checkpoint_abort</code>       | See <a href="#">section 8.3.7.5, “Preemption Using Checkpoint”, on page 399</a>     |
| Periodic checkpointing of a job, as specified by <code>qsub -c &lt;checkpoint spec&gt;</code> , or the queue's <code>checkpoint_min</code> attribute | Snapshot                            | See <a href="#">section 8.3.7.1, “Periodic Job Checkpointing”, on page 398</a>      |
| Periodic checkpoint of an application, where checkpoint script triggers application checkpoint                                                       | Snapshot and application checkpoint | See <a href="#">section 8.3.7.7, “Periodic Application Checkpoint”, on page 400</a> |
| User sends application checkpoint signal, or user creates checkpoint trigger file                                                                    | Application checkpoint              | See <a href="#">section 8.3.7.8, “Manual Application Checkpoint”, on page 400</a>   |

### 8.3.2.3 Effect of Checkpointing on Jobs

When a job is checkpointed and aborted (requeued), its accumulated queue waiting time depends on how that time is calculated:

- If you are using eligible time, the accumulated waiting time is preserved
- If you are not using eligible time, the accumulated waiting time is lost

The job exit code for being checkpointed and aborted is -12, named `JOB_EXEC_CHKP`.



When a job is restarted, it runs on the same machine as it did when it was checkpointed.

### 8.3.2.4 Effect of Checkpointing on Job Resources

When a job is checkpointed and aborted, all of its resources are freed.

A snapshot checkpoint does not affect a job's resources.

### 8.3.2.5 Restarting a Job

When a job is restarted, MoM runs the restart script specified in the `$action restart MoM` parameter. This script looks in the checkpoint directory (see [section 8.3.6.5, “Specifying Checkpoint Path”, on page 397](#)) for the restart file for that job. It uses the information in that file to restart the job.

For a job that was checkpointed and aborted because it was held, the job has had a hold placed on it so that it will not be eligible for execution until the hold is released. In order for a checkpointed and held job to be eligible for execution, the hold must be removed using the `qrls` command. The job's owner can remove a User hold, but other holds must be removed by a Manager or Operator. See [“qrls” on page 183 of the PBS Professional Reference Guide](#).

If the job was preempted via checkpointing, the scheduler releases the hold on the job immediately after checkpointing the job. This will show up in the scheduler's log file, but the job will not appear to be held because the hold duration is very short.

A job that was checkpointed and requeued during shutdown is not held. This job is eligible for execution as soon as the necessary daemons are back up. See [section 8.3.7.4, “Checkpointed Jobs and Server Restart”, on page 399](#).

A job that was snapshot checkpointed and later requeued because the epilogue returned a special exit status is requeued in the `Q` state, and is eligible to be restarted when the scheduler selects it for execution.

When a checkpointed and aborted job is restarted, MoM resumes tracking the job. She tracks either the original PID of the job, or the PID of the restart script, depending on the setting of the `$restart_transmogrify MoM` parameter. See [section 8.3.4.3, “Setting \\$restart transmogrify MoM Parameter”, on page 393](#).

## 8.3.3 Prerequisites for Checkpointing Jobs

The following are the prerequisites for checkpointing jobs:

- The MoM must be configured for checkpointing
  - Specified checkpoint directories must correspond to available directories (see [section 8.3.6.5, “Specifying Checkpoint Path”, on page 397](#))
  - Checkpoint and restart MoM configuration parameters must be specified (see [section 8.3.4.2, “Specifying Checkpoint and Restart Parameters”, on page 391](#))
- A checkpointing script or tool must be available for each type of checkpointing to be used

### 8.3.3.1 Restrictions on Checkpointing

- Checkpointing is not supported for job arrays.
- PBS does not directly support OS-level checkpointing.
- You can configure only one snapshot script, so if more than one kind of snapshot checkpointing is required, the script must distinguish which kind of snapshot to perform.
- You can configure only one checkpoint\_abort script, so if more than one kind of checkpoint\_abort is required, the script must also distinguish which kind of checkpoint\_abort to perform.
- You can configure only one restart script. The restart script is run once for each of the job's tasks, so if some restarts are for application checkpointing, the script must handle those restarts correctly (application restarts may require only one iteration.)
- A restarted job must run on the same machine where it was running when it was checkpointed.
- Checkpointing cannot be used for interactive jobs. See [section 8.3.8.2, “Sockets and Checkpointing”, on page 400](#).

## 8.3.4 Configuring Checkpointing

### 8.3.4.1 Overview of Configuring Checkpointing

You configure checkpointing by editing the MoM configuration file, `PBS_HOME/mom_priv/config`. You edit MoM configuration parameters to do the following:

- Specify script paths
  - Specify path to checkpoint\_abort script, if needed
  - Specify path to snapshot script, if needed
  - Specify path to restart script
- Set `$restart_transmogrify` MoM parameter to fit your restart script
- Make the checkpoint path match that specified in the restart script

#### 8.3.4.1.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 8.3.4.2 Specifying Checkpoint and Restart Parameters

To configure checkpointing, you specify a path to a script that MoM executes when checkpointing is called for. You can specify a separate path/script for each of checkpoint\_abort, snapshot, and restart using the following MoM configuration parameters:

`$action checkpoint timeout !path/script script-args`

Specifies snapshot behavior.

`$action checkpoint_abort timeout !path/script script-args`

Specifies checkpoint\_abort behavior.

`$action restart timeout !path/script script-args`

Specifies restart behavior.

where

`$action`

Specifies that MoM perform the indicated action.

**checkpoint**

MoM executes the script specified in `path/script` once for each of the job's tasks when a snapshot is called for.

**checkpoint\_abort**

MoM executes the script specified in `path/script` once for each of the job's tasks when a `checkpoint_abort` is called for.

**restart**

MoM executes the script specified in `path/script` once for each of the job's tasks when a restart is called for.

**timeout**

The number of seconds allowed for the script or tool to execute. The value of the `$restart_transmogrify` MoM parameter determines whether this limit is applied. Values for `$restart_transmogrify`, and resulting behavior:

**False**

If the script/tool does not finish running during this time, it is killed and handled as if it had returned failure.

**True**

No timeout limit is applied.

**path/script**

The path to the script, including the name of the script. The path can be absolute or relative. If the path is relative, it is relative to `PBS_HOME/mom_priv`.

Examples of absolute paths and script names:

`/usr/bin/checkpoint/snapshot`

`/usr/bin/checkpoint/checkpt-abort`

`/usr/bin/checkpoint/restart`

**script-args**

These are the arguments to the script, if any.

PBS automatically expands some arguments to checkpoint and restart scripts. The following table lists the arguments that are expanded by PBS:

**Table 8-6: Checkpoint Script Arguments Expanded by PBS**

| Argument | Description                                     |
|----------|-------------------------------------------------|
| %globid  | Global ID (no longer used)                      |
| %jobid   | Job ID                                          |
| %sid     | Session ID                                      |
| %taskid  | Task ID                                         |
| %path    | File or directory name to contain restart files |

### 8.3.4.2.i Examples of Checkpoint and Restart Parameters

The following are examples of snapshot, `checkpoint_abort`, and restart MoM parameters:

```
$action checkpoint 60 !/usr/bin/checkpoint/snapshot %jobid %sid %taskid %path
```

```
$action checkpoint_abort 60 !/usr/bin/checkpoint/checkpt-abort %jobid %sid %taskid %path
```

```
$action restart 30 !/usr/bin/checkpoint/restart %jobid %sid %taskid %path
```

### 8.3.4.3 Setting \$restart\_transmogrify MoM Parameter

The \$restart\_transmogrify MoM parameter controls how MoM runs the restart script, and whether she expects to resume tracking the job's original PID or a new PID. When she runs a restart script, MoM forks a child process, which `exec()`s the start script. If \$restart\_transmogrify is *True*, the start script becomes the top task of the job. If \$restart\_transmogrify is *False*, the start script does not become the top task of the job.

If your restart script preserves the job's original PID, set \$restart\_transmogrify to *False*. This way, the script does not become the top task of the job, and MoM continues to track the job's original PID.

If your restart script results in a new PID for the job, set \$restart\_transmogrify to *True*. This way, the restart script becomes the top task of the job, and MoM tracks the PID of the new top process, which is the script.

## 8.3.5 Parameters and Attributes Affecting Checkpointing

### 8.3.5.1 MoM Configuration Parameters Affecting Checkpointing

**\$action checkpoint <timeout> !<script-path> <args>**

Checkpoint the job, allowing the job to continue running.

**\$action checkpoint\_abort <timeout> !<script-path> <args>**

Checkpoints, kills, and requeues the job.

**\$action restart <timeout> !<script-path> <args>**

Restarts checkpointed job.

The <timeout> is the time allowed for checkpoint or restart script to run.

**\$checkpoint\_path <path>**

MoM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to `PBS_HOME/mom_priv`. Overrides default. Overridden by path specified in the `pbs_mom -C` option and by `PBS_CHECKPOINT_PATH` environment variable.

**\$restart\_background <True|False>**

Specifies whether MoM runs the restart script in the background (MoM doesn't wait) or foreground (MoM waits). When set to *True*, MoM runs the restart script in the background.

Automatically set by MoM; Controlled by value of \$restart\_transmogrify. When \$restart\_transmogrify is *True*, \$restart\_background is set to *False*. When \$restart\_transmogrify is *False*, \$restart\_background is set to *True*.

Format: *Boolean*

Default: *False*

**\$restart\_transmogrify <True|False>**

Specifies which PID MoM tracks for a job that has been checkpointed and restarted.

When this parameter is set to *True*, MoM tracks the PID of the restart script. When this parameter is set to *False*, MoM tracks the PID of the original job.

The value of \$restart\_transmogrify controls the value of \$restart\_background.

Format: *Boolean*

Default: *False*

### 8.3.5.2 Options to `pbs_mom` Affecting Checkpointing

#### `-C checkpoint_directory`

Specifies the path to the directory where MoM creates job-specific subdirectories used to hold each job's restart files. MoM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods. Any directory specified with the `-C` option must be owned, readable, writable, and executable by root only (*rwX,---,---*, or *0700*), to protect the security of the restart files. See the `-d` option to `pbs_mom`.

Format: *String*

Default: `PBS_HOME/checkpoint`

### 8.3.5.3 Job Attribute Affecting Checkpointing

#### Checkpoint

Determines when the job will be checkpointed. Can take on one of the following values:

**c**

Checkpoint at intervals, measured in CPU time, set on the job's execution queue. If there is no interval set on the queue, the job is not checkpointed.

**c=<minutes of CPU time>**

Checkpoint at intervals of the specified number of minutes of job CPU time. This value must be greater than zero. If the interval specified is less than that set on the job's execution queue, the queue's interval is used.

Format: *Integer*

**w**

Checkpoint at intervals, measured in walltime, set on the job's execution queue. If there is no interval set at the queue, the job is not checkpointed.

**w=<minutes of walltime>**

Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: *Integer*

**n**

No checkpointing.

**s**

Checkpoint only when the server is shut down.

**u**

Unset. Defaults to behavior when interval argument is set to **s**.

Default: *u*.

Format: *String*

### 8.3.5.4 Queue Attribute Affecting Checkpointing

#### `checkpoint_min`

Specifies the minimum number of minutes of CPU time or walltime allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in `checkpoint_min` is used for both CPU minutes and walltime minutes. See the `Checkpoint` job attribute.

Format: *Integer*

Default: None

Python attribute value type: `pbs.duration`

### 8.3.5.5 Environment Variable Affecting Checkpointing

#### PBS\_CHECKPOINT\_PATH

MoM passes this path to the checkpoint and restart scripts. Overridden by -C option to pbs\_mom; overrides \$checkpoint\_path MoM parameter and default. See [section 8.3.6.5, “Specifying Checkpoint Path”, on page 397](#).

#### PBS\_NODEFILE

PBS uses the \$PBS\_NODEFILE to restart the job. Make sure it is available.

### 8.3.5.6 The Epilogue

PBS will requeue a job which was snapshot checkpointed, if the epilogue returns the value 2. See [section 8.3.7.3, “Requeueing via Epilogue”, on page 398](#).

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an `execjob_epilogue` hook instead.

## 8.3.6 Checkpoint and Restart Scripts

The restart script is run by the same MoM that ran the checkpoint script. The checkpoint and restart scripts are run for each task of the job. When MoM executes a checkpoint or restart script, she forks a child process, which `exec()`s the script. The restart script looks for the restart file in the job-specific subdirectory created by MoM, under the specified path. See [section 8.3.6.5, “Specifying Checkpoint Path”, on page 397](#).

### 8.3.6.1 Environment Variables for Scripts

PBS sets the following variables in the checkpoint and restart scripts' environments before running the scripts:

**Table 8-7: Checkpoint/Restart Script Environment Variables**

| Environment Variable | Value of Variable                                                                  |
|----------------------|------------------------------------------------------------------------------------|
| GID                  | Job owner's group ID                                                               |
| HOME                 | Job owner's PBS home directory                                                     |
| LOGNAME              | Job owner's login name                                                             |
| PBS_JOBCOOKIE        | 128-bit random number used as token to authenticate job processes                  |
| PBS_JOBID            | The job's ID                                                                       |
| PBS_JOBNAME          | The job's name                                                                     |
| PBS_MOMPORT          | Port number on which MoM listens for resource manager requests                     |
| PBS_NODEFILE         | Path and filename of this job's node file                                          |
| PBS_NODENUM          | Index into the node file; index of this vnode; starts at 0                         |
| PBS_QUEUE            | Name of the job's execution queue                                                  |
| PBS_SID              | Session ID of task for which script is being called                                |
| PBS_TASKNUM          | Index into task table for this job; index of task for which script is being called |
| SHELL                | Job owner's login shell                                                            |

**Table 8-7: Checkpoint/Restart Script Environment Variables**

| Environment Variable | Value of Variable                                 |
|----------------------|---------------------------------------------------|
| UID                  | Job owner's execution ID                          |
| USER                 | Job owner's username                              |
| USERPROFILE          | (Windows only) Job owner's Windows home directory |
| USERNAME             | (Windows only) Job owner's Windows username       |

### 8.3.6.2 The Checkpoint Script

The checkpoint script writes a restart file that is specific to the job being checkpointed. The checkpoint script must save all of the information needed to restart the job. This is the information that will be used by the restart script to restart the job. PBS runs the script for each running job task, on each vnode where a task is running.

#### 8.3.6.2.i Requirements for Checkpoint Script

- The first line of the script must specify the shell to be used, for example:  
#!/bin/sh
- The script should return the following error codes:
  - *Zero* for success
  - *Non-zero* for failure
- The script should block until the checkpoint process is finished.
- The restart file and its directory should be owned by root, and writable by root only, with permission 0755.
- Under Linux, the checkpoint script should be owned by root, and writable by root only, with permission 0755.
- Under Windows, the checkpoint script must have at least Full Control permission for the local Administrators group.
- The checkpoint script must write the restart file(s) in the location expected by the restart script. You don't have to use the %path parameter passed by MoM.
- If the script is for checkpoint-abort, the script must ensure that all processes are killed, whether directly or indirectly, for example by touching a file. All job processes must exit.

### 8.3.6.3 The Restart Script

The restart script does only one of the following:

- Reinstates the job's original PID, so that MoM tracks the original PID
- Becomes the new top process of the job, so that MoM tracks the PID of the script

If `$restart_transmogrify` is set to *True*, the restart script becomes the new top task for the job, and MoM begins tracking its process ID, where she was tracking the job's original process ID. If `$restart_transmogrify` is set to *False*, MoM continues to track the original job PID.

The restart script can use `pbs_attach ( )` to attach job processes to the original job PID, or to the script's PID. See [“pbs\\_attach” on page 56 of the PBS Professional Reference Guide](#).

#### 8.3.6.3.i Caveats for Restart Script

The `pbs_attach ( )` command is not supported under Windows.

#### 8.3.6.3.ii Requirements for Restart Script

The restart script must handle everything required to restart the job from the information saved by the checkpoint script.



The restart script must block until the restart process is finished.

Under Linux, the restart script should be owned by root, and writable by root only, with permission 0755.

Under Windows, the restart script must have at least Full Control permission for the local Administrators group.

### 8.3.6.3.iii Return Values for Restart Script

The restart script must inform PBS of success or failure. It must return one of the following:

- *Zero* for success
- *Non-zero* for failure

## 8.3.6.4 Scripts for Application Checkpointing

If a user's application can be checkpointed periodically according to walltime or CPU time, you can use the PBS snapshot checkpoint facility to trigger snapshot checkpointing by the application.

If a user's application can be checkpointed, you can use the PBS `checkpoint_abort` facility before shutting down PBS to avoid losing intermediate results.

Some applications produce a restart file when they are sent a specific signal, or when a specific file is affected. A checkpoint script for this purpose sends the application the correct signal, or makes the correct change to the file.

Some applications only need the checkpoint and restart scripts to be run once each. In this case, the checkpoint and restart scripts should handle this requirement.

## 8.3.6.5 Specifying Checkpoint Path

When a job is checkpointed, information about the job is saved into a file. The location for this file can be any directory accessible to MoM.

The path to the checkpoint directory is composed of two parts. The first part is common to all jobs; this part can be specified. The second part is a job-specific subdirectory, created by MoM for each job, under the common directory. The job's restart file is written in this job-specific subdirectory.

The default common directory, `PBS_HOME/checkpoint`, is provided for convenience.

You can specify the filename and the path for the common directory using any of the following methods. If the first is specified, PBS uses it. If not, and the second is specified, PBS uses the second, and so on.

- The `-C` path option to the `pbs_mom` command
- The `PBS_CHECKPOINT_PATH` environment variable
- The `$checkpoint_path` MoM configuration option in `PBS_HOME/mom_priv/config`
- The default value of `PBS_HOME/checkpoint`

The job-specific subdirectory is named with the following format:

`<job ID>.CK`

For example, if you specify `/usr/bin/checkpoint` for the common directory, and the job's ID is `1234.host1`, the job's restart file is written under `/usr/bin/checkpoint/1234.host1.CK`.

The restart file and its directory should be owned by root, and writable by root only.

### 8.3.6.5.i Checkpoint Path Caveats

If the checkpoint file is in `PBS_HOME/checkpoint/<job ID>.CK`, and MoM thinks that a checkpoint failed (the checkpoint script returned non-zero), she will remove the checkpoint file. If the checkpoint script puts the checkpoint file in another location, MoM does not remove the checkpoint file.

## 8.3.7 Using Checkpointing

### 8.3.7.1 Periodic Job Checkpointing

If a job's Checkpoint attribute is set to *c*, *c=<minutes>*, *w*, or *w=<minutes>*, the job is periodically checkpointed. The checkpoint interval is specified either in the job's Checkpoint attribute or in the queue's `checkpoint_min` attribute. See [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#). The job's Checkpoint attribute is set using the `-c <interval>` option to the `qsub` command. See [“qsub” on page 216 of the PBS Professional Reference Guide](#).

When this attribute is set, at every `<interval>` the job is checkpointed and a restart file is written, but the job keeps running.

### 8.3.7.2 Checkpointing During Shutdown

The effect on jobs of shutting down PBS depends on the method used to shut PBS down. When a job is checkpointed during shutdown, MoM runs the `checkpoint_abort` script, and PBS kills and requeues the job. PBS does not hold the job, so the job is eligible to be run again as soon as the server starts up.

If you use the `qterm` command, there are three different suboptions to the `-t` option to control whether jobs are checkpointed, requeued, or allowed to continue running.

If you use the PBS start/stop script, the script affects only the host where the script is run. Any jobs running completely or partly on that host are killed and requeued, but not checkpointed. Any jobs not running on that host are left running.

The effect of each shutdown method is described here:

**Table 8-8: Effect of Shutdown on Jobs**

| Shutdown Method                                                       | Effect on Checkpointable Jobs                                                                                                                                        | Effect on Non-checkpointable Jobs                                                                                                                                    |
|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>qterm -t quick</code>                                           | Continue to run                                                                                                                                                      | Continue to run                                                                                                                                                      |
| <code>qterm -t delay</code>                                           | Checkpointed, killed, requeued, held                                                                                                                                 | Requeued if rerunnable; continue to run if not rerunnable                                                                                                            |
| <code>qterm -t immediate</code>                                       | Checkpointed, killed, requeued, held                                                                                                                                 | Requeued if rerunnable; deleted if not rerunnable                                                                                                                    |
| <code>systemctl stop pbs</code><br>or<br><code>init.d/pbs stop</code> | Any jobs running completely or partly on host where stop script is run are killed and requeued<br>Jobs not running on host where stop script is run are left running | Any jobs running completely or partly on host where stop script is run are killed and requeued<br>Jobs not running on host where stop script is run are left running |

Any running subjobs of a job array keep running when the server is shut down.

### 8.3.7.3 Requeueing via Epilogue

You can configure MoM to requeue a failed job that was snapshot checkpointed during its execution. For example, if a job terminates, but had a hardware failure during execution, PBS can requeue the job, and MoM will run the start script, which can restart the job from its restart file.

When the job is requeued via the epilogue mechanism, it is in the `Q` state.

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an `execjob_epilogue` hook instead.

### 8.3.7.3.i Requirements for Requeueing via Epilogue

The following requirements must be met in order for a job to be requeued via the epilogue mechanism:

- The epilogue must return a value of 2
- The job must have been checkpointed under the control of PBS
- The MoM must be configured with a restart script in the `$action restart MoM` configuration parameter
- The MoM must be configured to snapshot checkpoint jobs in the `$action checkpoint MoM` configuration parameter
- The jobs must request checkpointing via their `Checkpoint` attribute. See [section 8.3.7.1, “Periodic Job Checkpointing”, on page 398](#)
- The epilogue script in `PBS_HOME/mom_priv/epilogue` must return the following:
  - *Zero (0)* for successful termination (requeue is not required)
  - *Two (2)* for failure (requeue is required)

### 8.3.7.4 Checkpointed Jobs and Server Restart

When the server is restarted using the `pbs_server -t warm` command, `systemd`, or the `init.d/pbs start` script, jobs that were checkpointed and aborted upon shutdown are waiting in their queues, and are eligible to be run according to the scheduler's algorithm.

When the server is restarted using the `pbs_server -t hot` command, jobs that were checkpointed and aborted upon shutdown are immediately rerun, before the scheduler selects which jobs to run.

### 8.3.7.5 Preemption Using Checkpoint

When a job is preempted via checkpointing, MoM runs the `checkpoint_abort` script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed. For a description of using preemption, see [section 4.9.33, “Using Preemption”, on page 180](#).

### 8.3.7.6 Holding a Job

When anyone uses the `qhold` command to hold a checkpointable job, MoM runs the `checkpoint_abort` script, which kills all job processes, and PBS requeues, and holds the job.

A job with a hold on it must have the hold released via the `qrls` command in order to be eligible to run.

The following is the sequence of events when a job is held:

- MoM runs the `checkpoint_abort` script
- The job's execution is halted
- The resources assigned to the job are released
- The job is placed in the *Held* state in the execution queue
- The job's `Hold_Types` attribute is set appropriately

---

A held job is waiting in its queue. The following is the sequence of events when a held job is restarted:

- The hold is released by means of the `qr1s` command; the job is now in the *Queued* state
- The job continues to wait in its queue until the scheduler schedules it for execution
- The scheduler selects the job for execution
- The job is sent to its original MoM for execution
- The MoM runs the restart script

### 8.3.7.6.i Restrictions on Holding a Job

A job in the process of provisioning cannot be held.

The `qhold` command can be used on jobs and job arrays, but not on subjobs or ranges of subjobs.

If the job cannot be checkpointed and aborted, `qhold` simply sets the job's `Hold_Types` attribute. The job continues to execute.

The checkpoint-abort script must terminate all job processes, or the `qhold` command will appear to hang.

### 8.3.7.7 Periodic Application Checkpoint

The snapshot checkpoint script can trigger checkpoint by a job's application, if the application is written to support checkpointing itself. Note that an application may be designed to be checkpointed at specific stages in its execution, rather than at specific points in time. If an application can be usefully checkpointed at specific points in time, then snapshot checkpointing may be useful. See [section 8.3.7.1, “Periodic Job Checkpointing”, on page 398](#).

### 8.3.7.8 Manual Application Checkpoint

When an application is checkpointed manually, the user triggers checkpointing by the application by sending the application a specific signal, or by creating a file.

## 8.3.8 Advice and Caveats

### 8.3.8.1 PBS\_NODEFILE Required

Make sure that the `$PBS_NODEFILE` is available during restart.

### 8.3.8.2 Sockets and Checkpointing

Multi-vnode jobs may cause network sockets to be opened between submission and execution hosts, and open sockets may cause a checkpointing script or tool to fail. The following use sockets:

- An interactive job, i.e. a job submitted using `qsub -I`, opens unprivileged sockets. `qsub` binds a socket to a port, then waits to accept a connection from MoM on that socket. Data from standard in is written to the socket and data from the socket is written to standard out.
- The `pbs_demux` process collects `stdio` streams from all tasks
- The `pbsdsh` program spawns tasks. The `-o` option to this command prevents it from waiting for spawned tasks to finish, so that no socket is left open to the MoM to receive task manager events. When the `-o` option is used, the shell must use some other method to wait for the tasks to finish. See [“pbsdsh” on page 30 of the PBS Professional Reference Guide](#).

### 8.3.9 Accounting

If a job is checkpointed and requeued, the exit status passed to the epilogue and recorded in the accounting record is the following:

-12, meaning that the job was checkpointed and aborted

A checkpoint ("C") record is written in the accounting log when the job is checkpointed and requeued, as when the `qhold` command is used, or the job is checkpointed and aborted.

## 8.4 Reservation Fault Tolerance

If the vnodes associated with an advance reservation, the soonest occurrence of a standing reservation, or a job-specific reservation become unavailable, PBS marks the reservation as *degraded* (state 10). If the vnodes are instead taken over by a maintenance reservation, PBS marks the reservation as *in conflict* (state 12).

PBS attempts to reconfirm degraded or in-conflict reservations by finding replacements for vnodes that have become unavailable.

When a reservation is degraded, PBS may still be able to use the unavailable original vnodes, if they become available in time. When a reservation is in conflict, the vnodes that were taken over by the maintenance reservation are removed from the reservation; they are no longer in the reservation's `resv_nodes` attribute, and PBS looks for other vnodes.

States of available vnodes:

*free*

*busy*

*job-exclusive*

*job-sharing*

*job-busy*

States of unavailable vnodes:

*down*

*maintenance*

*offline*

*provisioning*

*stale*

*state-unknown, down*

*unresolvable*

*wait-provisioning*

### 8.4.1 States for Degraded and In-conflict Reservations

A degraded reservation's state becomes `RESV_DEGRADED`, abbreviated DG, and its substate becomes `RESV_DEGRADED`.

If vnodes associated with an occurrence later than the soonest occurrence of a standing reservation become unavailable, the reservation stays in state `RESV_CONFIRMED`, but its substate becomes `RESV_DEGRADED`.

During the time that a degraded advance or job-specific reservation, or the soonest occurrence of a degraded standing reservation is running, its state is `RESV_RUNNING`, and its substate is `RESV_DEGRADED`.

---

An in-conflict reservation's state becomes *RESV\_IN\_CONFLICT*, abbreviated IC, and its substate becomes *RESV\_IN\_CONFLICT*.

For a table of degraded and in-conflict reservation states and substates, see [“Degraded Reservation Substates” on page 368 of the PBS Professional Reference Guide](#). For a table of numeric values for reservation states and substates, see [“Reservation States” on page 367 of the PBS Professional Reference Guide](#).

## 8.4.2 Finding Replacement Vnodes for Degraded and In-conflict Reservations

PBS attempts to reconfirm reservations by finding replacements for vnodes that have become unavailable. If a reservation is not running, PBS will use any available vnodes. If it is running, any vnode without a running job on it may change.

PBS attempts to reconfirm a reservation only during periods when this makes sense:

- If a reservation is not actively running, PBS waits the time specified in `reserve_retry_time`, then starts periodically trying to reconfirm the reservation, including making an attempt to reconfirm the reservation just before the start time of each occurrence.
- If an in-conflict reservation is actively running, PBS does not attempt to reconfirm it.
- If a degraded reservation is actively running, and the reservation is not in conflict, and the reservation has no running jobs on the unavailable vnodes, PBS periodically attempts to reconfirm it every `reserve_retry_time` seconds.
- If an actively running reservation is degraded because a vnode becomes unavailable, and the reservation has running jobs:
  - If the unavailable vnode has any jobs running on it, PBS waits until those jobs are finished to periodically attempt to reconfirm the reservation.
  - If the unavailable vnode has no jobs running on it, PBS does not wait until the jobs are finished to periodically attempt to reconfirm the reservation. PBS periodically attempts to reconfirm it every `reserve_retry_time` seconds.

A degraded or in-conflict reservation has a read-only reservation attribute called `reserve_retry`, whose value is the next time at which the reservation is due to be reconfirmed.

### 8.4.2.1 Attributes Affecting Reservation Reconfirmation

#### `reserve_retry_time`

Server attribute. The time period between attempts to reconfirm the reservation.

Settable by Manager; readable by all

Format: Integer (seconds)

Values: Must be greater than *zero*

Default: *600* (10 minutes)

Python attribute value type: `int`

---

### 8.4.3 Allocating New Vnodes

Once new vnodes are allocated for a reservation:

- The reservation has been confirmed
- If the reservation is not running, the state and substate of the reservation are *RESV\_CONFIRMED*
- If the reservation is running, the state of the reservation is *RESV\_RUNNING* and the substate is *RESV\_CONFIRMED*
- The reservation's `resv_nodes` attribute lists the new vnodes

### 8.4.4 Restarting the Server

When the server is restarted, reservations are assumed confirmed until associated vnodes are recognized as unavailable. If any reservations become degraded or in conflict after a server restart, PBS sets the time when the reservation becomes degraded to the time of the restart. If a vnode is set *offline* before the restart, it is considered unavailable after the restart, so all its associated reservations become degraded.

## 8.5 Vnode Fault Tolerance for Job Start and Run

PBS lets you allocate extra vnodes to a job so that the job can successfully start and run even if some vnodes fail. You can allocate the extra vnodes only for startup, or for the life of the job. Later, for jobs where the extra vnodes are needed only for reliable startup, you can trim the allocated vnodes back to just what the job will use to run, releasing the unneeded vnodes for other jobs.

You allocate extra vnodes in a `queuejob` hook using the `pbs.select.increment_chunks()` method, and you release vnodes in an `execjob_launch` or `execjob_prologue` hook using the `pbs.event().job.release_nodes()` method.

We provide an example hook in `$PBS_EXEC/unsupported/ReliableJobStartup.py`.

### 8.5.1 Overview of Padding and Trimming Vnode Requests

Here is an overview of the steps for improving job startup and run reliability. We describe each of them in detail in the next subsections, and we give an example at the end of this section.

- Use a `queuejob` hook to do the following:
  - Save the job's initial vnode request
  - Set the job's `tolerate_node_failures` attribute to the desired value
  - Pad the job's vnode request
- Configure primary MoMs to wait for sister MoMs to acknowledge joining job
- Configure primary MoMs to wait for hooks to complete
- Use an `execjob_launch` or `execjob_prologue` hook to trim the vnodes not used by the job from the job's vnode request



## 8.5.2 Saving Job Initial Vnode Request

To save the job's initial resource request so that you know how much to trim later, use a `queuejob` hook to save it into a built-in resource such as the `site` resource (currently, it cannot be a custom resource). Here is a code snippet:

```
import pbs
e=pbs.event()
j = e.job
e.job.Resource_List["site"] = str(e.job.Resource_List["select"])
```

## 8.5.3 Configuring Primary MoMs to Wait for Sister MoMs

When the primary MoM gets a job whose `tolerate_node_failures` attribute is set to *all* or *job\_start*, the primary MoM can wait to start the job for up to a configured number of seconds if the sister MoMs do not immediately acknowledge joining the job. This gives the sister MoMs more time to join the job. You configure the number of seconds for the primary MoM to wait for sister MoMs via the `sister_join_job_alarm` configuration parameter in MoM's config file:

```
$sister_join_job_alarm <number of seconds to wait>
```

The default value for this parameter is the sum of the values of the alarm attributes of any enabled `execjob_begin` hooks. If there are no enabled `execjob_begin` hooks, the default value is 30 seconds. For example, if there are two enabled `execjob_begin` hooks, one with `alarm = 30` and one with `alarm = 20`, the default value of MoM's `sister_join_job_alarm` is 50 seconds.

After all the sister MoMs have joined the job, or MoM has waited for the value of the `sister_join_job_alarm` parameter, she starts the job.

## 8.5.4 Configuring MoMs to Wait for Hooks

When the primary MoM gets a job whose `tolerate_node_failures` attribute is set to *all* or *job\_start*, the primary MoM can wait to start the job (running the job script or executable) for up to a configured number of seconds. During this time, `execjob_prologue` hooks can finish and the primary MoM can check for communication problems with sister MoMs. You configure the number of seconds for the primary MoM to wait for hooks via the `job_launch_delay` configuration parameter in MoM's config file:

```
$job_launch_delay <number of seconds to wait>
```

The default value for this parameter is the sum of the values of the alarm attributes of any enabled `execjob_prologue` hooks. If there are no enabled `execjob_prologue` hooks, the default value is 30 seconds. For example, if there are two enabled `execjob_prologue` hooks, one with `alarm = 30` and one with `alarm = 60`, the default value of MoM's `job_launch_delay` is 90 seconds.

After all the `execjob_prologue` hooks have finished, or MoM has waited for the value of the `job_launch_delay` parameter, she starts the job.

### 8.5.4.1 Caveats for Configuring MoMs to Wait for Hooks

This configuration option is not supported under Windows.

## 8.5.5 Padding Vnode Request

To add extra vnodes to a job's vnode request, specify for the job whether you want more vnodes for startup, for the life of the job, or not at all, and specify how you want to pad the job's vnode request.

### 8.5.5.1 Specifying Whether and When to Pad Vnode Request

To specify whether and when the job gets extra vnodes, set the job's `tolerate_node_failures` attribute to one of *none*, *job\_start*, or *all*.

**Table 8-9: Behavior for `tolerate_node_failures`**

| Value of <code>tolerate_node_failures</code> | Behavior                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>none or unset</i>                         | No extra vnodes are allocated to the job. Default behavior.                                                                                                                                                                                                                                                                                                                   |
| <i>job_start</i>                             | Extra vnodes are allocated only long enough to start the job successfully.<br>Tolerate vnode failures that occur only during job start, just before executing the job's top level shell or executable or any <code>execjob_launch</code> hooks.<br>Failures tolerated are those such as an assigned sister MoM failing to join the job and communication errors between MoMs. |
| <i>all</i>                                   | Extra vnodes are allocated for the life of the job.<br>Tolerate all node failures resulting from communication problems, such as polling problems, between the primary MoM and the sister MoMs assigned to the job<br>Tolerate failures due to rejections from <code>execjob_begin</code> or <code>execjob_prologue</code> hooks run at sister MoMs.                          |

#### 8.5.5.1.i Setting the `tolerate_node_failures` Job Attribute

You or the job submitter can set the job's `tolerate_node_failures` attribute via `qsub`, `qalter`, or in a Python hook, for example a `queuejob` hook. If set via `qalter` while the job is already running, the attribute is consulted the next time the job is rerun.

You can set a value for `tolerate_node_failures` for all jobs via the server's `default_qsub_arguments` attribute.

Examples of setting this attribute:

- Via `qsub`:  
`qsub -W tolerate_node_failures="all" <job script>`
- Via `qalter`:  
`qalter -W tolerate_node_failures="job_start" <job ID>`
- Via a hook. The following code snippet shows how to set this attribute:  

```
cat qjob.py
import pbs
e=pbs.event()
e.job.tolerate_node_failures = "all"
```

### 8.5.5.2 Specifying How Chunks Are Padded

To specify how you want each chunk padded, use the `pbs.select.increment_chunks(<increment specification>)` method. This method increments the job's chunks according to the rules you give in the *increment specification*. See ["Method to Increment select Object Chunks" on page 175 in the PBS Professional Hooks Guide](#).

### 8.5.5.2.i Example of Padding Chunks

The following code snippet illustrates padding a job's vnode request by one extra vnode per chunk:

```
import pbs
e=pbs.event()
j = e.job
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

### 8.5.5.3 Caveats for Padding Vnode Requests

The `tolerate_node_failures` job attribute is not supported on Cray systems. It is ignored on Cray systems.

## 8.5.6 Trimming Vnode Request

When you trim a job's vnode request, you can trim the larger padded amount back to the job's initial vnode request. To trim a job's vnode request, use the `pbs.event().job.release_nodes(keep_select)` method. This method automatically selects vnodes that satisfy the new request and are healthy, keeps them in the job's vnode request, and releases all others. The method automatically trims out any vnodes in the `pbs.event().vnode_list_fail[]` list.

You can call `pbs.event().job.release_nodes(keep_select = <desired vnodes>)` in an `execjob_launch` or `execjob_prologue` hook. Note that despite the method being named "release\_nodes", it **keeps** the specified vnodes and **releases** all other vnodes. You can specify the job's original vnode request as the vnodes to keep.

The `pbs.event().job.release_nodes()` method returns a PBS job object which has the updated values for the job's `exec_vnode` and `Resource_List` attributes.

See ["Job Object Method to Release Vnodes" on page 141 in the PBS Professional Hooks Guide](#).

### 8.5.6.1 Example of Trimming Job Vnode Request

Here we use an `execjob_prologue` hook to trim a job's vnode request:

```
pj = e.job.release_nodes(keep_select="ncpus=2:mem=2gb+ncpus=2:mem=2gb+ncpus=1:mem=1gb")
if pj != None:
 pbs.logmsg(pbs.LOG_DEBUG, "pj.exec_vnode=%s" % (pj.exec_vnode,))
else:
 # returned None job object, so we can put a hold on the job and requeue it,
 # rejecting the hook event
 e.job.Hold_Types = pbs.hold_types("s")
 e.job.rerun()
 e.reject("unsuccessful at LAUNCH")
```

### 8.5.6.2 Offlining Vnodes that Have Gone Bad During Start or Run

See ["Using List of Failed Vnodes to Offline Vnodes that Have Gone Bad During Start or Run" on page 72 in the PBS Professional Hooks Guide](#).

## 8.5.7 Checking Vnodes and Marking Them as Failed

For each `execjob_prologue` and `execjob_launch` event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. PBS records this list in the `pbs.event().vnode_list_fail[]` object. See ["The Failed Vnode List Event Member" on page 125 in the PBS Professional Hooks Guide](#).

Any sister vnodes that are able to join the job are considered healthy.

The successful outcome of a join job request may be the result of a check made by a remote `execjob_begin` hook. After successfully joining the job, the vnode may further check its status via a remote `execjob_prologue` hook. A rejection by the remote `execjob_prologue` hook causes the primary MoM to treat the sister vnode as a problem vnode, and the sister vnode is marked as unhealthy.

If there's an `execjob_prologue` hook in place, the primary MoM tracks vnode hosts that have acknowledged their execution of the `execjob_prologue` hook. Then after some `job_launch_delay` amount of time for job startup, the primary MoM starts reporting as failed vnodes those which have not given their positive acknowledgement during `execjob_prologue` hook execution.

If after some time, a vnode's host comes back with an acknowledgement of successful `execjob_prologue` hook execution, the primary MoM adds that host back to the healthy list.

You may want to offline any bad vnodes; see ["Offlining Bad Vnodes" on page 72 in the PBS Professional Hooks Guide](#).

## 8.5.8 Example of Reliable Job Startup and Run

In order to have a job start reliably, we need these:

- A `queuejob` hook that does the following:
  - Makes the job tolerate vnode failures by setting the `tolerate_node_failures` job attribute to `job_start`
  - Adds extra chunks to the job's select specification using the `pbs.event().job.select.increment_chunks()` method
  - Saves the job's original vnode request into a built-in string resource (for example, "site")
- An `execjob_launch` hook that calls `pbs.event().job.release_nodes()` to trim the job's vnode request back to the original.

### 8.5.8.1 Example Queuejob Hook for Setup and Padding

We will use a `queuejob` hook called `qjob.py` in our example. In the `queuejob` hook:

- Make the job tolerant of failures:
 

```
import pbs
e=pbs.event()
j = e.job
j.tolerate_node_failures = job_start
```
- Save the job's initial vnode request in the built-in resource named `site`:
 

```
e.job.Resource_List["site"] = str(e.job.Resource_List["select"])
```
- Add extra chunks to the vnode request:
 

```
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

Instantiate the `queuejob` hook:

```
qmgr -c "c h qjob event=queuejob"
qmgr -c "i h qjob application/x-python default qjob.py"
```

### 8.5.8.2 Example Hook for Trimming

We will use an `execjob_launch` hook named *launch.py* to trim the job's padded vnode request back to the original vnode request. This hook runs before the job executes.

```
import pbs
e=pbs.event()
if 'PBS_NODEFILE' not in e.env:
 e.accept()
j = e.job
pj = j.release_nodes(keep_select=e.job.Resource_List["site"])
if pj is None: # not successful pruning the vnodes
 j.rerun() # rerun (requeue) the job
 e.reject("something went wrong pruning the job back to its original select request")
```

Instantiate the `execjob_launch` hook:

```
qmgr -c "c h launch event=execjob_launch"
qmgr -c "i h launch application/x-python default launch.py"
```

### 8.5.8.3 Example Job

Here is our example job:

```
% cat jobr.scr
#PBS -l select="ncpus=3:mem=1gb+ncpus=2:mem=2gb+ncpus=1:mem=3gb"
#PBS -l place=scatter:excl

echo $PBS_NODEFILE
cat $PBS_NODEFILE
echo END
echo "HOSTNAME tests"
echo "pbsdsh -n 0 hostname"
pbsdsh -n 0 hostname
echo "pbsdsh -n 1 hostname"
pbsdsh -n 1 hostname
echo "pbsdsh -n 2 hostname"
pbsdsh -n 2 hostname
echo "PBS_NODEFILE tests"
for host in `cat $PBS_NODEFILE`
do
 echo "HOST=$host"
 echo "pbs_tmrsh $host hostname"
 pbs_tmrsh $host hostname
 echo "ssh $host pbs_attach -j $PBS_JOBID hostname"
 ssh $host pbs_attach -j $PBS_JOBID hostname
done
```

### 8.5.8.4 Example of Job Vnode Assignment Padding and Trimming

When our job first starts, it is assigned 5 vnodes, because its select specification was modified by adding 2 vnodes:

```
% qstat -f 20
Job Id: 20.mars.example.com
...
exec_host = mars/0*3+jupiter/0*2+saturn/0*2+mercury/0+neptune/0
exec_vnode =
 (mars:ncpus=3:mem=1048576kb)+(jupiter:ncpus=2:mem=2097152kb)+(saturn:ncpus=2:mem=2097152kb)+
 (mercury:ncpus=1:mem=3145728kb)+(neptune:ncpus=1:mem=3145728kb)
Resource_List.mem = 11gb
Resource_List.ncpus = 9
Resource_List.nodect = 5
Resource_List.place = scatter:excl
Resource_List.select = ncpus=3:mem=1gb+2:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb
Resource_List.site = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb

tolerate_node_failures = job_start
```

Now jupiter and neptune go down, and just before the job runs its program, the `execjob_launch` hook executes and prunes the job's vnode assignment back to the original select request. Now the job has this vnode assignment:

```
% qstat -f 20
Job Id: 20.mars.example.com
...
exec_host = mars/0*3+saturn/0*2+mercury/0*2
exec_vnode =
 (mars:ncpus=3:mem=1048576kb)+(saturn:ncpus=2:mem=2097152kb)+(mercury:ncpus=1:mem=3145728kb)
Resource_List.mem = 6gb
Resource_List.ncpus = 6
Resource_List.nodect = 3
Resource_List.place = scatter:excl
Resource_List.select = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb

Resource_List.site = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb
```

A snapshot of the job's output shows the pruned list of vnodes:

```
/var/spool/PBS/aux/20.mars.example.com <-- updated contents of $PBS_NODEFILE
mars.example.com
saturn.example.com
mercury.example.com
END

HOSTNAME tests

pbsdsh -n 0 hostname
mars.example.com
pbsdsh -n 1 hostname
saturn.example.com
pbsdsh -n 2 hostname
mercury.example.com

PBS_NODEFILE tests
HOST=mars.example.com
pbs_tmrsh mars.example.com hostname
mars.example.com
ssh mars.example.com pbs_attach -j 20.mars.example.com hostname
mars.example.com
HOST=saturn.example.com
pbs_tmrsh saturn.example.com hostname
saturn.example.com
ssh saturn.example.com pbs_attach -j 20.mars.example.com hostname
saturn.example.com
HOST=mercury.example.com
pbs_tmrsh mercury.example.com hostname
mercury.example.com
ssh mercury.example.com pbs_attach -j 20.mars.example.com hostname
mercury.example.com
```

## 8.6 Preventing Communication and Timing Problems

### 8.6.1 Introduction

PBS communicates with remote execution hosts in order to track their availability and manage the jobs running on them. PBS is dependent upon your network for this communication. If there are network outages, or if the execution node becomes too busy for MoM to be able to respond to the server's queries, PBS will not be able to function properly.

Failover can cause a delay in communication with the server and/or scheduler, which can impact whether e.g. `qsub` can connect to the server.

You can configure PBS to be better able to withstand these types of communication issues.



The following attributes and parameters control how PBS handles communication timing:

**Table 8-10: Attributes and Parameters For Communication and Timing**

| Attribute or Parameter              | Description                                                                                                                                                                                                                                                                                                     | Cross Reference                                                                                         |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>Configuration Parameters</b>     |                                                                                                                                                                                                                                                                                                                 |                                                                                                         |
| PBS_CLIENT_CONN_RETRY_INTERVAL      | Interval in seconds at which qsub attempts to establish a socket when connecting to the server before timing out.<br>Default: 30                                                                                                                                                                                | See <a href="#">section 8.6.4, “Configuring Timeout for qsub Connection”</a> , on page 414              |
| PBS_CLIENT_CONN_RETRY_TIMEOUT       | Maximum time in seconds that qsub will try to establish a socket when connecting to the server.<br>Default: 0 (disabled)                                                                                                                                                                                        | See <a href="#">section 8.6.4, “Configuring Timeout for qsub Connection”</a> , on page 414              |
| <b>Server Attributes</b>            |                                                                                                                                                                                                                                                                                                                 |                                                                                                         |
| job_requeue_timeout                 | Controls how long the process of requeueing a job is allowed to take                                                                                                                                                                                                                                            | See <a href="#">section 8.6.3, “Setting Job Requeue Timeout”</a> , on page 414                          |
| node_fail_requeue                   | Controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job's primary execution host                                                                                                                                                                   | See <a href="#">section 8.6.2, “Node Fail Requeue: Jobs on Failed Vnodes”</a> , on page 412             |
| rpp_max_pkt_check                   | Maximum number of TPP messages processed by the main server thread per iteration.<br>Default: 64                                                                                                                                                                                                                | See <a href="#">“Communication” on page 47 in the PBS Professional Installation &amp; Upgrade Guide</a> |
| rpp_retry                           | In a fault-tolerant setup (multiple pbs_comms), when the first pbs_comm fails partway through a message, this is number of times TPP tries to use any other remaining pbs_comms to send the message.<br>Integer<br>Valid values: <i>Greater than or equal to zero</i><br>Default: 10<br>Python type: <i>int</i> | See <a href="#">“Communication” on page 47 in the PBS Professional Installation &amp; Upgrade Guide</a> |
| rpp_highwater                       | This is the maximum number of messages per stream (meaning the maximum number of messages between each pair of endpoints).<br>Integer<br>Valid values: <i>Greater than or equal to one</i><br>Default: 1024<br>Python type: <i>int</i>                                                                          | See <a href="#">“Communication” on page 47 in the PBS Professional Installation &amp; Upgrade Guide</a> |
| <b>MoM Configuration Parameters</b> |                                                                                                                                                                                                                                                                                                                 |                                                                                                         |
| \$max_load                          | Vnode is considered to be <i>busy</i> if it is above this load.                                                                                                                                                                                                                                                 | See <a href="#">section 8.6.6, “Managing Load Levels on Vnodes”</a> , on page 415                       |

**Table 8-10: Attributes and Parameters For Communication and Timing**

| Attribute or Parameter  | Description                                                                   | Cross Reference                                                                         |
|-------------------------|-------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| \$ideal_load            | Vnode is considered to be not <i>busy</i> if it is below this load.           | See <a href="#">section 8.6.6, “Managing Load Levels on Vnodes”</a> , on page 415       |
| \$prologalarm           | Maximum number of seconds the prologue and epilogue may run before timing out | See <a href="#">section 8.6.7, “Prologue &amp; Epilogue Running Time”</a> , on page 417 |
| <b>Queue Attributes</b> |                                                                               |                                                                                         |
| route_retry_time        | Interval between retries at routing a job                                     | See <a href="#">section 8.6.8, “Time Between Routing Retries”</a> , on page 417         |

See [“Robust Communication” on page 53 in the PBS Professional Installation & Upgrade Guide](#).

## 8.6.2 Node Fail Requeue: Jobs on Failed Vnodes

The `node_fail_requeue` server attribute controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job's primary execution host.

### 8.6.2.1 How Node Fail Requeue Works

You can specify how long the server waits after it loses contact with the primary execution host before deleting or requeueing her jobs. This behavior is controlled by the server's `node_fail_requeue` attribute.

This attribute's value is the delay between the time the server determines that the primary execution host MoM cannot be contacted and the time it requeues the job, and does not include the time it takes to determine that the host is out of contact.

If this attribute is set to a value other than zero, and the server loses contact with an execution host, all jobs for which this is the primary execution host are requeued or deleted at the same time.

If `node_fail_requeue` is unset, and the host where primary execution is running fails, the server assumes that the job is still running until one of the following happens:

- The primary execution host MoM comes back up and tells the server to requeue the job
- The job is manually rerun

### 8.6.2.2 Effect Of Requeueing On Jobs

When a job is thus requeued, it retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job's owner has gone over their fairshare limit, etc.

Any resources that were being used by a job are freed when the job is requeued.

### 8.6.2.3 The `node_fail_requeue` Server Attribute

Format: *Integer*

#### 8.6.2.3.i Allowable Values

The `node_fail_requeue` attribute can take these values:

**Greater than zero**

The server waits for the specified number of seconds after losing contact with a primary execution host MoM, then attempts to contact the primary execution host MoM, and if it cannot, requeues any jobs that can be rerun and deletes any jobs that cannot be rerun.

**Zero**

Jobs are not requeued; they are left in the *Running* state until the execution host MoM is recovered, whether or not the server has contact with their primary execution host MoM.

**Less than zero**

The attribute is treated as if it were set to *1*, and jobs are deleted or requeued after the server has been out of contact with the primary execution host MoM for 1 second.

**Unset**

Behaves as if set to the default value of *310*.

**8.6.2.3.ii Default Value**

The default value for this attribute is *310*, meaning that when the server loses contact with an execution host, it waits for 310 seconds after losing contact with the primary execution host MoM before requeueing or deleting jobs.

**8.6.2.4 Where node\_fail\_requeue Applies**

The server's `node_fail_requeue` attribute applies only in the case where the server loses contact with the primary execution host MoM.

When the primary execution host MoM loses contact with a sister MoM, the job is immediately deleted or requeued.

**8.6.2.5 Jobs Eligible to be Requeued**

Jobs are eligible to be requeued if they meet either of the following criteria:

- The job's `Rerunable` attribute is set to *y*
- The job did not begin execution, for example:
  - a multi-host job did not start on one or more vnodes
  - provisioning failed for the job

Jobs are ineligible to be requeued if their `Rerunable` attribute is set to *n* and they have started execution.

See [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#) and [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

**8.6.2.6 Using node\_fail\_requeue**

The number of seconds selected should be long enough to exceed any transient non-vnode failures, but short enough to requeue the job in a timely fashion. Transient non-vnode failures can prevent MoM from reporting back to the server before the server marks the vnode *down*. These include:

- Network outages
- Vnode is too busy to respond, perhaps due to heavy swapping

Using this feature requires that you take the following into account:

- If the host where the primary execution host MoM is running fails, and `node_fail_requeue` is unset, the server assumes that the job is still running until one of the following happens:
  - The primary execution host MoM comes back up and tells the server to requeue the job
  - The job is manually rerun

If your site has hosts that fail and are not monitored, failed jobs may go unnoticed for a long time.

- If your network has temporary failures, and `node_fail_requeue` is set to a duration shorter than the outage, jobs will be unnecessarily requeued. This can be especially annoying when the job has been running for days.

### 8.6.2.7 Advice and Caveats

- If your site experiences frequent network failures or your execution hosts are often too busy to respond to the server, it is recommended that you either set `node_fail_requeue` to a value greater than the time MoM is unavailable, or set it to `zero`. This way jobs won't be requeued just because the network had a temporary outage or the vnode was too busy. Choose a value greater than both the longest likely network outage time and the time MoM is unavailable. For example, one site has set the value to 10 minutes, and another has set it to 15 minutes (900 seconds) to avoid problems due to swapping.
- The value shown in the server log for the time between losing communication and requeueing a job is sometimes one or two seconds less than the specified value.
- If the server is restarted when `node_fail_requeue` is set to a given value, `node_fail_requeue` retains that value. If the server is started when `node_fail_requeue` is unset, `node_fail_requeue` reverts to its default value.

### 8.6.3 Setting Job Requeue Timeout

When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. The time for requeueing includes post-processing such as staging files out, deleting files, and changing the job's state from *R* to *Q*. See [section 4.9.33, “Using Preemption”, on page 180](#). The time allowed for a job to be requeued is controlled by the `job_requeue_timeout` server attribute.

You can use `qmgr` to set the `job_requeue_timeout` server attribute to a value that works for the jobs at your site. This attribute is of type `Duration`, with a minimum allowed value of 1 second and a maximum allowed value of 3 hours. The default timeout is 45 seconds. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

### 8.6.4 Configuring Timeout for qsub Connection

The `qsub` command may require extra time to connect to the server if failover is happening, the server is being restarted, or your network has glitches.

When `qsub` connects to the PBS server, it first establishes a socket, then it waits for up to 600 seconds for a reply from the server.

You can configure the maximum amount of time that `qsub` will attempt to establish a socket while connecting to the server in the `PBS_CLIENT_CONN_RETRY_TIMEOUT` configuration parameter in `pbs.conf`. The default value for this parameter is 0, meaning that the timeout is disabled (if `qsub` is not immediately successful in establishing a socket, it gives up).

You can configure the interval at which `qsub` attempts to establish a socket while it has not yet succeeded, in the `PBS_CLIENT_CONN_RETRY_INTERVAL` configuration parameter in `pbs.conf`. The default value for this parameter is 30 seconds.

We recommend setting the `PBS_CLIENT_CONN_RETRY_TIMEOUT` configuration parameter to slightly longer than the expected time required for failover to take place.

We recommend setting the `PBS_CLIENT_CONN_RETRY_INTERVAL` configuration parameter to a time interval that will not cause undue burden on the network; consider the number of simultaneous `qsubs` when setting it.

Note that `PBS_CLIENT_CONN_RETRY_TIMEOUT` and `PBS_CLIENT_CONN_RETRY_INTERVAL` take effect only when `PBS_CLIENT_CONN_RETRY_TIMEOUT > 0` and `>= PBS_CLIENT_CONN_RETRY_INTERVAL`.

## 8.6.5 Setting MoM Reconnection Timeout

When the primary execution host MoM detects that a sister mom has lost connectivity (e.g. MoM went down or the network is having problems) it waits for a specified amount of time for the sister to reconnect before it gives up and kills the job. You can configure the time the primary execution host MoM waits by setting MoM's `$max_poll_downtime` parameter in `PBS_HOME/mom_priv/config`. The default value is five minutes.

## 8.6.6 Managing Load Levels on Vnodes

An overloaded execution host may end up too busy for MoM to respond to the server's queries, and causing the server to mark the MoM as *down*.

PBS can track the state of each execution host, running new jobs on the host according to whether the host is marked *busy* or not.

This behavior is somewhat different from load balancing, described in [section 4.9.27, “Using Load Balancing”, on page 159](#). In load balancing, the scheduler estimates how much load a job would produce, and will not place a job where doing so would put the load above the limit. When managing load levels on vnodes as described here, the scheduler uses the state of the vnode to determine whether to place a job on that vnode.

The state of the vnode is set by MoM, according to its load. You can set two load levels using the `$max_load` and `$ideal_load` MoM configuration parameters. When the load goes above `$max_load`, the vnode is marked as *busy*. When the load drops below `$ideal_load`, the vnode is marked *free*.

PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked *busy*
- Vnodes whose resources, such as `ncpus`, are already fully allocated
- Vnodes where the load is above `$max_load`, when load balancing is turned on. See [section 4.9.27, “Using Load Balancing”, on page 159](#).
- Vnodes where running the job would cause the load to go above `$max_load`, when load balancing is turned on. See [section 4.9.27, “Using Load Balancing”, on page 159](#).

The load used by MoM is the following:

- On Linux, it is the raw one-minute averaged "loadave" returned by the operating system
- On Windows, it is based on the processor queue length

The `$max_load` and `$ideal_load` MoM configuration parameters are also used for cycle harvesting (see [section 4.9.9.6, “Cycle Harvesting Based on Load Average”, on page 124](#)) and load balancing (see [section 4.9.27, “Using Load Balancing”, on page 159](#).)

MoM checks the load average on her host every 10 seconds.

When a vnode's state changes, for example from *free* to *busy*, MoM informs the server.

### 8.6.6.1 Techniques for Managing Load

Whether or not you set `$max_load`, PBS will not run jobs requesting a total of more than the available number of CPUs, which is set in `resources_available.ncpus`. So for example if `resources_available.ncpus` is set to 4, and a job running on the vnode has requested 2 CPUs, PBS will not run jobs requesting a total of more than 2 CPUs.

### 8.6.6.1.i Types of Workload

How you manage load depends on your workload. Some jobs do not lend themselves to sharing CPUs, but some jobs can share CPUs without being hindered. Most MPI jobs would be hindered if some processes had to wait because others were slowed by sharing a CPU. If you need a job to have reproducible timing, it cannot share a CPU. Certain single-vnode jobs that alternate between CPU usage and I/O can share a CPU without being slowed significantly, thereby increasing throughput.

### 8.6.6.1.ii How Not To Share CPUs

For vnodes primarily running jobs that would be slowed or invalidated by sharing a CPU, have PBS assign jobs according to the number of available CPUs, so that there is no sharing of CPUs. Set `resources_available.ncpus` to the number of available CPUs. Do not set `$max_load` or `$ideal_load`.

### 8.6.6.1.iii How To Share CPUs

For vnodes running only jobs that can share CPUs, you can have PBS manage jobs according to the load on the vnodes, not the number of CPUs. This is called oversubscribing the CPUs. Set `resources_available.ncpus` to a value greater than the actual number of CPUs, such as two or three times the actual number. Set `$max_load` to a reasonable value so that PBS will run new jobs until `$max_load` is reached. Set `$ideal_load` to the minimum load that you want on the vnode.

### 8.6.6.1.iv Suspending Jobs on Overloaded Vnodes

You can specify that MoM should suspend jobs when the load goes above `$max_load`, by adding the `suspend` argument to the `$max_load` parameter. See [section , “\\$max\\_load <load> \[suspend\]”, on page 417](#). In this case, MoM suspends all jobs on the vnode until the load drops below `$ideal_load`, then resumes them. This option is useful only when the source of the load includes work other than PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between being overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, being overloaded, and so on.

## 8.6.6.2 Caveats and Recommendations

- It is recommended that the value for `$ideal_load` be lower than the value for `$max_load`. The value for `$ideal_load` should be low enough that new jobs are not run before existing jobs are done using the vnode's spare load.
- If you set only one of `$max_load` and `$ideal_load`, for example you set `$max_load`, but not `$ideal_load`, PBS sets the other to the same value.
- Do not allow reservations on hosts where `$max_load` and `$ideal_load` are configured. Set the `resv_enable` vnode attribute on these hosts to *False*.
- If you are using cycle harvesting via load balancing, be careful with the settings for `$ideal_load` and `$max_load`. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working. See [section 4.9.9.6, “Cycle Harvesting Based on Load Average”, on page 124](#).

### 8.6.6.2.i Allowing Non-job Processes on Execution Host

If you wish to run non-PBS processes on a host, you can prevent PBS from using more than you want on that host. Set the `$ideal_load` and `$max_load` MoM configuration parameters to values that are low enough to allow other processes to use some of the host.

### 8.6.6.3 Load Configuration Parameters

**\$ideal\_load <load>**

MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the **\$max\_load** parameter.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

**\$max\_load <load> [suspend]**

MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the **\$ideal\_load** parameter.

If the optional **suspend** argument is specified, PBS suspends jobs running on the vnode when the load average exceeds **\$max\_load**, regardless of the source of the load (PBS and/or logged-in users).

Example:

```
$max_load 3.5
```

Format: *Float*

Default: number of CPUs

## 8.6.7 Prologue & Epilogue Running Time

Each time the scheduler runs a job, it waits for the prologue to finish before it runs another job. In order to prevent a hung prologue from halting job execution, prologues and epilogues are only allowed to run for a specified amount of time before PBS kills them. The running time is specified in the **\$prologalarm** MoM configuration parameter. The default value for this parameter is *30 seconds*.

### 8.6.7.1 Prologue Timeout Configuration Parameter

**\$prologalarm <timeout>**

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

```
$prologalarm 30
```

Format: *Integer*

Default: *30*

## 8.6.8 Time Between Routing Retries

If the network is flaky, PBS may not be able to route a job from a routing queue to the destination queue. If all destination queues for a routing queue are at capacity, a job in a routing queue remains where it is. The time between routing retries is controlled by the **route\_retry\_time** queue attribute.

If the network experiences long outages, you may wish to set the time between retries to a sufficiently long time that PBS is not wasting cycles attempting to route jobs.

If jobs in a routing queue are not being routed because the destination queues are full, and most jobs are long-running jobs, you may wish to set the time between retries so that attempts are infrequent. It is recommended that the time between retries be no longer than the longest time acceptable to have an open slot in an execution queue.



### 8.6.8.1 Routing Retry Attribute

`route_retry_time`

Time delay between routing retries. Typically used when the network between servers is down. Used only with routing queues.

Format: *Integer seconds*

Default: *30 seconds*

Python type: `pbs.duration`

## 8.7 Preventing File System Problems

### 8.7.1 Avoid Filling Location of Temp Files for PBS Components

When the location used by PBS components to store temporary files becomes full, various failures may result, including jobs not initializing properly. To help avoid this, you can set the root directory for these files to a location less likely to fill up. See [section 9.9, “Temporary File Location for PBS Components”, on page 450](#).

In addition, we recommend periodic cleaning of this location.

### 8.7.2 Avoid Filling Filesystem with Log Files

You must avoid having log files fill up the available space. You may have to rotate and archive log files frequently to ensure that adequate space remains available. See [“Adequate Space for Logfiles” on page 8 in the PBS Professional Installation & Upgrade Guide](#).

## 8.8 OOM Killer Protection

PBS automatically protects against OOM killers. If the system hosting a PBS daemon or data service runs low on memory, the system may use an out-of-memory killer (OOM killer) to terminate processes. The PBS daemons and data service are protected from being terminated by an OOM killer.

# 9

# Administration

## Contents

|       |                                                                              |        |
|-------|------------------------------------------------------------------------------|--------|
| 9.1   | Specifying Scheduler Username                                                | AG-420 |
| 9.1.1 | Steps for Changing Scheduler Username                                        | AG-421 |
| 9.2   | The PBS Configuration File                                                   | AG-421 |
| 9.2.1 | Location of Configuration File                                               | AG-421 |
| 9.2.2 | Format of Configuration File                                                 | AG-422 |
| 9.2.3 | Example of Configuration File                                                | AG-422 |
| 9.2.4 | Contents of Configuration File                                               | AG-422 |
| 9.2.5 | Configuration File Caveats and Recommendations                               | AG-427 |
| 9.3   | Environment Variables                                                        | AG-427 |
| 9.3.1 | Environment Variables For Daemons, Commands, and Jobs                        | AG-428 |
| 9.3.2 | Job-specific Environment Variables                                           | AG-428 |
| 9.4   | Event Logging                                                                | AG-429 |
| 9.4.1 | PBS Events                                                                   | AG-429 |
| 9.4.2 | Event Logfiles                                                               | AG-429 |
| 9.4.3 | Log Levels                                                                   | AG-429 |
| 9.4.4 | Event Logfile Format and Contents                                            | AG-432 |
| 9.4.5 | Logging Job Usage                                                            | AG-433 |
| 9.4.6 | Managing Log Files                                                           | AG-433 |
| 9.4.7 | Extracting Logged Information                                                | AG-434 |
| 9.4.8 | Using the Linux <code>syslog</code> Facility                                 | AG-434 |
| 9.5   | Managing Machines                                                            | AG-435 |
| 9.5.1 | Offlining Hosts and Vnodes                                                   | AG-435 |
| 9.5.2 | Performing Maintenance on Powered-up Vnodes                                  | AG-436 |
| 9.5.3 | Changing Hostnames or IP Addresses                                           | AG-437 |
| 9.5.4 | Discovering Last Reboot Time of Server                                       | AG-438 |
| 9.5.5 | Changing Network Configuration                                               | AG-438 |
| 9.5.6 | Replacing or Reimaging Nodes                                                 | AG-438 |
| 9.5.7 | Restricting User Access to Execution Hosts                                   | AG-438 |
| 9.6   | Managing the Data Service                                                    | AG-439 |
| 9.6.1 | PBS Monitors Data Service                                                    | AG-439 |
| 9.6.2 | Data Service Accounts                                                        | AG-439 |
| 9.6.3 | Data Service Account Password                                                | AG-439 |
| 9.6.4 | Starting and Stopping the Data Service                                       | AG-440 |
| 9.6.5 | Changing Data Service Port                                                   | AG-441 |
| 9.6.6 | File Ownership                                                               | AG-441 |
| 9.7   | Setting File Transfer Mechanism                                              | AG-441 |
| 9.7.1 | Letting MoM Know Whether Transfer is Local or Remote                         | AG-441 |
| 9.7.2 | Specifying Local File Transfer Mechanism                                     | AG-442 |
| 9.7.3 | Specifying Remote File Transfer Mechanism                                    | AG-443 |
| 9.7.4 | Options Passed to File Transfer Commands                                     | AG-444 |
| 9.7.5 | Using Custom File Transfer Mechanism                                         | AG-444 |
| 9.7.6 | When Multiple Attempts Are Required                                          | AG-446 |
| 9.7.7 | Allowing Direct Write of Standard Output and Error to <code>/dev/null</code> | AG-446 |
| 9.7.8 | Troubleshooting File Transfer                                                | AG-446 |

|        |                                                                  |        |
|--------|------------------------------------------------------------------|--------|
| 9.7.9  | Advice on Improving File Transfer Performance. . . . .           | AG-447 |
| 9.7.10 | General Advice on File Transfer . . . . .                        | AG-448 |
| 9.8    | Some Performance Tips . . . . .                                  | AG-449 |
| 9.8.1  | Configurable Job Dispatch Rate. . . . .                          | AG-449 |
| 9.8.2  | Improving Scheduling Performance. . . . .                        | AG-450 |
| 9.8.3  | Improving Communication Performance. . . . .                     | AG-450 |
| 9.8.4  | Improving Hook Speed. . . . .                                    | AG-450 |
| 9.9    | Temporary File Location for PBS Components . . . . .             | AG-450 |
| 9.9.1  | Default Location for Temporary Files . . . . .                   | AG-450 |
| 9.9.2  | Configuring Temporary File Location for PBS Components . . . . . | AG-450 |
| 9.9.3  | Requirements . . . . .                                           | AG-451 |
| 9.9.4  | Advice and Recommendations for Temporary File Location . . . . . | AG-451 |
| 9.10   | Administration Caveats . . . . .                                 | AG-452 |
| 9.10.1 | General Caveats . . . . .                                        | AG-452 |
| 9.10.2 | Windows Caveats . . . . .                                        | AG-452 |
| 9.11   | Support for Globus . . . . .                                     | AG-452 |
| 9.12   | Support for Hyperthreading . . . . .                             | AG-452 |
| 9.12.1 | Linux Machines with HTT . . . . .                                | AG-452 |
| 9.12.2 | Windows Machines with HTT . . . . .                              | AG-452 |
| 9.12.3 | Using Number of Physical CPUs . . . . .                          | AG-452 |
| 9.12.4 | Hyperthreading Caveats . . . . .                                 | AG-452 |
| 9.13   | How To.... . . . .                                               | AG-453 |
| 9.13.1 | How to Drain Jobs . . . . .                                      | AG-453 |
| 9.13.2 | How to Find Out Which Daemons Should Be Running. . . . .         | AG-453 |
| 9.14   | Avoiding Memory Hogging by Python Scripts. . . . .               | AG-453 |
| 9.14.1 | Using Garbage Collection . . . . .                               | AG-453 |
| 9.14.2 | Restarting the Python Interpreter . . . . .                      | AG-453 |
| 9.14.3 | Attributes for Managing Python Memory Usage. . . . .             | AG-454 |

## 9.1 Specifying Scheduler Username

By default, the PBS daemons run as root. However, you can specify that the scheduler should run as some other user. You can do this either by setting `PBS_DAEMON_SERVICE_USER` in the environment when doing an `rpm` install, or by specifying the username in the `PBS_DAEMON_SERVICE_USER` parameter in `/etc/pbs.conf`.

Certain directory and file permissions need to be compatible with the scheduler username:

- Server dynamic resource scripts (`server_dyn_res`) need to be owned, readable, and writable by `PBS_DAEMON_SERVICE_USER`
- When a scheduler starts, it verifies that `sched_priv` and `sched_logs` are owned and can be read and executed by `PBS_DAEMON_SERVICE_USER`. If the file or directory permissions are wrong, it logs an error and exits
- The `pbsfs` command sets fairshare usage, and writes a file that needs to be readable by `PBS_DAEMON_SERVICE_USER`
- The `habitat` script creates `sched_priv` and `sched_log`, both which need to be owned, readable, and writable by `PBS_DAEMON_SERVICE_USER`

If you set the scheduler username before starting PBS, the scheduler(s) and the `pbsfs` command automatically run as `PBS_DAEMON_SERVICE_USER`, even if they are started as root.

If you change the scheduler username after starting PBS, you must restart the server and scheduler(s), and make sure that PBS daemons and commands can read and write necessary files and directories.

You can use the `pbs_probe` command to check and fix the permissions for the `sched_priv` and `sched_logs` directories.

### 9.1.1 Steps for Changing Scheduler Username

1. Stop server and scheduler(s):  
`systemctl stop pbs`  
or  
`/etc/init.d/pbs stop`
2. Change `PBS_DAEMON_SERVICE_USER` parameter in `/etc/pbs.conf`:  
`PBS_DAEMON_SERVICE_USER=<scheduler username>`
3. Change ownership and permissions on any `server_dyn_res` scripts:  
`chown ...`  
`chmod ...`
4. Run the `pbs_probe` command to fix any permissions problems. On each host:  
`pbs_probe -f`
5. Restart the server and scheduler(s):  
`systemctl start pbs`  
or  
`/etc/init.d/pbs start`
6. If other PBS complexes are peering with this complex, make sure that `PBS_DAEMON_SERVICE_USER` is a manager on the servers for those complexes

## 9.2 The PBS Configuration File

During the installation of PBS Professional, the installation script creates a configuration file named `pbs.conf`. This configuration file controls which daemons are to run on the local system, the directory tree location, and various runtime configuration options. Each host in a complex should have its own `pbs.conf` file.

### 9.2.1 Location of Configuration File

The configuration file is located in one of the following:

Linux:

`/etc/pbs.conf`

Windows:

`[PBS Destination Folder]\pbs.conf`

where `[PBS Destination Folder]` is the path specified when PBS is installed on the Windows platform, for example:

`C:\Program Files\PBS\pbs.conf`

or

`C:\Program Files (x86)\PBS\pbs.conf`

You can set the value of `PBS_CONF_FILE` in your environment in order to specify an alternate location for `pbs.conf`.

## 9.2.2 Format of Configuration File

Each line in the `/etc/pbs.conf` file gives a value for one parameter, or is a comment, or is blank. The order of the elements is not important.

### 9.2.2.1 Specifying Parameters

When you specify a parameter value, do not include any spaces in the line. Format for specifying a parameter value:

```
<parameter>=<value>
```

For example, to specify a value for `PBS_START_MOM` on the local host:

```
PBS_START_MOM=1
```

### 9.2.2.2 Comment Lines in Configuration File

You can comment out lines you are not using. Precede a comment with the hashmark ("`#`"). For example:

```
#This is a comment line
```

## 9.2.3 Example of Configuration File

The following is an example of a `pbs.conf` file for a host which is to run the server, the scheduler, and a MoM. The server runs on the host named `Host1.ExampleDomain`.

```
PBS_EXEC=/opt/pbs/M.N.P.S
PBS_HOME=/var/spool/PBS
PBS_START_SERVER=1
PBS_START_MOM=1
PBS_START_SCHED=1
PBS_SERVER=Host1.ExampleDomain
```

## 9.2.4 Contents of Configuration File

The `/etc/pbs.conf` file contains configuration parameters for PBS. The following table describes the parameters you can use in the `pbs.conf` configuration file:

**Table 9-1: Parameters in `pbs.conf`**

| Parameter                               | Description                                                                                                                                                                                         |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>PBS_AUTH_METHOD</code>            | Specifies default authentication method and library to be used by PBS. Used only at authenticating client. Case-insensitive.<br>Default value: <i>resvport</i><br>To use MUNGE, set to <i>munge</i> |
| <code>PBS_BATCH_SERVICE_PORT</code>     | Port on which server listens. Default: <i>15001</i>                                                                                                                                                 |
| <code>PBS_BATCH_SERVICE_PORT_DIS</code> | DIS port on which server listens.                                                                                                                                                                   |

**Table 9-1: Parameters in pbs.conf**

| Parameter                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PBS_CLIENT_CONN_RETRY_INTERVAL | Interval in seconds at which qsub attempts to establish a socket when connecting to the server before timing out.<br><br>This attribute takes effect only when PBS_CLIENT_CONN_RETRY_TIMEOUT is > 0 and is >= this attribute.<br><br>Overridden by the PBS_CLIENT_CONN_RETRY_INTERVAL environment variable.<br><br>Default: 30                                                                                                                                                    |
| PBS_CLIENT_CONN_RETRY_TIMEOUT  | Maximum time in seconds that qsub will try to establish a socket when connecting to the server.<br><br>When this attribute is set to 0 (zero), PBS_CLIENT_CONN_RETRY_INTERVAL is disabled (no retry, meaning that qsub immediately gives up trying to establish a socket).<br><br>This attribute takes effect only when it is >0 and >= PBS_CLIENT_CONN_RETRY_INTERVAL.<br><br>Overridden by the PBS_CLIENT_CONN_RETRY_TIMEOUT environment variable.<br><br>Default: 0 (disabled) |
| PBS_COMM_LOG_EVENTS            | Communication daemon log mask. Default: 511                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| PBS_COMM_ROUTERS               | Tells a pbs_comm the location of the other pbs_comms .                                                                                                                                                                                                                                                                                                                                                                                                                            |
| PBS_COMM_THREADS               | Number of threads for communication daemon.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| PBS_CORE_LIMIT                 | Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string "unlimited". If unset, core file size limit is inherited from the shell environment.                                                                                                                                                                                                                                                                                            |
| PBS_CP                         | Specifies command for MoM to use for local copy                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| PBS_DAEMON_SERVICE_USER        | Username under which scheduler(s) run. Default: root                                                                                                                                                                                                                                                                                                                                                                                                                              |
| PBS_DATA_SERVICE_PORT          | Used to specify non-default port for connecting to data service. Default: 15007                                                                                                                                                                                                                                                                                                                                                                                                   |
| PBS_ENCRYPT_METHOD             | Specifies method and library for encrypting and decrypting data in client-server communication. Used only at authentication client side. Case-insensitive.<br><br>To use TLS encryption in client-server communication, set this parameter to <i>tls</i> .<br><br>No default; if this is not set, PBS does not encrypt or decrypt data.                                                                                                                                           |
| PBS_ENVIRONMENT                | Location of pbs_environment file.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| PBS_EXEC                       | Location of PBS bin and sbin directories.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| PBS_HOME                       | Location of PBS working directories.                                                                                                                                                                                                                                                                                                                                                                                                                                              |

Table 9-1: Parameters in pbs.conf

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PBS_LEAF_NAME             | <p>Tells endpoint what hostname to use for network.</p> <p>The value does not include a port, since that is usually set by the daemon.</p> <p>By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network.</p> <p>The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in /etc/pbs.conf to something else, make sure you set the Mom attribute at vnode creation.</p> <p>TPP internally resolves the name to a set of IP addresses, so you do not affect how pbs_comm works.</p> |
| PBS_LEAF_ROUTERS          | Location of endpoint's pbs_comm daemon(s).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| PBS_LOCALLOG=<value>      | <p>Enables logging to local PBS log files. Valid values:</p> <p>0: no local logging</p> <p>1: local logging enabled</p> <p>Only available when using syslog.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| PBS_LOG_HIGHRES_TIMESTAMP | <p>Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is <i>HH:MM:SS</i>. With microsecond logging, format is <i>HH:MM:SS:XXXXXX</i>.</p> <p>Does not affect accounting log. Not applicable when using syslog.</p> <p>Overridden by environment variable of the same name.</p> <p>Valid values: 0, 1. Default: 0 (no microsecond logging)</p>                                                                                                                                                                                                                                                                                                                                                       |
| PBS_MAIL_HOST_NAME        | <p>Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute.</p> <p>Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). For how this is used in email address, see <a href="#">section 2.2.3, “Specifying Mail Delivery Domain”, on page 22</a>.</p>                                                                                                                                                                                                                                                                                                                                                                         |
| PBS_MANAGER_SERVICE_PORT  | Port on which MoM listens. Default: 15003                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| PBS_MOM_HOME              | Location of MoM working directories.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| PBS_MOM_NODE_NAME         | <p>Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by <code>gethostname()</code>.</p> <p>If you use the IP address for a vnode name, set PBS_MOM_NODE_NAME=&lt;IP address&gt; in pbs.conf on the execution host.</p> <p>Dots are not allowed in this parameter unless they are part of an IP address.</p>                                                                                                                                                                                                                                                                                                                             |
| PBS_MOM_SERVICE_PORT      | Port on which MoM listens. Default: 15002                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |



**Table 9-1: Parameters in pbs.conf**

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PBS_OUTPUT_HOST_NAME | <p>Host to which all job standard output and standard error are delivered. If specified in <code>pbs.conf</code> on a job submission host, the value of <code>PBS_OUTPUT_HOST_NAME</code> is used in the host portion of the job's <code>Output_Path</code> and <code>Error_Path</code> attributes. If the job submitter does not specify paths for standard output and standard error, the current working directory for the <code>qsub</code> command is used, and the value of <code>PBS_OUTPUT_HOST_NAME</code> is appended after an at sign ("@"). If the job submitter specifies only a file path for standard output and standard error, the value of <code>PBS_OUTPUT_HOST_NAME</code> is appended after an at sign ("@"). If the job submitter specifies paths for standard output and standard error that include host names, the specified paths are used.</p> <p>Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See <a href="#">"Delivering Output and Error Files" on page 61 in the PBS Professional Administrator's Guide</a>.</p> |
| PBS_PRIMARY          | <p>Hostname of primary server. Used only for failover configuration. Overrides <code>PBS_SERVER_HOST_NAME</code>.</p> <p>If you set <code>PBS_LEAF_NAME</code> on the primary server host, make sure that <code>PBS_PRIMARY</code> matches <code>PBS_LEAF_NAME</code> on the corresponding host. If you do not set <code>PBS_LEAF_NAME</code> on the server host, make sure that <code>PBS_PRIMARY</code> matches the hostname of the server host.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| PBS_RCP              | Location of <code>rcp</code> command if <code>rcp</code> is used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| PBS_REMOTE_VIEWER    | <p>Specifies remote viewer client.</p> <p>If not specified, PBS uses native Remote Desktop client for remote viewer.</p> <p>Set on submission host(s).</p> <p>Supported on Windows only.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| PBS_SCHED_THREADS    | <p>Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable.</p> <p>Overridden by <code>pbs_sched -t</code> option and <code>PBS_SCHED_THREADS</code> environment variable.</p> <p>Default: 1</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| PBS_SCP              | Location of <code>scp</code> command if <code>scp</code> is used; setting this parameter causes PBS to first try <code>scp</code> rather than <code>rcp</code> for file transport.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| PBS_SECONDARY        | <p>Hostname of secondary server. Used only for failover configuration. Overrides <code>PBS_SERVER_HOST_NAME</code>.</p> <p>If you set <code>PBS_LEAF_NAME</code> on the secondary server host, make sure that <code>PBS_SECONDARY</code> matches <code>PBS_LEAF_NAME</code> on the corresponding host. If you do not set <code>PBS_LEAF_NAME</code> on the server host, make sure that <code>PBS_SECONDARY</code> matches the hostname of the server host.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

Table 9-1: Parameters in pbs.conf

| Parameter                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PBS_SERVER                 | <p>Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in <code>pbs.conf</code>. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER.</p> <p>Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.</p> |
| PBS_SERVER_HOST_NAME       | <p>The FQDN of the server host. Used by clients to contact server. Overridden by PBS_PRIMARY and PBS_SECONDARY failover parameters. Overrides PBS_SERVER parameter. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See <a href="#">"Contacting the Server" on page 61 in the PBS Professional Administrator's Guide</a>.</p>                                                                  |
| PBS_START_COMM             | Set this to <code>1</code> if a communication daemon is to run on this host.                                                                                                                                                                                                                                                                                                                                                                |
| PBS_START_MOM              | Default is <code>0</code> . Set this to <code>1</code> if a MoM is to run on this host.                                                                                                                                                                                                                                                                                                                                                     |
| PBS_START_SCHED            | <b>Deprecated.</b> Set this to <code>1</code> if default scheduler is to run on this host. Overridden by scheduler's <code>scheduling</code> attribute.                                                                                                                                                                                                                                                                                     |
| PBS_START_SERVER           | Set this to <code>1</code> if server is to run on this host.                                                                                                                                                                                                                                                                                                                                                                                |
| PBS_SUPPORTED_AUTH_METHODS | <p>Specifies supported authentication methods for client-server communication. Used by authenticating server (PBS server, scheduler, MoM, or comm); ignored at client. Case-insensitive.</p> <p>If this parameter is set, PBS accepts only the methods listed.</p> <p>Format: comma-separated list of authentication methods.</p> <p>Default value: <i>resvport</i></p> <p>Example: <i>munge,GSS</i></p>                                    |
| PBS_SYSLOG=<value>         | <p>Controls use of <code>syslog</code> facility under which the entries are logged.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>0: no syslogging</li> <li>1: logged via LOG_DAEMON facility</li> <li>2: logged via LOG_LOCAL0 facility</li> <li>3: logged via LOG_LOCAL1 facility</li> <li>...</li> <li>9: logged via LOG_LOCAL7 facility</li> </ul>                                                                    |
| PBS_SYSLOGSEVR=<value>     | <p>Filters <code>syslog</code> messages by severity. Valid values:</p> <ul style="list-style-type: none"> <li>0: only LOG_EMERG messages are logged</li> <li>1: messages up to LOG_ALERT are logged</li> <li>...</li> <li>7: messages up to LOG_DEBUG are logged</li> </ul>                                                                                                                                                                 |

**Table 9-1: Parameters in pbs.conf**

| Parameter       | Description                                                                               |
|-----------------|-------------------------------------------------------------------------------------------|
| PBS_TMPDIR      | Location of temporary files/directories used by PBS components.                           |
| PBS_WEBAPI_PORT | Port where PBS server provides API endpoint.<br>Default: 8080                             |
| PBS_WEBAPI_TLS  | Boolean. When set to <b>1</b> , the PBS server listens to the API endpoint in HTTPS mode. |

For information on how to use the `pbs.conf` file when configuring PBS for failover, see [section 8.2.5.2, “Configuring the pbs.conf File for Failover”](#), on page 378.

## 9.2.5 Configuration File Caveats and Recommendations

- Each parameter in `pbs.conf` can also be expressed as an environment variable.
- Environment variables override `pbs.conf` parameter settings.
- When you change a setting in a `pbs.conf` file, you must restart the daemon that reads the file.
- If you specify a location for `PBS_HOME` in the shell environment, make sure that this agrees with that specified in `pbs.conf`.
- Do not change a hostname without updating the corresponding Version 2 configuration file.
- Use a name for the server in the `PBS_SERVER` variable in the `pbs.conf` file that is not longer than 255 characters. If the short name for the server resolves to the correct host, you can use this in `pbs.conf` as the value of `PBS_SERVER`. However, if the fully-qualified domain name is required in order to resolve to the correct host, then this must be the value of the `PBS_SERVER` variable.
- If you set `PBS_LEAF_NAME` on a primary or secondary server host, make sure that `PBS_PRIMARY` and `PBS_SECONDARY` match `PBS_LEAF_NAME` on the corresponding host. If you do not set `PBS_LEAF_NAME` on a server host, make sure that `PBS_PRIMARY` and `PBS_SECONDARY` match the hostnames of the server hosts.
- The server only queries for the canonicalized address of the MoM host, unless you let it know via the `Mom` attribute; if you have set `PBS_LEAF_NAME` in `/etc/pbs.conf` to something else, make sure you set the `Mom` attribute at vnode creation.
- Do not include shell-style comments in the configuration file.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

## 9.3 Environment Variables

PBS sets environment variables for different purposes: some variables are used by the daemons, commands, and jobs, and some environment variables are set individually for each job. Each parameter in `pbs.conf` can also be expressed as an environment variable. Environment variables override `pbs.conf` parameters.

---

## 9.3.1 Environment Variables For Daemons, Commands, and Jobs

The PBS installer creates an environment file called `pbs_environment`. This file is used by the daemons, commands, and jobs:

- Each PBS daemon initializes its environment using this environment file
- Several commands use environment variables to determine things like the name of the default server. The environment file is useful for setting environment variables for `mpirun`, etc.
- Jobs inherit the contents of this environment file before they acquire settings from `.profile` and `.login` files. Job scripts can use the environment variables set in the job's environment.

You can edit the environment file.

### 9.3.1.1 Contents of Environment File

When this file is created, it contains the following:

```
TZ=<local timezone, e.g. US/Pacific>
PATH=/bin:/usr/bin
```

For a list of PBS environment variables, see [“PBS Environment Variables” on page 397 of the PBS Professional Reference Guide](#).

To support X forwarding, edit MoM's `PATH` variable to include the directory containing the `xauth` utility.

### 9.3.1.2 Location of Environment File

The PBS environment file is located here:

```
PBS_HOME/pbs_environment
```

### 9.3.1.3 Environment File Requirements

You must restart each daemon after making any changes to the environment file.

### 9.3.1.4 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

## 9.3.2 Job-specific Environment Variables

For each job, the `qsub` command creates environment variables beginning with `PBS_O_`, and puts them in the job's environment. They are not written to `pbs_environment`. The server sets some of these environment variables if the `qsub` command does not set them.

For each job, the MoM on the primary execution host creates a file of the hosts to be used by the job. The node file is put in the job's environment, but the host list is not written to `pbs_environment`. The location of the node file is specified in the `PBS_NODEFILE` environment variable, which is set for the job only. See [“The Job Node File”, on page 79 of the PBS Professional User's Guide](#).

Some environment variables are set by commands. The PBS `mpiexec` script sets `PBS_CPUSSET_DEDICATED`.

For a list of environment variables used and set by the `qsub` command, see [“Environment Variables” on page 233 of the PBS Professional Reference Guide](#).

## 9.4 Event Logging

PBS provides event logging for the server, the scheduler, the communication daemon, and each MoM. You can use logfiles to monitor activity in the PBS complex.

### 9.4.1 PBS Events

The amount and type of output in the PBS event logfiles depends on the specified log filters for each component. Each PBS daemon can be directed to record only messages pertaining to certain levels of importance, called log levels. The specified log levels are logically "or-ed" to produce a mask representing the events to be logged by the daemon. The hexadecimal value for each log level is shown in [section 9.4.3, "Log Levels", on page 429](#). When events appear in the log file, they are tagged with their hexadecimal value, without a preceding "0x".

### 9.4.2 Event Logfiles

Each PBS daemon writes a separate event logfile. Each multisched writes its own logfile. By default, each daemon writes a file that has the current date as its name in the `PBS_HOME/<component>_logs` directory. The location of the logfile can be overridden with the `-L` option to each daemon's command. For example, to override the server's logfile location:

```
pbs_server -L <new path>
```

Whenever a new log file is opened, the daemon logs `PBS_LEAF_NAME`, `PBS_MOM_NODE_NAME`, and the host-name. The daemon also logs all network interfaces, listing each interface and all of the hostnames associated with that interface. In addition, it logs the PBS version and the build information.

Each daemon closes the day's log file and opens a new log file on the first message written after midnight. If no messages are written, the old log file stays open. Each daemon closes and reopens the same logfile when the daemon receives a `SIGHUP`.

Each daemon writes its version and build information to its event logfile each time it is started or restarted, and also when the logfile is automatically rotated out. The version and build information appear in individual records. These records contain the following substrings:

```
pbs_version = <PBSPro_stringX.stringY.stringZ.5-digit seq>
build = <status line from config.status, etc>
```

Example:

```
pbs_version = PBSPro_9.2.0.63106
build = '--set-cflags=-g -O0' --enable-security=KCRYPT ...
```

If the daemon cannot write to its log file, it writes the error message to the console. Some errors that appear before the daemon has backgrounded itself may appear on standard error.

The maximum number of characters in the message portion of a log entry is 4096.

### 9.4.3 Log Levels

PBS allows specification of the types of events that are logged for each daemon. Each type of log event has a different log level. All daemons use the same log level for the same type of event.

The following table lists the log level for each type of event.

**Table 9-2: PBS Events and Log Levels**

| Name               | Decimal | Hex    | Event Description                                                  |
|--------------------|---------|--------|--------------------------------------------------------------------|
| PBSEVENT_ERROR     | 1       | 0x0001 | Internal PBS errors                                                |
| PBSEVENT_SYSTEM    | 2       | 0x0002 | System (OS) errors, such as malloc failure                         |
| PBSEVENT_ADMIN     | 4       | 0x0004 | Administrator-controlled events, such as changing queue attributes |
| PBSEVENT_JOB       | 8       | 0x0008 | Job related events, e.g. submitted, ran, deleted                   |
| PBSEVENT_JOB_USAGE | 16      | 0x0010 | Job resource usage                                                 |
| PBSEVENT_SECURITY  | 32      | 0x0020 | Security related events                                            |
| PBSEVENT_SCHED     | 64      | 0x0040 | When the scheduler was called and why                              |
| PBSEVENT_DEBUG     | 128     | 0x0080 | Common debug messages                                              |
| PBSEVENT_DEBUG2    | 256     | 0x0100 | Debug event class 2                                                |
| PBSEVENT_RESV      | 512     | 0x0200 | Reservation-related messages                                       |
| PBSEVENT_DEBUG3    | 1024    | 0x0400 | Debug event class 3. Debug messages rarer than event class 2.      |
| PBSEVENT_DEBUG4    | 2048    | 0x0800 | Debug event class 4. Limit-related messages.                       |

### 9.4.3.1 Specifying Log Levels

Each daemon uses an integer representation of a bit string to specify its log levels. The bit string can be decimal (or hexadecimal, for the MoM). Each daemon's log levels are specified in a bit string that includes the events to be logged. You can specify each multischd's log levels individually.

For example, if you want the server to log all events except those at event classes 512, 1024, and 2048 (hex 0x0200, 0x0400, and 0x0800), you would use a log level of 511. This is  $256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$ . If you want to log events at event classes 1, 2, and 16, you would set the log level to 19.

The following table shows the log event parameter for each daemon:

**Table 9-3: Daemon Log Event Specification Parameters**

| PBS           | Parameter/Attribute                                     | Reference                                                                                                | How to Make Parameter Take Effect               |
|---------------|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| Server        | log_events attribute                                    | <a href="#">“Server Attributes” on page 281 of the PBS Professional Reference Guide</a>                  | Takes effect immediately with <code>qmgr</code> |
| MoM           | \$logevent parameter                                    | <a href="#">“Contents of MoM Configuration File” on page 244 of the PBS Professional Reference Guide</a> | Requires <code>SIGHUP</code> to MoM             |
| Scheduler     | log_events attribute                                    | <a href="#">“Configuration Parameters” on page 252 of the PBS Professional Reference Guide</a>           | Takes effect immediately with <code>qmgr</code> |
| communication | PBS_COMM_LOG_EVENT S parameter in <code>pbs.conf</code> | <a href="#">“Daemon Log Mask” on page 48 in the PBS Professional Installation &amp; Upgrade Guide</a>    | Restart the communication daemon                |

When reading the PBS event logfiles, you may see messages of the form "Type 19 request received from PBS\_Server...". These "type codes" correspond to different PBS batch requests. See [“Request Codes” on page 393 of the PBS Professional Reference Guide](#).

### 9.4.3.1.i Specifying Server Log Events

You can specify the server's log events by setting the server's `log_events` attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the `qmgr` command:

```
Qmgr: set server log_events = <value>
```

The new value takes effect immediately.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to `1025` (1024 + 1, or 0x0401). To include all events, set the value to `4095` or `-1`. The default value for this attribute is 511. It can be set by Operators and Managers only. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

You can set the server's log level when you start the server using `pbs_server -e <log level>`. Note that you can specify a hexadecimal value this way, but not via the server's `log_events` attribute. When you use the `-e <log level>` option to `pbs_server`, that sets the server's `log_events` attribute to the corresponding integer value.

### 9.4.3.1.ii Specifying MoM Log Events

Each MoM's log events are specified in the `$logevent` parameter in that MoM's configuration file `PBS_HOME/mom_priv/config`. The parameter is an integer representation of a bit string, where the integer includes all events to be logged. For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to `1025` (1024 + 1, or 0x0401). To set the value, add the `$logevent` line in `PBS_HOME/mom_priv/config`, then HUP the MoM. To include all events, set the value to `4095` (0xffff). The default value used by MoM is 975 (0x03cf). This parameter can be set by root only. See [“Contents of MoM Configuration File” on page 244 of the PBS Professional Reference Guide](#).

### 9.4.3.1.iii Specifying Scheduler Log Events

You can specify log events for the scheduler and for each multisched by setting each scheduler's `log_events` attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the `qmgr` command:

```
Qmgr: set sched <scheduler name> log_events = <value>
```

The new value takes effect immediately.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to `1025` (1024 + 1, or 0x0401). To include all events, set the value to `4095` or `-1`. The default value for this attribute is 767. It can be set by Operators and Managers only. See [“Scheduler Attributes” on page 299 of the PBS Professional Reference Guide](#).

### 9.4.3.1.iv Specifying Communication Daemon Log Events

The communication daemon's log events are specified in the `PBS_COMM_LOG_EVENTS` parameter in `/etc/pbs.conf`. This parameter is an integer representation of a bit string, where the integer includes all events to be logged. HUP the daemon after you set the parameter.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to `1025` (1024 + 1, or 0x0401). To include all events, set the value to `2047` (or `-1`). The default value for this attribute is 511 (0x1ff). See [“Logging and Errors” on page 55 in the PBS Professional Installation & Upgrade Guide](#) and [“Contents of Configuration File” on page 422](#).



## 9.4.4 Event Logfile Format and Contents

### 9.4.4.1 Event Logfile Format

Each component event logfile is a text file with each entry terminated by a new line. The format of an entry is:

`<logfile date and time>;<event code>;<server name>;<object type>;<object name>;<message>`

- The *logfile date and time* field is a date and time stamp in the format:

`mm/dd/yyyy hh:mm:ss[.xxxxxx]`

If microsecond logging is enabled, microseconds are logged using the `.xxxxxx` portion. Microseconds may be preceded by zeroes. Microsecond logging is controlled per host via the [PBS LOG HIGHRES TIMESTAMP](#) configuration parameter or environment variable.

- The *event code* is a bitmask for the type of event which triggered the event logging. It corresponds to the bit position, 0 to n, of each log event in the event mask of the PBS component writing the event record. See [section 9.4.1, “PBS Events”, on page 429](#) for a description of the event mask.
- The *server name* is the name of the server which logged the message. This is recorded in case a site wishes to merge and sort the various logs in a single file.
- The *object type* is the type of object which the message is about. All messages are associated with an *object type*. The following lists each possible *object type*:

**Table 9-4: List of Event Logfile Object Types**

| Object Type | Usage                 |
|-------------|-----------------------|
| Svr         | for server            |
| Que         | for queue             |
| Job         | for job               |
| Req         | for request           |
| Fil         | for file              |
| Act         | for accounting string |
| Node        | for vnode or host     |
| Resv        | for reservation       |
| Sched       | for scheduler         |

- The *object name* is the name of the specific object.
- The *message* field is the text of the log message.

### 9.4.4.2 Scheduler Commands

These commands tell a scheduler why a scheduling cycle is being started. These commands appear in the server's log-file. Each has a decimal value, shown below. The following table shows commands from the server to a scheduler.

**Table 9-5: Commands from Server to Scheduler**

| Value | Event Description                                          |
|-------|------------------------------------------------------------|
| 1     | New job enqueued                                           |
| 2     | Job terminated                                             |
| 3     | Scheduler time interval reached                            |
| 4     | Cycle again after scheduling one job                       |
| 5     | Scheduling command from operator or manager                |
| 7     | Configure                                                  |
| 8     | Quit ( <code>qterm -s</code> )                             |
| 9     | Ruleset changed                                            |
| 10    | Schedule first                                             |
| 11    | A reservation's start time has been reached                |
| 12    | Schedule a job ( <code>qrun</code> command has been given) |
| 13    | Stopped queue is started                                   |
| 14    | Job moved into local queue (queue at this server)          |
| 15    | <code>eligible_time_enable</code> is turned on             |
| 16    | PBS attempting to reconfirm degraded reservation           |

### 9.4.5 Logging Job Usage

PBS can log per-vnode cputime usage. The primary execution host MoM logs cputime in the format "hh:mm:ss" for each vnode of a multi-vnode job. The log level of these messages is 0x0100.

Under Linux, to append job usage to standard output for an interactive job, use a shell script for the epilogue which contains the following:

```
#!/bin/sh
tracejob -sl $1 | grep 'cput'
```

This behavior is not available under Windows.

### 9.4.6 Managing Log Files

#### 9.4.6.1 Disk Space for Log Files

It is important not to run out of disk space for logging. You should periodically check the available disk space, and check the size of the log files PBS is writing, so that you know how fast you are using up disk space. Make sure that you always have more than enough disk space available for log files.

### 9.4.6.2 Dividing Up Log Files

You may wish to divide a day's logging up into more than one file. You may want to create a logfile that contains only the entries of interest. You can specify a file for a daemon's event log. See [section 9.4.6.3, “Specifying Log File Path”, on page 434](#). The next sections describe how to break up your log files.

#### 9.4.6.2.i Dividing Log Files on Linux

On Linux systems, all daemons close and reopen the same named log file when they are sent a SIGHUP. The process identifier (PID) of each daemon is available in its lock file in its home directory. You can move the current log file to a new name and send SIGHUP to restart the file using the following commands:

```
cd $PBS_HOME/<daemon>_logs
mv <current log file> <archived log file>
kill -HUP `cat ../<daemon>_priv/<daemon>.lock`
```

#### 9.4.6.2.ii Dividing Log Files on Windows

On Windows systems, you can rotate the event log files by stopping the service for which you want to rotate the logfile, moving the file, and then restarting that service. For example:

```
cd "%PBS_HOME%\mom_logs"
net stop pbs_mom
move <current log file> <archived log file>
net start pbs_mom
```

### 9.4.6.3 Specifying Log File Path

You may wish to specify an event logfile path that is different from the default path. Each daemon has an option to specify a different path for the daemon's event logfile. This option is the `-L logfile` option, and it is the same for all daemons. For example, to start the scheduler so that it logs events in `/scratch/my_sched_log`:

```
pbs_sched -L /scratch/my_sched_log
```

See the `pbs_server(8B)`, `pbs_sched(8B)`, and `pbs_mom(8B)` manual pages.

## 9.4.7 Extracting Logged Information

You can use the `tracejob` command to extract information from log files, such as why a job is not running or when a job was queued. The `tracejob` command can read both event logs and accounting logs. See the `tracejob(8B)` manual page.

## 9.4.8 Using the Linux `syslog` Facility

Each PBS component logs various event classes of information about events in its own log file. While having the advantage of a concise location for the information from each component, the disadvantage is that in a complex, the logged information is scattered across each execution host. The Linux `syslog` facility can be useful.

If your site uses the `syslog` subsystem, PBS may be configured to make full use of it. The following entries in `pbs.conf` control the use of `syslog` by the PBS components:

**Table 9-6: Entries in `pbs.conf` for Using Syslog**

| Entry                         | Description                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>PBS_LOCALLOG=x</code>   | Enables logging to local PBS log files. Only possible when logging via <code>syslog</code> feature is enabled.<br><br>0 = no local logging<br>1 = local logging enabled                                                                                                                                                                              |
| <code>PBS_SYSLOG=x</code>     | Controls the use of syslog and syslog facility under which the entries are logged. If <code>x</code> is:<br><br>0 - no syslogging<br>1 - logged via <code>LOG_DAEMON</code> facility<br>2 - logged via <code>LOG_LOCAL0</code> facility<br>3 - logged via <code>LOG_LOCAL1</code> facility<br>...<br>9 - logged via <code>LOG_LOCAL7</code> facility |
| <code>PBS_SYSLOGSEVR=y</code> | Controls the severity level of messages that are logged; see <code>/usr/include/sys/syslog.h</code> . If <code>y</code> is:<br><br>0 - only <code>LOG_EMERG</code> messages are logged<br>1 - messages up to <code>LOG_ALERT</code> are logged<br>...<br>7 - messages up to <code>LOG_DEBUG</code> are logged                                        |

### 9.4.8.1 Caveats

- `PBS_SYSLOGSEVR` is used in addition to PBS's `log_events` mask which controls the class of events (job, vnode, ...) that are logged.
- If you use `syslog`, you cannot have daemons log events at microsecond resolution.

## 9.5 Managing Machines

### 9.5.1 Offlining Hosts and Vnodes

For using hooks to offline vnodes, see ["Offlining Bad Vnodes" on page 72 in the PBS Professional Hooks Guide](#).

To offline an entire host, use the `pbsnodes` command. Use the name of the parent vnode, which is usually the name of the host:

```
pbsnodes -o <name of parent vnode>
```

All vnodes on this host are offlined.

To offline a single vnode, use the `qmgr` command, with the name of the vnode:

```
qmgr -c "set node foo[3] state=offline"
```

### 9.5.1.1 Caveats of Offlining

If you set a vnode with no running jobs *offline*, the server will not attempt to communicate with the vnode. Therefore, the server will not notice that the vnode is up until you clear the *offline* state. For example, a vnode that is both *down* and *offline* will not be marked up by the server until you clear the *offline* state.

## 9.5.2 Performing Maintenance on Powered-up Vnodes

### 9.5.2.1 Reserving Vnodes for Maintenance

You can create maintenance reservations using `pbs_rsub --hosts <host list>`. Maintenance reservations are designed to make the specified hosts available for the specified amount of time, regardless of what else is happening:

- You can create a maintenance reservation that includes or is made up of vnodes that are down or offline.
- Maintenance reservations ignore the value of a vnode's `resv_enable` attribute.
- PBS immediately confirms any maintenance reservation.
- Maintenance reservations take precedence over other reservations; if you create a maintenance reservation that overlaps an advance or standing job reservation, the overlapping vnodes become unavailable to the job reservation, and the job reservation is in conflict with the maintenance reservation. PBS looks for replacement vnodes; see ["Reservation Fault Tolerance" on page 401 in the PBS Professional Administrator's Guide](#).

PBS will not start any new jobs on vnodes overlapping or in a maintenance reservation. However, jobs that were already running on overlapping vnodes continue to run; you can let them run or requeue them.

You cannot specify place or select for a maintenance reservation; these are created by PBS:

- PBS creates the reservation's placement specification so that hosts are assigned exclusively to the reservation. The placement specification is always the following:

`-lplace=exclhost`

- PBS sets the reservation's `resv_nodes` attribute value so that all CPUs on the reserved hosts are assigned to the maintenance reservation. The select specification is always the following:

`-lselect=host=<host1>:ncpus=<number of CPUs at host1>+host=<host2>:ncpus=<number of CPUs at host2>+...`

Maintenance reservations are prefixed with *M*. A maintenance reservation ID has the format:

`M<sequence number>.<server name>`

You cannot create a recurring maintenance reservation.

Creating a maintenance reservation does not trigger a scheduling cycle.

You must have manager or operator privilege to create a maintenance reservation.

### 9.5.2.2 Putting Vnodes into Maintenance State

You may want to perform maintenance on a vnode while it is powered up, but you don't want job processes running on it. You can suspend a job on a vnode and put the vnode into a *maintenance* state, where the scheduler will not start any new jobs on the vnode, using `qsig -s admin-suspend <job ID>`. You must suspend each job on the vnode; if you suspend only one, the rest will keep running. When you *admin-suspend* a job, all of the job's vnodes are put into the *maintenance* state, the job goes into the *S* state, and the job's processes are suspended.

Once the maintenance is finished, you can resume the *admin-suspended* jobs using `qsig -s admin-resume <job ID>`. The *admin-resume* signal directly resumes the job, without waiting for the scheduler. Once all of the vnode's jobs are *admin-resumed*, the vnode leaves the *maintenance* state.

You can see the list of jobs that were running on a vnode then *admin-suspended* in the *maintenance\_jobs* vnode attribute. This attribute is a list of job IDs, and is readable only by managers.

### 9.5.2.2.i Resource Release on Suspension

When you *admin-suspend* a job, resources are released according to how you have configured the *restrict\_res\_to\_release\_on\_suspend* server attribute; see [section 5.9.6.2, “Job Suspension and Resource Usage”, on page 247](#). However, no new jobs will run while the job is suspended.

### 9.5.2.2.ii Caveats for admin-suspend and admin-resume

- We recommend that before you *admin-suspend* any job, you disable scheduling and wait for the current scheduling cycle to finish. The scheduler queries vnode state only at the beginning of the scheduling cycle. If a vnode goes into the *maintenance* state after the start of the cycle, the scheduler could still schedule jobs onto that vnode.
- The *suspend* and *resume* signals are not interchangeable with the *admin-suspend* and *admin-resume* signals. For example, if a job is suspended via normal the *suspend* signal (`qsig -s suspend <job ID>`), it cannot be resumed with the *admin-resume* signal.

Similarly, if a job is suspended with the *admin-suspend* signal, it cannot be resumed with the *resume* signal. Either request will be rejected with the following message:

"Job can not be resumed with the requested resume signal"

- If there are multiple jobs on a vnode, we recommend using either *suspend* and *admin-suspend*, but not both. If you have a *suspended* job on a vnode that was in the *maintenance* state but is no longer, the scheduler could run jobs on the resources owned by the *suspended* job.
- If you want to perform maintenance on a vnode that has no jobs running on it, we recommend putting the vnode into the *offline* state before performing the maintenance.
- Any reservations on vnodes in the *maintenance* state are marked *degraded*. PBS searches for alternate vnodes for those reservations.
- Any vnode which had only *admin-suspended* subjobs will stay in the *maintenance* state after a server restart.

## 9.5.3 Changing Hostnames or IP Addresses

- Do not change a hostname without updating the corresponding Version 2 configuration file.
- Do not change the IP address or hostname of a machine in the complex while PBS is running. Stop PBS (server, scheduler, and MoMs), change the IP address, and restart PBS.

To change a hostname or IP address:

1. Make sure no jobs are running
2. Stop all PBS daemons
3. Make a backup of `PBS_HOME`
4. Change the hostname or IP address
5. If you are using the IP address as a vnode name, update `PBS_MOM_NODE_NAME` in `pbs.conf` on the execution host to the new IP address.
6. Restart all PBS daemons
7. If a host has a corresponding Version 2 configuration file, make sure that it is consistent with the new hostname
8. If you are running `nsd`, restart `nsd` on all hosts

### 9.5.4 Discovering Last Reboot Time of Server

Under Linux, you can find the timestamp of the most recent time PBS started up in `/var/tmp/pbs_boot_check`.

The permission of this file is set to 0644; only the PBS init script should modify this file. Do not modify this file. If you do so, you violate the configuration requirements of PBS.

This file is not available under Windows.

### 9.5.5 Changing Network Configuration

If you change any network configuration, restart the PBS daemons.

### 9.5.6 Replacing or Reimaging Nodes

When `PBS_HOME` is removed on a node by reimaging, etc., make sure that the server knows that there are no legitimate jobs on the node. Send each job `qsig -s SIGNULL` after the node is up again, which causes the server to contact the MoM and discover that any jobs are gone as far as MoM is concerned. The server then requeues and reruns any of MoM's gone jobs. Otherwise zombie jobs will ensure that no new jobs are scheduled to the node even after it's been reimaged.

MoM depends on its `PBS_HOME` to know which jobs are gone. When a node goes down on PBS complexes with either diskless nodes or nodes with integrated disk drives, sometimes the cluster manager will reimage the node before the complex reintegrates the node. In that case, `PBS_HOME` is gone, so MoM no longer knows about any jobs she was managing. The server will never get any updates or obits for those jobs, so they'll stay in state *R*.

If the reimaging process is longer than `node_fail_requeue`, the server will requeue the jobs, but your complex may use `node_fail_requeue` set to 0 for very good reasons, for example if there are Cray or HPE NUMA machines in the complex.

### 9.5.7 Restricting User Access to Execution Hosts

PBS provides a facility to prevent users who are not running PBS jobs from using machines controlled by PBS. You can turn this feature on by using the `$restrict_user` MoM directive. This directive can be fine-tuned by using the `$restrict_user_exceptions` and `$restrict_user_maxsysid` MoM directives. This feature can be set up host by host.

- A user requesting exclusive access to a set of hosts (via `place=excl`) can be guaranteed that no other user will be able to use the hosts assigned to his job, and PBS will not assign any unallocated resources on the vnode to another job.
- A user requesting non-exclusive access to a set of hosts can be guaranteed that no non-PBS users are allowed access to the hosts.
- A privileged user can be allowed access to the complex such that they can log into a host without having a job active.
- An abusive user can be denied access to the complex hosts.

The administrator can find out when users try to access hosts without going through PBS. The administrator can ensure that application performance is consistent on a complex controlled by PBS. PBS will also be able to clean up any job processes remaining after a job finishes running.



For a vnode with access restriction turned on:

- Any user not running a job who logs in or otherwise starts a process on that vnode will have his processes terminated.
- A user who has logged into a vnode where he owns a job will have his login terminated when the job is finished.
- When MoM detects that a user that is not exempt from access restriction is using the system, that user's processes are killed and a log message is output:  

```
01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
```

You can set up a list of users who are exempted from the restriction via the `$restrict_user_exceptions` directive. This list can contain up to 10 usernames.

Example 9-1: Turn access restriction on for a given node:

```
$restrict_user True
```

Example 9-2: Limit the users affected to those with a user ID greater than 500:

```
$restrict_user_maxsysid 500
```

Example 9-3: Exempt specific users from the restriction:

```
$restrict_user_exceptions userA, userB, userC
```

Note that a user who has a job running on a particular host will be able to log into that host.

### 9.5.7.1 Windows Restriction

The user access restriction feature is not supported on Windows.

## 9.6 Managing the Data Service

### 9.6.1 PBS Monitors Data Service

PBS monitors its connection to the data service. If the connection is broken (for example, because the data service is down), PBS tries to reestablish the connection. If necessary, PBS restarts the data service.

If failover is configured, and PBS cannot reestablish a connection, PBS quits.

If failover is not configured, PBS attempts to reestablish the connection until it succeeds.

When the server is stopped, it stops the data service.

### 9.6.2 Data Service Accounts

On Linux, PBS uses a PBS data service management account and an internal data service account. They are described here: [“Create PBS Data Service Management Account” on page 24 in the PBS Professional Installation & Upgrade Guide.](#)

### 9.6.3 Data Service Account Password

The default password for the internal data service account is a random password that is generated at installation, and which is known only to the PBS server.

### 9.6.3.1 Setting Data Service Account Name and Password

Changing the password is necessary only if you want to manually log into the database to check data or change something. Otherwise it is not necessary.

Use the `pbs_ds_password` command to change the password of the data service internal user account (not the PBS data service management account).

You can change the user account and/or password for the data service account using the `pbs_ds_password` command. Use this command if you need to change the user account or update the password for the data service account. You must be root or administrator to run the `pbs_ds_password` command. See [“pbs\\_ds\\_password” on page 62 of the PBS Professional Reference Guide](#).

To change the data service account name:

```
pbs_ds_password -C <new user account>
```

To change the data service account password:

```
pbs_ds_password
```

### 9.6.3.2 Caveats

- When you specify a new name for the data service account, there must already be a data service management account with that name
- The account name cannot be changed while the data service is running.
- Do not delete `PBS_HOME/server_priv/db_password`. Doing so will prevent the `pbs_ds_password` command from being able to function.
- Do not change the data service password using any method other than the `pbs_ds_password` command.
- If you change the data service account after installing PBS, and then you want to upgrade PBS, you must change it back in order to perform the upgrade. After the upgrade, you can change it again. This is covered in the upgrading instructions.
- If you type in a password, make sure it does not contain restricted characters. The `pbs_ds_password` command generates passwords containing the following characters:

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$$%^&*()_+
```

When creating a password manually, do not use `\` (backslash) or `'` (backquote). This can prevent certain commands such as `pbs_server`, `pbs_ds_password`, and `print job` from functioning properly, as they rely on connecting to the database. The format is also described in ["PBS Password" on page 357](#).

### 9.6.4 Starting and Stopping the Data Service

PBS automatically starts and stops the data service. However, you can start, stop, or check the status of the PBS data service using the `pbs_dataservice` command. See [“pbs\\_dataservice” on page 61 of the PBS Professional Reference Guide](#).

To start the data service:

```
pbs_dataservice start
```

To stop the data service:

```
pbs_dataservice stop
```

To get the status of the data service:

```
pbs_dataservice status
```

---

### 9.6.4.1 Caveats for Starting and Stopping Data Service

- **Do not** start or stop the data service using anything except the `pbs_dataservice` command. Start or stop the data service using only the `pbs_dataservice` command.
- The data service cannot be stopped while the PBS server is running.

## 9.6.5 Changing Data Service Port

You can change the port that the data service listens on by changing the setting of the `PBS_DATA_SERVICE_PORT` entry in `pbs.conf`.

### 9.6.5.1 Caveats

- The PBS daemons must not be running when the port is changed.
- The data service must not be running when the port is changed.

## 9.6.6 File Ownership

The files under `PBS_HOME/datastore` are owned by the data service user account.

# 9.7 Setting File Transfer Mechanism

MoM does the work of transferring files, using the mechanisms you specify. MoM transfers files when she stages them in or out for a job, and when she delivers output and error files.

MoM always tries to determine whether the source and destination for a file transfer are both local. If MoM knows that she is performing a local file transfer, she uses her local copy mechanism. Otherwise, she uses her remote copy mechanism. You can specify MoM's copy mechanisms for local copying, and for copying to or from remote hosts.

PBS does not impose limitations on the size of files being transferred. Any limitations are caused by the commands themselves.

## 9.7.1 Letting MoM Know Whether Transfer is Local or Remote

To tell MoM which directories can be treated as local, specify the mappings between local and mounted directories in MoM's `$usecp` configuration parameter.

### 9.7.1.1 Configuring the `$usecp` MoM Parameter

The `$usecp` configuration parameter tells MoM where to look for files in a shared file system, so that she can use the local copy mechanism for these files. This is useful when you have common mount points across execution hosts.

Format:

`$usecp <hostname>:<source directory> <destination directory>`

You can use a wildcard ("`*`") as the first element only, to replace *hostname*.

MoM uses her local copy mechanism to transfer files when staging or delivering output, under the following circumstances:

- The destination is a network mounted file system
- The source and destination are both on the local host
- The *source directory* can be replaced with the *destination directory* on *hostname*

You can map multiple directories. Use one line per mapping.

You must HUP MoM after making this change.

### 9.7.1.1.i Linux and \$usecp

Format:

`$usecp <hostname>:<source directory> <destination directory>`

Use trailing slashes on both the source and destination directories.

Example 9-4: Configuring \$usecp on Linux:

```
$usecp */home/ /home/
$usecp *.example.com:/home/ /home/
$usecp */home/user/ /home/user/
$usecp */data/ /data/
$usecp HostA:/users/work/myproj/ /sharedwork/proj_results/
```

### 9.7.1.1.ii Windows and \$usecp

Format:

`$usecp <host name>:<drive name>:<directory> <drive name>:<directory>`

When a network location is mapped to a local drive, you can cover all host names and case-sensitivity using entries similar to these:

```
$usecp *:Q: Q:
$usecp *:q: q:
```

Using this mapping, when MoM sees files with this format:

`<hostname>:Q:<file path>`

or

`<hostname>:q:<file path>`

she passes them to the copy command with this format:

`Q:<file path>`

or

`q:<file path>`

Example 9-5: Mapping locations with different directory names:

```
$usecp HostB:C:/xxxxx C:/yyyyy
```

## 9.7.2 Specifying Local File Transfer Mechanism

You can specify the local copy command in the `PBS_CP` parameter in `pbs.conf`.

On Linux the local copy mechanism defaults to `/bin/cp`, and on Windows it defaults to `xcopy`.

### 9.7.2.0.i How MoM Calls Local Copy Command

MoM calls the command this way on Linux:

```
$PBS_CP -rp <path to source> <path to destination>
```

You cannot specify options inside the PBS\_CP entry.

## 9.7.3 Specifying Remote File Transfer Mechanism

MoM can use either `scp` or `rcp` for remote copying. In `/etc/pbs.conf`, you specify which `scp` in the `PBS_SCP` parameter, and you specify which `rcp` in the `PBS_RCP` parameter. You can specify both `PBS_SCP` and `PBS_RCP`.

### 9.7.3.1 How MoM Chooses Remote File Copy Mechanism

- If a command is specified in `PBS_SCP`, MoM uses the command specified in `PBS_SCP`.
- If a command is specified in `PBS_RCP`, and `PBS_SCP` is not defined, MoM uses the command specified in `PBS_RCP`.
- If no `pbs.conf` parameters are defined, MoM uses `pbs_rcp`.

### 9.7.3.2 Configuring MoM to use `scp` or `PBS_SCP` Parameter

The `PBS_SCP` `pbs.conf` parameter is the absolute path to a command or script used for remote transfer.

At installation PBS looks for `scp` in the system `PATH`, and if it finds `scp` in the system `PATH`, PBS sets `PBS_SCP` in `pbs.conf` to that `scp`.

If `PBS_SCP` is not set in `pbs.conf` on Linux, or you want MoM to use `scp` for remote copying on Windows, follow the steps below.

1. Make sure that `scp` and `ssh` are installed on each host involved in file transfer
2. MoM calls her `scp` mechanism with the `-B` option, which requires passwordless authentication. If you use plain `scp` without a wrapper script, make sure that passwordless authentication is set up on all machines involved in file transfer. See [section 9.7.10.1, “Enabling Passwordless Authentication”, on page 448](#)
3. To use `scp` on Windows, set up passwordless authentication on all machines involved in file transfer. See [section 9.7.10.1, “Enabling Passwordless Authentication”, on page 448](#)
4. Set `PBS_SCP` to the absolute path to `scp` or your wrapper script
5. If MoM is already running, restart MoM

### 9.7.3.2.i How MoM Calls `scp` Command

MoM calls the command this way on Linux:

```
$PBS_SCP -Brvp <path to source> <username>@<destination>.<host>:<path to destination>
```

You cannot specify options inside the `PBS_SCP` entry.

### 9.7.3.3 Configuring MoM to use `rcp`, `pbs_rcp` or `PBS_RCP` Entry

The `PBS_RCP` `pbs.conf` parameter is the absolute path to a command or script used for remote transfer.

PBS ships with a version of `rcp` called `pbs_rcp`. On Windows, PBS uses this version by default. The `pbs_rcp` command should be as fast as other implementations of `rcp`.

If you want MoM to use a different `rcp`, or a wrapper script for `rcp`:

1. Make sure that `rcp` and `rsh` are installed on each host involved in file transfer
2. Set `PBS_RCP` in `/etc/pbs.conf`, to the absolute path to the desired `rcp` or wrapper script
3. If `PBS_SCP` is defined in `/etc/pbs.conf`, comment it out
4. If MoM is already running, restart MoM

### 9.7.3.3.i How MoM Calls `rcp` Command

MoM calls the command this way on Linux:

```
$PBS_RCP -rp <path to source> <username>@<destination>.<host>:<path to destination>
```

You cannot specify options inside the `PBS_RCP` entry.

## 9.7.4 Options Passed to File Transfer Commands

### 9.7.4.1 Options Passed on Linux

MoM automatically uses these options on Linux:

**Table 9-7: File Transfer Mechanism Options on Linux**

| Distance | Mechanism                                                                  | Options            |
|----------|----------------------------------------------------------------------------|--------------------|
| Remote   | <code>PBS_RCP</code> parameter, <code>rcp</code> , or <code>pbs_rcp</code> | <code>-rp</code>   |
| Remote   | <code>PBS_SCP</code> parameter                                             | <code>-Brvp</code> |
| Local    | <code>PBS_CP</code> parameter, or <code>/bin/cp</code>                     | <code>-rp</code>   |

### 9.7.4.2 Options Passed on Windows

MoM automatically uses these options on Windows:

**Table 9-8: File Transfer Mechanism Options on Windows**

| Distance | Mechanism                                                                  | Options               |
|----------|----------------------------------------------------------------------------|-----------------------|
| Remote   | <code>PBS_RCP</code> parameter, <code>rcp</code> , or <code>pbs_rcp</code> | <code>-E -r</code>    |
| Remote   | <code>PBS_SCP</code> parameter                                             | <code>-Brv</code>     |
| Local    | <code>PBS_CP</code> parameter, or <code>xcopy</code>                       | <code>/e/i/q/y</code> |

## 9.7.5 Using Custom File Transfer Mechanism

You can also tell MoM to use any script or command for file transfer, such as `rsync`, `gsiftp`, etc.

### 9.7.5.1 Using Custom Local File Transfer Mechanism

If you want MoM to use different flags to `cp`, or a different command, or your own script, for local file transfer:

1. If needed, write a script that does what you need
2. Specify the path to the command or script in `PBS_CP` in `pbs.conf`
3. If the MoM is already running, restart the MoM

When MoM calls `PBS_CP`, she calls it with the `-rp` (Linux) or `-e/i/q/y` (Windows) flags. This means that when you are writing a script, the arguments being passed to the script are:

```
$1 -rp or -e/i/q/y
$2 path to source
$3 path to destination
```

You choose which arguments the script passes to the command inside the script. If you are using a different command, make sure that you pass the correct flags to it.

### 9.7.5.2 Using Custom Remote File Transfer Mechanism

If you want MoM to use different flags to `rcp` or `scp`, or a different command, or your own script, for remote file transfer:

1. If needed, write a script that does what you need
2. Specify the path to the command or script in `PBS_SCP` in `pbs.conf`
3. If the MoM is already running, restart the MoM.

When MoM calls `PBS_SCP`, she calls it with the `-Brvp` (Linux) or `-Brv` (Windows) flags. This means that when you are writing a script, the arguments being passed to the script are:

```
$1-Brvp or -Brv
$2path to source
$3path to destination
```

You choose which arguments the script passes to the command inside the script. If you are using a different command, make sure that you pass the correct flags to it.

Example 9-6: Pass desired options to `scp` by writing a wrapper script for `scp` that contains the desired options, and pointing `PBS_SCP` to the wrapper script. In this case, we don't use the default `-Brvp`, which is passed to the script as `$1`. The script does not pass `$1` to `scp`; instead, it specifies `-Br`. We do pass in the source and destination as `$2` and `$3`.

```
In pbs.conf:
PBS_SCP=/usr/bin/scp_pbs

In /usr/bin/scp_pbs:
#!/bin/sh
/usr/bin/scp -Br $2 $3
```

Example 9-7: Use `rsync` by writing a wrapper script that passes all arguments except for the first (`-Brvp`) to `rsync`, and pointing `PBS_SCP` to the wrapper script. In this case, the script passes all but the first argument to `rsync` as `$*`. We get rid of the first argument using the `shift` command.

```
In pbs.conf:
PBS_SCP=/usr/bin/rsync_pbs
```



```
In /usr/bin/rsync_pbs:
#!/bin/sh
shift
/usr/bin/rsync -avz -e ssh $*
```

Remember that for remote copying, MoM tries the `PBS_SCP` entry in `pbs.conf` first. If you configure both `PBS_RCP` and `PBS_SCP` with scripts or commands, put the script or command that you want MoM to try first in `PBS_SCP`.

## 9.7.6 When Multiple Attempts Are Required

If necessary, MoM tries to transfer a file multiple times, with an increasing delay between each attempt:

- If MoM is using her local copy mechanism, she tries it up to four times
- If MoM is using the entry in `PBS_SCP`:
  - She first tries this, and if it fails, she tries `rcp`, `pbs_rcp`, or the entry in `PBS_RCP` if it is configured
  - She repeats this sequence four times
- If MoM is using `rcp`, `pbs_rcp`, or the entry in `PBS_RCP`, she tries it up to four times

## 9.7.7 Allowing Direct Write of Standard Output and Error to /dev/null

Standard output and standard error are normally written to a location such as `/var/spool`, then copied to their final location. To avoid creating these files at all, and to avoid copying them, you need two things:

In MoM's version 1 configuration file, add this:

```
$usecp $<MoM hostname>:/dev/null /dev/null
```

Job submitters can use direct write to send them to `/dev/null`:

```
qsub -koed -o /dev/null -e /dev/null
```

## 9.7.8 Troubleshooting File Transfer

### 9.7.8.1 Problems with rcp

When using `rcp`, the copy of output or staged files can fail for the following reasons:

- The user lacks authorization to access the specified system
- Under Linux, if the user's `.cshrc` prints any characters to standard output, e.g. contains an `echo` command, the copy will fail

You may encounter a strange hang in stageout, with jobs stuck in the *E* state for a long time. This can happen because `rcp` may be trying to connect to a port that's already in use. If your standard copy mechanism is `scp`, and you don't want to let PBS fall back on `pbs_rcp`, do one of the following:

- You can move `pbs_rcp`
- If you specify `PBS_SCP`, set `PBS_RCP` to `/bin/false` in `pbs.conf`
- If you are using CM/PAS and specify `PBS_SCP` in `/etc/conf`, put the `PBS_SCP` line after the `PBS_RCP` line

### 9.7.8.2 Problems with Directory Access

Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist
- A directory in the specified destination path is not searchable by the user
- The target directory is not writable by the user

## 9.7.9 Advice on Improving File Transfer Performance

### 9.7.9.1 Avoiding Server Host Overload

Avoid staging files from the server host, unless you can isolate the daemons from the effects of CPU and memory usage by `scp/ssh`, by using a mechanism such as `cpusets`. Consider the impact from a large job array that causes many files to be staged from the server host. Instead, use a shared filesystem. See [section 9.7.9.2, “Avoiding Remote Transfers in Large Complexes”](#), on page 447.

### 9.7.9.2 Avoiding Remote Transfers in Large Complexes

If you are running a very large HPC complex, consider using MoM's `$usecp` directive to avoid `rcp` and `scp` transfers. Instead, have your users place input files on a shared filesystem before submitting jobs, write their output to the shared filesystem, and keep as much as possible out of `stdout` and `stderr`.

### 9.7.9.3 Improving Performance for ssh

If network bandwidth is a limiting factor, you can use compression to improve performance. However, if CPU usage and/or memory are limiting factors, do not use compression, because compression also requires CPU and memory.

You can use compression ciphers that minimize the CPU and memory load required, for example `arcfour` or `blowfish-cbc`:

```
ciphers arcfour,blowfish-cbc
```

### 9.7.9.4 Improving Performance when Staging Similar Files

If you are staging in many similar files, for example, for job arrays, you can use `rsync` in a wrapper script. Follow the instructions in [section 9.7.5.1, “Using Custom Local File Transfer Mechanism”](#), on page 445.

### 9.7.9.5 Avoiding Limits on ssh Connections

To prevent `scp` requests being denied when using `ssh`, you can set higher limits on incoming `ssh` connections. By default `ssh` is configured to treat more than 10 incoming connections (plus 10 in the authentication phase) as a denial-of-service attack, even on machines that could service many more requests.

Set higher limits in `/etc/ssh/sshd_config` for servers that are meant to service a lot of incoming openSSH sessions, but only on machines that have enough CPU and memory to service all of the requests.

See the `MaxSessions` and `MaxStartups` parameters in the man page for `sshd_config`. You can make these at least as large as the number of hosts in the cluster plus 10, assuming that any MoM only has one `scp` session open at any one time.

### 9.7.9.6 Alternatives to Changing ssh Limits

To avoid having to change limits on incoming `ssh` connections, you can do the following:

- Use a mounted directory and employ `$usecp` MoM parameters. See [section 9.7.9.2, “Avoiding Remote Transfers in Large Complexes”](#), on page 447.
- Use compression to service more requests with the same amount of hardware resources. See [section 9.7.9.3, “Improving Performance for ssh”](#), on page 447.

### 9.7.9.7 Getting Around Bandwidth Limits

If you have bandwidth limits, you can use a command such as `gsiftp`, which allows you to specify the bandwidth you want to use for file transfer. Follow the instructions in [section 9.7.5, “Using Custom File Transfer Mechanism”](#), on page 444.

## 9.7.10 General Advice on File Transfer

### 9.7.10.1 Enabling Passwordless Authentication

You must enable passwordless authentication so that job files can be staged in and out. You must also choose and set a file transfer mechanism such as `rsh` or `scp` for remote file copying. Before you set up the remote file copy mechanism, enable passwordless authentication for it.

Enable passwordless authentication for each machine in the complex, and for any machine from which or to which files will be transferred.

You can use any authentication method you want, such as a `shosts.equiv` file, an authorized keys file, or `.rhosts` authentication. You can choose a cipher and use encryption; balance the CPU time required by encryption with the CPU time required by MoMs and job tasks.

PBS requires that `rsh/rsh` and/or `ssh/scp` works between each pair of hosts where files will be transferred. Test whether you have succeeded by logging in as root, and using your chosen file transfer mechanism to copy a file between machines.

### 9.7.10.2 Using scp for Security

Unless your complex is a closed system, we recommend using `scp` instead of `rsh`, because `scp` is more secure.

### 9.7.10.3 Avoiding Asynchronous Writes to NFS

Asynchronous writes to an NFS server can cause reliability problems. If using an NFS file system, mount the NFS file system synchronously (without caching.)

### 9.7.10.4 Returning Output

If your site has disabled the use of remote operation functions ("r" commands) and output cannot be returned for jobs running on compute nodes, enable the use of the `cp` command by adding `$usecp` to the `$PBS_HOME/mom_priv/config` file on each login node. See [section 9.7.1, “Letting MoM Know Whether Transfer is Local or Remote”](#), on page 441.

### 9.7.10.5 Editing the pbs.conf File Under Windows

You can edit the `pbs.conf` file by calling the PBS program named `"pbs-config-add"`. For example, on Windows systems:

```
\Program Files (x86)\PBS\exec\bin\pbs-config-add "PBS_SCP=\winnt\scp.exe"
```

Do not edit `pbs.conf` directly; this could reset the permission on the file, which could prevent other users from running PBS.

### 9.7.10.6 The pbs\_rcp Command

#### 9.7.10.6.i Exit Values for pbs\_rcp

The `pbs_rcp` command exits with a non-zero exit status for any error. This tells MoM whether or not the file was delivered.

### 9.7.10.7 Caveats

- Output is not delivered if the path specified by `PBS_SCP` or `PBS_RCP` in `pbs.conf` is incorrect.
- When a job is rerun, its `stdout` and `stderr` files are sent to the server and stored in `PBS_HOME/spool`. When the job is sent out for execution again, its `stdout` and `stderr` are sent with it. The copy mechanism used for these file transfers is internal to PBS; you cannot alter it or manage it in any way.

## 9.8 Some Performance Tips

### 9.8.1 Configurable Job Dispatch Rate

You can use the `job_run_wait` scheduler attribute to instruct the scheduler to dispatch jobs without waiting, after waiting only for any `runjob` hooks to execute, or after waiting for any `execjob_*` hooks to execute. Allowed values for `job_run_wait`:

*none*: run next job immediately

*runjob\_hook*: run next job after waiting for any `runjob` hooks to execute

*execjob\_hook*: run next job after waiting for any `execjob_*` hooks to execute

#### 9.8.1.1 Caveats for Configurable Job Dispatch Rate

The `job_run_wait` scheduler attribute replaces the `throughput_mode` scheduler attribute. We recommend using `job_run_wait`, and we recommend not setting or using `throughput_mode`.

If you set `job_run_wait` to *none* or *runjob\_hook*, PBS changes `throughput_mode` to *True*.

If you set `job_run_wait` to *execjob\_hook*, PBS changes `throughput_mode` to *False*.

If you set `throughput_mode` to *True*, PBS changes `job_run_wait` to *runjob\_hook*.

If you set `throughput_mode` to *False*, PBS changes `job_run_wait` to *execjob\_hook*.

## 9.8.2 Improving Scheduling Performance

- If you limit the number of jobs queued in execution queues, you can speed up the scheduling cycle. See [section 4.5.8.2, “Limiting Number of Jobs Queued in Execution Queues”, on page 102](#).
- Avoid using dynamic resources where possible; see [section 5.4.4, “Static vs. Dynamic Resources”, on page 232](#).
- We give advice on minimizing the impact hooks can have on scheduling in [“Scheduling Impact of Hooks” on page 78 in the PBS Professional Hooks Guide](#).

## 9.8.3 Improving Communication Performance

- We give recommendations for improving communication daemon performance in [“Recommendations for Maximizing Communication Performance” on page 53 in the PBS Professional Installation & Upgrade Guide](#).
- You can use placement sets to keep job processes topologically close to one another; see [section 4.9.32, “Placement Sets”, on page 168](#).
- See our recommendations on file transfer performance improvement in [section 9.7.9, “Advice on Improving File Transfer Performance”, on page 447](#).

## 9.8.4 Improving Hook Speed

- See our hook performance recommendations in [“Performance Considerations” on page 79 in the PBS Professional Hooks Guide](#).

# 9.9 Temporary File Location for PBS Components

You can configure where all PBS components put their temporary files and directories on each system. You may want to avoid using the usual temporary file locations of `/tmp` and `/var/tmp`, because users tend to fill these up.

## 9.9.1 Default Location for Temporary Files

By default, on Linux platforms, PBS components put their temporary files and directories in `/var/tmp`. PBS uses this location because it is persistent across restarts or crashes, allowing diagnosis of a problem, whereas the contents of `/tmp` may be lost.

On Windows, the default location is `C:\WINNT\TEMP` if it is present, or `C:\WINDOWS\TEMP`.

## 9.9.2 Configuring Temporary File Location for PBS Components

You configure the location of temporary files and directories for PBS components by setting the value of the `PBS_TMPDIR` configuration parameter in the `/etc/pbs.conf` file on each system. Set this parameter to the directory to be used for storing temporary files and directories by all PBS components on that system.

After you set the location of temporary files and directories, restart all PBS components:

```
<path to init.d>/init.d/pbs restart
```

The location for temporary files and directories for PBS components is determined by the following settings, in order of decreasing precedence:

1. `$tmpdir` in `mom_priv/config` (affects `pbs_mom` only, not other components)
2. `PBS_TMPDIR` (for Linux) or `TMP` (for Windows) environment variable
3. `PBS_TMPDIR` in PBS configuration file
4. If none of the preceding settings are present, PBS uses default values:
  - `/var/tmp` (for Linux)
  - `C:\WINNT\TEMP` or `C:\WINDOWS\TEMP` (for Windows)

### 9.9.3 Requirements

- The specified directory must exist.  
If the configured temporary file location does not exist, PBS prints the following error message:  
`<command>: No such file or directory (2) in chk_file_sec, Security violation "<directory>" resolves to "<directory>"`  
`<command>: Unable to configure temporary directory.`
- The directory must be globally readable and writable.
- On Linux systems, the directory must have the sticky bit set in the file permissions.
- The directory must not present a security risk:
  - All parent directories of the configured temporary directory must be owned by a UID less than 11 and a GID less than 10.
    - If the assigned owner has write permission, the UID must be 10 or less.
    - If the assigned group has write permission, the GID must be 9 or less.
  - Each parent directory must not be writable by "other".

If a PBS component detects a security risk for a file or directory, it prints the following messages and exits:

```
<command>: Not owner (1) in chk_file_sec, Security violation "<directory>" resolves to
"<directory>"
<command>: Unable to configure temporary directory.
```

### 9.9.4 Advice and Recommendations for Temporary File Location

- Make sure that the location you choose for temporary files is cleaned periodically.
- In the past, some PBS components defaulted to `/tmp` for storing temporary files. All components now default to `/var/tmp`, which is most likely a persistent storage location. You should take this into account and adjust the cleaning of `/var/tmp` accordingly.
- If a PBS component prints a security error message and exits, fix the security problem and restart the component.

## 9.10 Administration Caveats

### 9.10.1 General Caveats

Do not manually delete files in PBS private directories.

### 9.10.2 Windows Caveats

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

## 9.11 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus.

## 9.12 Support for Hyperthreading

On Linux machines that have Hyper-Threading Technology, PBS can end up reporting and using the number of logical processors, instead of the number of physical CPUs, as the value for `resources_available.ncpus`.

PBS does not control how CPUs are allocated to processes within a job. That is handled by the OS kernel.

### 9.12.1 Linux Machines with HTT

On Linux, PBS uses the number of CPUs shown in `/proc/cpuinfo`. If the CPUs are hyper-threaded and hyper-threading is enabled, the number of virtual and physical CPUs is different.

### 9.12.2 Windows Machines with HTT

On Windows, PBS calls the `CPUCount` Windows function, which reports whether hyper-threading is enabled. If hyper-threading is enabled, MoM uses the number of physical CPUs. If hyper-threading is not enabled, MoM uses the number of CPUs reported by the OS. MoM logs whether or not hyper-threading is enabled.

### 9.12.3 Using Number of Physical CPUs

If you do not wish to use hyper-threading, you can configure PBS to use the number of physical CPUs. Do this by setting `resources_available.ncpus` to the number of physical cpus:

```
Qmgr: set node <vnode name> resources_available.ncpus=<number of physical CPUs>
```

### 9.12.4 Hyperthreading Caveats

On a cpusetted system, NEVER change the value for `resources_available.ncpus`, `resources_available.vmem`, or `resources_available.mem`.



---

## 9.13 How To...

### 9.13.1 How to Drain Jobs

You can drain jobs from the entire complex by setting up dedicated time. Do not allow jobs in the dedicated time queue. See [section 4.9.10, “Dedicated Time”, on page 128](#).

You can drain jobs from a specific set of vnodes by creating a reservation that blocks out those vnodes for the desired amount of time. See [section 4.9.37, “Reservations”, on page 196](#).

### 9.13.2 How to Find Out Which Daemons Should Be Running

On the host in question, look in `/etc/pbs.conf`, or the location pointed to in the `PBS_CONF_FILE` environment variable. Check the settings that specify whether each daemon should run. 1 means the daemon should run.

```
PBS_START_MOM
PBS_START_COMM
PBS_START_SERVER
PBS_START_SCHED
```

## 9.14 Avoiding Memory Hogging by Python Scripts

PBS server python hook scripts can cause memory leaks, which can use up too much memory. Previously, PBS periodically restarted the Python interpreter as a way to manage memory usage by Python hooks. As of version 2022.1.1, PBS offers Python garbage collection, in addition to restarting the interpreter, as a way to manage hook memory usage. PBS can do the following periodically:

- Trigger Python garbage collection
- Restart the Python interpreter

### 9.14.1 Using Garbage Collection

You can choose whether to use periodic garbage collection, and how often.

You specify the frequency of garbage collection in the new server attribute named `"python_gc_min_interval"`. The default value for this attribute is 30 seconds.

To avoid using garbage collection, set the value of this attribute to zero.

### 9.14.2 Restarting the Python Interpreter

You can choose whether to periodically restart the interpreter, and how often.

The default interval for restarting the Python interpreter is one hour; you can set this interval in the server attribute named `"python_restart_min_interval"`.

To avoid restarting the Python interpreter, set the value of this attribute to zero (this behavior is new; previous releases of PBS did not allow setting this value to zero).

### 9.14.3 Attributes for Managing Python Memory Usage

#### `python_gc_min_interval`

Server attribute

Specifies frequency for Python garbage collection. For no garbage collection, set this to zero.

Format: integer seconds

Default: 30 seconds

#### `python_restart_min_interval`

Server attribute

Specifies frequency for Python interpreter restart. For no interpreter restart, set this to zero.

Format: integer seconds

Default: 3600 seconds (1 hour)

# 10

# Managing Jobs

## 10.1 Routing Jobs

You can route jobs to various places and by various criteria. You can reject submission of jobs that request too much of a given resource. You can force jobs into the correct queues. You can have all jobs submitted to a routing queue, then route them to the correct execution queues. You can use peer scheduling to have jobs executed at other PBS complexes. You can use hooks to move jobs. For information on routing jobs, see [section 4.9.39, “Routing Jobs”, on page 205](#).

## 10.2 Limiting Number of Jobs Considered in Scheduling Cycle

If you limit the number of jobs in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the `queued_jobs_threshold` queue or server attribute. See [section 5.15.1.9, “How to Set Limits at Server and Queues”, on page 292](#).

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See [section 4.9.39, “Routing Jobs”, on page 205](#).

## 10.3 Allocating Resources to Jobs

You can make sure that jobs request or inherit any resources required to manage those jobs. If a job does not request a resource, you can make sure that the resource is allocated to the job anyway.

In order for limits to be effective, each job must request each limited resource. For a complete description of how limits work, see [section 5.15, “Managing Resource Usage”, on page 283](#).

You can create custom resources specifically to allocate them to jobs. These resources can be visible, alterable, and requestable by users, or invisible, unalterable, and unrequestable, or visible but unalterable and unrequestable. For instructions on creating invisible or unrequestable resources, see [section 5.14.2.4, “Specifying Resource Visibility”, on page 257](#).

You can alter a job's resource request using the following methods:

- You can set defaults for resources at the server or at each queue. This way, you can have jobs inherit specific values for the resources by routing them to special queues, where they inherit the defaults. For how jobs inherit resources, see [section 5.9.4, “Allocating Default Resources to Jobs”, on page 244](#). For how to specify default resources, see [section 5.9.3, “Specifying Job Default Resources”, on page 241](#).

For how resource defaults change when a job is moved, see [section 5.9.4.3, “Moving Jobs Between Queues or Servers Changes Defaults”](#), on page 245.

- You can use a hook to assign a specific resource value to a job, if a job requests the wrong value for a resource. For how to use a hook to assign a resource to a job, see the PBS Professional Hooks Guide. For examples of using hooks to assign resources to jobs, see *PBS Professional Plugins (Hooks) Guide*.
- You can use the `qalter` command to change a job's resource request. For how to use the `qalter` command, see [“qalter” on page 130 of the PBS Professional Reference Guide](#).
- You can set default arguments the `qsub` command via the `default_qsub_arguments` server attribute. For how to use default arguments to `qsub`, see [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

## 10.3.1 Viewing Resources Allocated to a Job

### 10.3.1.1 The `exec_vnode` Attribute

The `exec_vnode` attribute displayed via `qstat` shows the resources allocated from each vnode for the job.

The `exec_vnode` line looks like:

```
exec_vnode = (<vnode name>:ncpus=W:mem=X)+(<vnode name>:ncpus=Y:mem=Z)
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

gets an `exec_vnode` of

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb) +(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the `exec_vnode` might be:

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
+(VNC1:ncpus=2:mem=1gb+VNC2:ncpus=2:mem=1gb)
```

Note also that if a vnode is allocated to a job because the job requests an arrangement of *exclhost*, only the vnode name appears in the chunk. For example, if a job requesting

```
-l select 2:ncpus=4 -l place = exclhost
```

is placed on a host with 4 vnodes, each with 4 CPUs, the `exec_vnode` attribute looks like this:

```
exec_vnode = (VN0:ncpus=4)+(VN1:ncpus=4)+(VN2)+(VN3)
```

### 10.3.1.2 The `schedselect` Attribute

The resources allocated from a vnode are only those specified in the job's `schedselect` attribute. This job attribute is created internally by starting with the select specification and applying any server and queue `default_chunk` resource defaults that are missing from the select statement. The `schedselect` job attribute contains only vnode-level resources. The `exec_vnode` job attribute shows which resources are allocated from which vnodes. See [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).

### 10.3.1.3 Resources for Requeued Jobs

When a job is requeued due to an error in the prologue or initialization, the job's `exec_host` and `exec_vnode` attributes are cleared. The only exception is when the job is checkpointed and must be rerun on the exact same system. In this case, the `exec_host` and `exec_vnode` attributes are preserved.

---

## 10.4 Grouping Jobs By Project

### 10.4.1 PBS Projects

In PBS, a project is a way to organize jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's `project` attribute specifies the job's project. Each job can be a member of up to one project.

Projects are not tied to users or groups. One user or group may run jobs in more than one project. For example, user Bob runs JobA in ProjectA and JobB in ProjectB. User Bill runs JobC in ProjectA. User Tom runs JobD in ProjectB. Bob and Tom are in Group1, and Bill is in Group2.

### 10.4.2 Assigning Projects to Jobs

A job's project can be set in the following ways:

- At submission, using the `qsub -P` option; see [“qsub” on page 216 of the PBS Professional Reference Guide](#)
- After submission, via the `qalter -P` option; see [“qalter” on page 130 of the PBS Professional Reference Guide](#)
- Via a hook; see the PBS Professional Hooks Guide

### 10.4.3 Managing Resource Use by Project

PBS can apply limits to the amount of resources used by jobs in projects, or the number of queued and running jobs belonging to projects. See [section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 283](#).

### 10.4.4 Managing Jobs by Project

You can arrange for the jobs belonging to a project to run on designated hardware; see [section 4.4.4, “Allocating Resources by User, Project or Group”, on page 84](#). You can also run jobs belonging to a project in designated time slots; see [section 4.4.6, “Scheduling Jobs into Time Slots”, on page 88](#). For more information on routing by project, see [section 4.9.39, “Routing Jobs”, on page 205](#).

### 10.4.5 Viewing Project Information

Each job's project, if any, is specified in its `project` attribute. To see the value of this attribute, use the `qstat -f` option. See [“qstat” on page 200 of the PBS Professional Reference Guide](#).

### 10.4.6 Selecting Jobs by Project

You can select jobs according to their project using the `qselect -P` option. See [“qsig” on page 195 of the PBS Professional Reference Guide](#).

### 10.4.7 Default Project Value

The default value for a job's `project` attribute is `"_pbs_project_default"`. Any job submitted without a specified value for the `project` attribute is given the default value. If you explicitly set the value to `"_pbs_project_default"`, the server prints a warning message saying that the value has been set to the default. If you unset the value of the attribute in a hook, the value becomes the default value. Using `qalter -P ""` sets the value to the default.

## 10.4.8 Error Messages

When a job would exceed a limit by running, the job's comment field is set to an error message. See [“Run Limit Error Messages” on page 385 of the PBS Professional Reference Guide](#).

## 10.5 Job Prologue and Epilogue

As of 2020.1, the prologue and epilogue are **deprecated**.

You can run a site-supplied script or program before and/or after each job runs. This allows initialization or cleanup of resources, such as temporary directories or scratch files. The script or program that runs before the job is the *prologue*; the one that runs after the job is the *epilogue*.

The primary purpose of the prologue is to provide a site with some means of performing checks prior to starting a job. The epilogue can be used to requeue a checkpointed job. See [section 8.3.7.3, “Requeueing via Epilogue”, on page 398](#).

If you have any `execjob_prologue` hooks, they supersede the prologue, and run when the prologue would run, and if you have any `execjob_epilogue` hooks, they supersede the epilogue, and run when the epilogue would run.

If you are running the `cgroups` hook, any epilogue script will not run. The `cgroups` hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the `cgroups` hook, make your epilogue script into an `execjob_epilogue` hook instead.

You can run a shell script as your prologue or epilogue, or you can use an `execjob_prologue` and/or `execjob_epilogue` hook to do the work. If you already have a shell script prologue and/or epilogue, you can run each via an appropriate `execjob_prologue` or `execjob_epilogue` hook. We show how to do this in [section 10.5.2, “Using Hooks for Prologue and Epilogue”, on page 462](#).

### 10.5.1 Using Shell Scripts for Prologue and Epilogue

Only one prologue and one epilogue may be used per PBS server. The same prologue and/or epilogue runs for every job in the complex.

Each script may be either a shell script or an executable object file.

#### 10.5.1.1 When Shell Prologue and Epilogue Run

The prologue runs before the job is executed. The epilogue runs after the job terminates, including normal termination, job deletion while running, error exit, or even if `pbs_mom` detects an error and cannot completely start the job. If the job is deleted while it is queued, then neither the prologue nor the epilogue is run. If the job is discarded while running, for example when the server loses contact with the MoM, the epilogue does not run.

If a prologue or epilogue script is not present, MoM continues in a normal manner.

#### 10.5.1.2 Where Shell Prologue and Epilogue Run

When multiple vnodes are allocated to a job, these scripts are run only by the MoM on the primary execution host.

The prologue runs with its current working directory set to `PBS_HOME/mom_priv`, regardless of the setting of the `sandbox` job attribute.

The epilogue runs with its current working directory set to the job's staging and execution directory. This is also where the job shell script is run.

### 10.5.1.3 Shell Prologue and Epilogue Location

Both the prologue and the epilogue must reside in the `PBS_HOME/mom_priv` directory.

### 10.5.1.4 Shell Prologue and Epilogue Requirements

In order to be run, the script must adhere to the following rules:

- The script must be in the `PBS_HOME/mom_priv` directory
- The prologue must have the exact name "prologue" under Linux, or "prologue.bat" under Windows
- The epilogue must have the exact name "epilogue" under Linux, or "epilogue.bat" under Windows
- The script must be written to exit with one of the zero or positive exit values listed in [section 10.5.1.12, "Shell Prologue and Epilogue Exit Codes", on page 461](#). The negative values are set by MoM
- Under Linux, the script must be owned by root, be readable and executable by root, and cannot be writable by anyone but root
- Under Windows, the script's permissions must give "Full Access" to the local Administrators group on the local computer

### 10.5.1.5 Shell Prologue and Epilogue Environment Variables

The prologue and epilogue run with the following set in their environment:

- The contents of the `pbs_environment` file
- The `PBS_JOBDIR` environment variable

`TMPDIR` is not set in the prologue environment or the epilogue environment.

### 10.5.1.6 Shell Prologue and Epilogue Permissions

Both the prologue and epilogue are run under root on Linux, or under an Admin-type account on Windows, and neither is included in the job session.

### 10.5.1.7 Shell Prologue and Epilogue Arguments

The prologue is called with the following arguments:

**Table 10-1: Arguments to Prologue**

| Argument             | Description                             |
|----------------------|-----------------------------------------|
| <code>argv[1]</code> | Job ID                                  |
| <code>argv[2]</code> | Username under which the job executes   |
| <code>argv[3]</code> | Group name under which the job executes |



The epilogue is called with the following arguments:

**Table 10-2: Arguments to Epilogue**

| Argument | Description                                                                                 |
|----------|---------------------------------------------------------------------------------------------|
| argv[1]  | Job ID                                                                                      |
| argv[2]  | Username under which the job executes                                                       |
| argv[3]  | Group name under which the job executes                                                     |
| argv[4]  | Job name                                                                                    |
| argv[5]  | Session ID                                                                                  |
| argv[6]  | Requested built-in resources (job's Resource_List)                                          |
| argv[7]  | List of resources used (job's resources_used) gathered from the primary execution host only |
| argv[8]  | Name of the queue in which the job resides                                                  |
| argv[9]  | Account string, if one exists                                                               |
| argv[10] | Exit status of the job                                                                      |

### 10.5.1.8 Shell Epilogue Argument Caveats

Under Windows and with some Linux shells, accessing argv[10] in the epilogue requires a shift in positional parameters. To do this, the script must do the following:

1. Call the arguments with indices 0 through 9
2. Perform a shift /8
3. Access the last argument using %9%

For example:

```
cat epilogue
> #!/bin/bash
>
> echo "argv[0] = $0" > /tmp/epiargs
> echo "argv[1] = $1" >> /tmp/epiargs
> echo "argv[2] = $2" >> /tmp/epiargs
> echo "argv[3] = $3" >> /tmp/epiargs
> echo "argv[4] = $4" >> /tmp/epiargs
> echo "argv[5] = $5" >> /tmp/epiargs
> echo "argv[6] = $6" >> /tmp/epiargs
> echo "argv[7] = $7" >> /tmp/epiargs
> echo "argv[8] = $8" >> /tmp/epiargs
> echo "argv[9] = $9" >> /tmp/epiargs
> shift
> echo "argv[10] = $9" >> /tmp/epiargs
```

### 10.5.1.9 Standard Input to Shell Prologue and Epilogue

Both scripts have standard input connected to a system-dependent file. The default for this file is /dev/null.

### 10.5.1.10 Standard Output and Error for Shell Prologue and Epilogue

The standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. There is one exception: if a job is an interactive PBS job, the standard output and error of the epilogue is pointed to `/dev/null` because the pseudo-terminal connection used was released by the system when the job terminated.

### 10.5.1.11 Shell Prologue and Epilogue Timeout

When the scheduler runs a job, it waits until the prologue has ended. To prevent an error condition within the prologue or epilogue from delaying PBS, MoM places an alarm around the script's/program's execution. The default value is *30 seconds*. If the alarm timeout is reached before the script has terminated, MoM will kill the script. The alarm value can be changed via the `$prologalarm` MoM configuration parameter. See [section 8.6.7, “Prologue & Epilogue Running Time”, on page 417](#).

### 10.5.1.12 Shell Prologue and Epilogue Exit Codes

Normally, the prologue and epilogue programs should exit with a zero exit status. The prologue and epilogue should be written to exit with one of the zero or positive values listed here. When there is a problem with the script, MoM sets the exit value to one of the negative values. Exit status values and their impact on the job are listed in the following table:

**Table 10-3: Prologue and Epilogue Exit Codes**

| Exit Code | Meaning                                                                                          | Prologue                  | Epilogue                                                                   |
|-----------|--------------------------------------------------------------------------------------------------|---------------------------|----------------------------------------------------------------------------|
| -4        | The script timed out (took too long).                                                            | The job will be requeued. | Ignored                                                                    |
| -3        | The <code>wait(2)</code> call waiting for the script to exit returned with an error.             | The job will be requeued  | Ignored                                                                    |
| -2        | The input file to be passed to the script could not be opened.                                   | The job will be requeued. | Ignored                                                                    |
| -1        | The script has a permission error, is not owned by root, and/or is writable by others than root. | The job will be requeued. | Ignored                                                                    |
| 0         | The script was successful.                                                                       | The job will run.         | Ignored                                                                    |
| 1         | The script returned an exit value of 1.                                                          | The job will be aborted.  | Ignored                                                                    |
| >1        | The script returned a value greater than one.                                                    | The job will be requeued. | Ignored                                                                    |
| 2         | The script returned a value of 2.                                                                | The job will be requeued. | If the job was checkpointed under the control of PBS, the job is requeued. |

MoM records in her log any case of a non-zero prologue or epilogue exit code, at event class 0x0001.

### 10.5.1.13 Shell Prologue and Epilogue Limitations and Caveats

- Consider having your epilogue write a lock file so that it can detect whether it is being run more than once for a job.
- You must exercise great caution in setting up the prologue to prevent jobs from being flushed from the system.
- Interactive-batch jobs cannot be requeued if the epilogue exits with a non-zero status. When this happens, these jobs are aborted.
- The prologue and epilogue cannot be used to modify the job environment or to change limits on the job.
- If any `execjob_prologue` hooks exist, they are run, and the prologue is not run.
- If any `execjob_epilogue` hooks exist, they are run, and the epilogue is not run.
- If you are running the `cgroups` hook, any epilogue script will not run. The `cgroups` hook has an `execjob_epilogue` event which takes precedence over an epilogue script, so if you are running the `cgroups` hook, make your epilogue script into an `execjob_epilog` hook instead.

## 10.5.2 Using Hooks for Prologue and Epilogue

You can run `execjob_prologue` and `execjob_epilogue` hooks to do whatever setup and cleanup you need before and after jobs run. Note that these hooks supersede the shell prologue and epilogue and prevent them from running. See ["execjob\\_prologue: Event Just Before Execution of Top-level Job Process" on page 104 in the PBS Professional Hooks Guide](#), and ["execjob\\_epilogue: Event Just After Killing Job Tasks" on page 111 in the PBS Professional Hooks Guide](#).

However, you can run your shell prologue and epilogue using `execjob_prologue` and `execjob_epilogue` hooks, and we provide an example hook called `run_pelog_shell.py`. The hook is included in `$PBS_EXEC/unsupported` as `run_pelog_shell.py`, along with its configuration file, `run_pelog_shell.ini`. You can see the contents at ["execjob\\_prologue and execjob\\_epilogue Hook Examples" on page 299 in the PBS Professional Hooks Guide](#).

You can use this hook when the `execjob_prologue` and `execjob_epilogue` events are used in other hooks, such as the `cgroups` hook, and you still want to run the classic prologue and epilogue scripts we describe in [section 10.5.1, "Using Shell Scripts for Prologue and Epilogue"](#), on page 458. Additionally, the hook introduces parallel prologue and epilogue shell scripts.

On the primary execution host (the first host listed in `PBS_NODEFILE`), the standard naming convention of 'prologue' and 'epilogue' apply. Parallel prologues and epilogues use the naming conventions 'pprologue' and 'pepilogue', respectively, but run only on the secondary execution hosts. On Windows, parallel prologues and epilogues expect a '.bat' extension, which results in 'pprologue.bat' and 'pepilogue.bat'. This hook does the normal checks PBS does to start a prologue, such as permissions, etc., for UNIX. This hook also allows you to use a parallel prologue/epilogue (pprologue/pepilogue).

Parallel prologues will not run until a task associated with the job (i.e. via `pbs_attach`, `pbs_tmrsh`) begins on the secondary execution hosts. Parallel epilogues run only if the prologue ran successfully on the primary execution host. Only the primary execution host will have a value for `resources_used` in epilogue argument \$7.

We assume the same requirements as listed for prologues/epilogues for running all types of prologue and epilogue shell scripts in [section 10.5.1.4, "Shell Prologue and Epilogue Requirements"](#), on page 459.

By default, parallel prologue/epilogue is set to *False*. To enable parallel behavior, edit the configuration file and set `ENABLE_PARALLEL` to *True*.

The hook kills the prologue/epilogue 5 seconds before the `hook_alarm` timeout. At this point the job is requeued/deleted depending on the value of `DEFAULT_ACTION`. The `hook_alarm` time defaults to 30 seconds, giving the prologue/epilogue approximately 25 seconds to complete.

### 10.5.2.1 Installing Prologue and Epilogue Hooks

You could create a single hook that runs on both the `execjob_prologue` and the `execjob_epilogue` events, but to ensure execution order we separate them into the individual events by creating two separate hooks that use the same hook script.

Edit `run_pelog_shell.ini` to make configuration changes, then create and import the hook as we show here.

As root, run the following:

```
qmgr << EOF
create hook run_prologue_shell
set hook run_prologue_shell event = execjob_prologue
set hook run_prologue_shell enabled = true
set hook run_prologue_shell order = 1
set hook run_prologue_shell alarm = 35
import hook run_prologue_shell application/x-python default run_pelog_shell.py
import hook run_prologue_shell application/x-config default run_pelog_shell.ini

create hook run_epilogue_shell
set hook run_epilogue_shell event = execjob_epilogue
set hook run_epilogue_shell enabled = true
set hook run_epilogue_shell order = 999
set hook run_prologue_shell alarm = 35
import hook run_epilogue_shell application/x-python default run_pelog_shell.py
import hook run_epilogue_shell application/x-config default run_pelog_shell.ini
EOF
```

Any further configuration changes to `run_pelog_shell.ini` require re-importing the file to both hooks:

```
qmgr << EOF
import hook run_prologue_shell application/x-config default run_pelog_shell.ini
import hook run_epilogue_shell application/x-config default run_pelog_shell.ini
EOF
RERUN=14
DELETE=6
DEBUG=False
```

We show the defaults for the following constants in `run_pelog_shell.ini`. We show the contents of the file in ["execjob prologue and execjob epilogue Hook Examples" on page 299 in the PBS Professional Hooks Guide](#). You can set them to match site preferences:

```
ENABLE_PARALLEL=False
VERBOSE_USER_OUTPUT=False
DEFAULT_ACTION=RERUN
TORQUE_COMPAT=False
```

## 10.6 Linux Shell Invocation

When PBS starts a job, it invokes the user's login shell, unless the user submitted the job with the `-S` option. PBS passes the job script, which is a shell script, to the login process.

PBS passes the name of the job script to the shell program. This is equivalent to typing the script name as a command to an interactive shell. Since this is the only line passed to the script, standard input will be empty to any commands. This approach offers both advantages and disadvantages:

## 10.6.1 Advantages

- Any command which reads from standard input without redirection will get an EOF.
- The shell syntax can vary from script to script. It does not have to match the syntax for the user's login shell. The first line of the script, even before any #PBS directives, should be

```
#!/shell
```

where *shell* is the full path to the shell of choice, */bin/sh*, */bin/csh*, ...

The login shell will interpret the *#!* line and invoke that shell to process the script.

## 10.6.2 Disadvantages

- An extra shell process is run to process the job script.
- If the script does start with a *#!* line, the wrong shell may be used to interpret the script and thus produce errors.
- If a non-standard shell is used via the *-S* option, it will not receive the script, but its name, on its standard input.

## 10.7 When Job Attributes are Set

The attributes of a job are set at various points in the life of the job. For a description of each job attribute, see [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).

### 10.7.1 Job Attributes Set By qsub Command

Before the job is passed to the server, the *qsub* command sets these job attributes, in this order:

1. Attributes specified as options on the command line
2. Attributes specified in #PBS directives within the job script
3. Job attributes specified in the *default\_qsub\_arguments* server attribute
4. If the following job attributes have not already been set, they are set as follows:
  - *Job\_Name*: set to the file name of the job script, or to "*STDIN*" if the script is entered via standard input
  - *Checkpoint*: set to "*u*" for unspecified.
  - *Hold\_Types*: set to "*n*"
  - *Join\_Path*: set to "*n*"
  - *Keep\_Files*: set to "*n*"
  - *Mail\_Points*: set to "*a*" for abort
  - *Priority*: set to *0* (zero)
  - *Rerunnable*: set to *True*
  - *run\_count*: can be set by job submitter
  - *Variable\_List*: the *qsub* command sets the following variables and appends them to the existing value of *Variable\_List*: *PBS\_O\_HOME*, *PBS\_O\_LANG*, *PBS\_O\_LOGNAME*, *PBS\_O\_PATH*, *PBS\_O\_MAIL*, *PBS\_O\_SHELL*, *PBS\_O\_WORKDIR*, *PBS\_O\_TZ*, and *PBS\_O\_SYSTEM*
  - *Submit\_arguments*: set to any submission arguments on the command line

## 10.7.2 Job Attributes Set at Server

When the job is passed from the `qsub` command to the server, the raw job information is available to any job submission hooks, which can alter the information. Once the job is at the server, the server sets the following attributes:

- **Job\_Owner:** set to `<username>@<submission host name>`
- **Variable\_List:** the following are added to the job's `Variable_List` attribute: `PBS_O_QUEUE`, `PBS_O_HOST`
- **Output\_Path:** if not yet specified, the `Output_Path` attribute is set
- **Error\_Path:** if not yet specified, the `Error_Path` attribute is set
- **Rerunable:** if the job is interactive, the `Rerunable` attribute is set to *False*
- **run\_count:** incremented each time job is run
- **project:** if unset, the `project` attribute is set to `"_pbs_project_default"`.
- **Read-only attributes:** the server sets the job's read-only attributes; see [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#)
- **Resource\_List:** adjusted to include inherited resources specified in the queue and server `Resources_Default` attributes, if those resources are not yet in the list
- **Comment** set when job is sent for execution or rejected; see [section 10.7.3.1, “Comment Set When Running Job”, on page 465](#)

## 10.7.3 Attributes Changed by Operations on Jobs

### 10.7.3.1 Comment Set When Running Job

Before the server sends the job to an execution host, the server sets the job's comment to "Job was sent for execution at `<time>` on `<execvnode>`".

After the server gets a confirmation from the MoM, the server updates the job's comment to "Job run at `<time>` on `<execvnode>`".

If the MoM rejects the job, the server changes the job comment to "Not Running: PBS Error: Execution server rejected request".

### 10.7.3.2 Attributes Changed When Moving Job

If you move a job to a different queue or server, any default resources from the current queue or server are removed, and new defaults are inherited. See [section 5.9.4.3, “Moving Jobs Between Queues or Servers Changes Defaults”, on page 245](#). For information on the `qmove` command, see [“qmove” on page 175 of the PBS Professional Reference Guide](#).

### 10.7.3.3 Attributes Changed When Altering Job

When the `qalter` command is used to alter a job, the changes to the job are changes to the equivalent job attributes. See [“qalter” on page 130 of the PBS Professional Reference Guide](#).

### 10.7.3.4 Attributes Changed When Requeueing or Rerunning a Job

When a job is requeued or rerun, its `exec_vnode` and/or `exec_host` attributes may be changed. The job may end up running on different vnodes. See [“qrun” on page 181 of the PBS Professional Reference Guide](#).

Each time a job is run, its `run_count` attribute is incremented by the server.

### 10.7.3.5 Attributes Changed by Holding or Releasing a Job

When a job is held using the `qhold` command, or released using the `qrls` command:

- The `Hold_Types` attribute reflects the change
- The `job_state` attribute may be changed

See [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#) and [“qhold” on page 150 of the PBS Professional Reference Guide](#).

### 10.7.3.6 Attributes Changed by Suspending or Resuming a Job

When a job is suspended or resumed using the `qsig` command, the job's `job_state` attribute reflects the change in state.

See [“qsig” on page 195 of the PBS Professional Reference Guide](#).

## 10.8 Job Termination

A job can be terminated for the following reasons:

- You or the submitter can use `qdel` to kill the job
- The job can be preempted and requeued
- The job can go over a limit and be killed
- The job is submitted to a routing queue, and can never be routed (accepted by a destination queue)
- The server is restarted and the job cannot be recovered
- The job specifies a dependency that fails or is terminated
- The job is killed by a signal

### 10.8.1 Normal Job Termination

When there is no `$action terminate` script and a running job is terminated, via the `qdel <job ID>` command, because of a server shutdown, or because the job has exceeded a limit, PBS waits for a configurable amount of time between sending a `SIGTERM` and a `SIGKILL` signal to the job. The amount of time is specified in the `kill_delay` queue attribute. The default value for this attribute is *10 seconds*. PBS takes the following steps.

For a single-vnode job:

1. PBS sends the job a `SIGTERM`
2. PBS waits for the amount of time specified in the `kill_delay` queue attribute
3. PBS sends the job a `SIGKILL`

For a multi-vnode job:

1. The primary execution host MoM sends a `SIGTERM` to all processes on the primary execution host
2. If any of the processes of the top task of the job are still running, PBS waits a minimum of *kill\_delay* seconds
3. The primary execution host MoM sends a `SIGKILL` to all remaining job processes on the primary execution host
4. The subordinate MoMs send a `SIGKILL` to all their processes belonging to this job



## 10.8.2 Using the `qdel` Command to Terminate a Job

You can delete a job using the `qdel` command. See [“qdel” on page 143 of the PBS Professional Reference Guide](#).

```
qdel <job ID>
```

If there is an `$action terminate` script, it is used to terminate the job.

If there is no `$action terminate` script, the SIGTERM-delay-SIGKILL sequence described in [section 10.8.1, “Normal Job Termination”, on page 466](#) is used to terminate the job.

This command does not terminate provisioning jobs.

```
qdel -wforce <job ID>
```

If MoM is reachable, MoM sends the job a SIGKILL signal, and files are staged out. If MoM is unreachable, the server discards the job. The job may or may not continue to run on the execution host(s).

This command terminates provisioning jobs.

## 10.8.3 Killing Job Processes

If you need to kill job processes, you can use the `printjob` command to find the job's session ID, and then kill those processes. See [“printjob” on page 128 of the PBS Professional Reference Guide](#).

## 10.8.4 Hooks and Job Termination

If you `qdel` a job, any `execjob_preterm` hooks run on all the hosts allocated to a job. On the primary execution host, the hook executes when the job receives a signal from the server for the job to terminate. On a sister host, this hook executes when the sister receives a request from the primary execution host MoM to terminate the job, just before the sister signals the task on this host to terminate.

The `execjob_preterm` hook does not run for any other job termination. For example, it does not run on a `qrerun` or when a job goes over its limit.

See [“execjob\\_preterm: Event Just Before Killing Job Tasks” on page 110 in the PBS Professional Hooks Guide](#).

## 10.8.5 Configuring Site-specific Job Termination

The default behavior of PBS is for MoM to terminate a job under the following circumstances:

- The job's usage of a resource exceeds the limit requested
- The job is deleted by the server on shutdown
- The job is deleted via the `qdel` command

MoM normally uses SIGTERM, waits for the amount of time specified in the queue's `kill_delay` attribute, then issues a SIGKILL. See [section 10.8, “Job Termination”, on page 466](#).

You may want PBS to run your own job termination script in place of the normal action. The termination script is run in place of a SIGTERM. The termination script runs only on the primary execution host. After the top job process is terminated, a KILL signal is sent to any other job processes running on other hosts.

You can define the desired termination behavior by specifying the script you want to run in the `$action terminate` parameter in the Version 1 configuration file. The `$action terminate` parameter takes this form:

```
$action terminate <timeout> ! <path to script> [args]
```

Where

`<timeout>` is the time, in seconds, allowed for the script to complete. A value of **zero** (0) indicates infinite time is allowed for the script to run.

`<path to script>` is the path to the script. If it is a relative path, it is evaluated relative to the `PBS_HOME/mom_priv` directory.

`<args>` are optional arguments to the script. Values for `<args>` may be any string not starting with a percent sign ("%").

Arguments with a percent sign, making up any of the following keywords, are replaced by MoM with the corresponding value:

**Table 10-4: \$action terminate Keywords**

| Keyword             | Value Used by MoM                  |
|---------------------|------------------------------------|
| <code>%jobid</code> | Job ID                             |
| <code>%sid</code>   | Session ID of task (job)           |
| <code>%uid</code>   | Execution UID of job               |
| <code>%gid</code>   | Execution GID of job               |
| <code>%login</code> | Login name associated with UID     |
| <code>%owner</code> | Job owner in form <i>name@host</i> |
| <code>%auxid</code> | Auxiliary ID (system-dependent)    |

### 10.8.5.1 Requirements for Termination Script

The script should exit with a value of **zero** when the job is terminated successfully. If the script exits successfully (with a zero exit status and before the time-out period), PBS does not send any signals or attempt to terminate the job. It is the responsibility of the termination script in this situation to ensure that the job has been terminated.

The script should exit with a non-zero value if the job was not successfully terminated. If the script exits with a non-zero exit status, the job is sent **SIGKILL** by PBS.

If the script does not complete in the time-out period, it is aborted and the job is sent **SIGKILL**.

### 10.8.5.2 Examples of Configuring Termination

Linux:

Example 10-1: To use a 60-second timeout, run `PBS_HOME/mom_priv/endjob.sh`, and pass the job's session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.sh %sid %uid %jobid
```

Example 10-2: To use an infinite timeout, run the system `kill` command with the signal **13**, and pass the job's session ID:

```
$action terminate 0 !/bin/kill -13 %sid
```

Windows:

Example 10-3: To use a 60-second timeout, run `endjob.bat`, and pass the job's session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.bat %sid %uid %jobid
```

Example 10-4: To use an infinite timeout, run the `pbskill` command, and pass the job's session ID:

```
$action terminate 0 !"C:/Program Files/PBS Pro/exec/bin/pbskill" %sid
```

---

### 10.8.5.3 Caveats and Restrictions on Termination

Under Windows, *<path to script>* must have a ".bat" suffix since it will be executed under the Windows command prompt `cmd.exe`. If the *<path to script>* specifies a full path, be sure to include the drive letter so that PBS can locate the file. For example, `C:\winnt\temp\terminate.bat`. The script must be writable by no one but an Administrator-type account.

## 10.8.6 Killing Jobs with a Signal

You or the job owner can kill a job by sending a kill signal to a job via `qsig`.

If a job is terminated via a signal while it is in the process of being sent to the execution host, the following happens:

- PBS writes a server log message:  
`Job;<job ID>;Terminated on signal <signal number>`
- The job is requeued
- If `qrun` is used to run the job, `qrun` does not set the job's comment

## 10.9 Job Exit Status Codes

The exit status of a job may fall in one of three ranges, listed in the following table:

**Table 10-5: Job Exit Status Ranges**

| Exit Status Range          | Reason                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $X < 0$                    | The job could not be executed            | See <a href="#">Table 10-6, “Job Exit Codes,” on page 471</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| $0 \leq X < 128$           | Exit value of shell or top process       | <p>This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the <code>.logout</code> script if the user has such a script (<code>csch</code>).</p> <p>The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| $X \geq 128$ and $X < 254$ | Job was killed with a signal             | <p>This means the job was killed with a signal. The signal is given by <math>X \text{ modulo } 128</math> (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (<math>137 \% 128 = 9</math>).</p> <p>The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see <code>wait(2)</code> or <code>waitpid(2)</code> for more information), are the value of the signal that killed the job.</p> <p>To interpret (or "decode") the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a <code>SIGTERM</code> (e.g. <math>143 - 128 = 15</math>, signal 15 is <code>SIGTERM</code>). See the <code>kill(1)</code> manual page for a mapping of signal numbers to signal name on your operating system.</p> |
| $X = 254$                  | Job had an <code>execve()</code> failure | <p>This means that the job experienced a failure during <code>execve()</code>, e.g.:</p> <ul style="list-style-type: none"> <li>The file or a script interpreter is not a regular file</li> <li>Execute permission is denied for the file or a script</li> <li>The filesystem is mounted <code>noexec</code></li> <li>The file pathname or a script or ELF interpreter does not exist</li> </ul> <p>For more information, see the <code>execve()</code> man page.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

The exit status of jobs is recorded in the PBS server logs and the accounting logs.

Negative exit status indicates that the job could not be executed. Negative exit values are listed in the table below:

**Table 10-6: Job Exit Codes**

| Exit Code | Name                              | Description                                                                                                                             |
|-----------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| 0         | <i>JOB_EXEC_OK</i>                | Job execution was successful                                                                                                            |
| -1        | <i>JOB_EXEC_FAIL1</i>             | Job execution failed, before files, no retry                                                                                            |
| -2        | <i>JOB_EXEC_FAIL2</i>             | Job execution failed, after files, no retry                                                                                             |
| -3        | <i>JOB_EXEC_RETRY</i>             | Job execution failed, do retry                                                                                                          |
| -4        | <i>JOB_EXEC_INITABT</i>           | Job aborted on MoM initialization                                                                                                       |
| -5        | <i>JOB_EXEC_INITRST</i>           | Job aborted on MoM initialization, checkpoint, no migrate                                                                               |
| -6        | <i>JOB_EXEC_INITRMG</i>           | Job aborted on MoM initialization, checkpoint, ok migrate                                                                               |
| -7        | <i>JOB_EXEC_BADRESRT</i>          | Job restart failed                                                                                                                      |
| -10       | <i>JOB_EXEC_FAILUID</i>           | Invalid UID/GID for job                                                                                                                 |
| -11       | <i>JOB_EXEC_RERUN</i>             | Job was rerun                                                                                                                           |
| -12       | <i>JOB_EXEC_CHKP</i>              | Job was checkpointed and killed                                                                                                         |
| -13       | <i>JOB_EXEC_FAIL_PASSWORD</i>     | Job failed due to a bad password                                                                                                        |
| -14       | <i>JOB_EXEC_RERUN_ON_SIS_FAIL</i> | Job was requeued (if rerunnable) or deleted (if not) due to a communication failure between the primary execution host MoM and a Sister |
| -15       | <i>JOB_EXEC_QUEST</i>             | Requeue job for restart from checkpoint                                                                                                 |
| -16       | <i>JOB_EXEC_FAILHOOK_RERUN</i>    | Job execution failed due to hook rejection; requeue for later retry                                                                     |
| -17       | <i>JOB_EXEC_FAILHOOK_DELETE</i>   | Job execution failed due to hook rejection; delete the job at end                                                                       |
| -18       | <i>JOB_EXEC_HOOK_RERUN</i>        | A hook requested for job to be requeued                                                                                                 |
| -19       | <i>JOB_EXEC_HOOK_DELETE</i>       | A hook requested for job to be deleted                                                                                                  |
| -20       | <i>JOB_EXEC_RERUN_MS_FAIL</i>     | Job requeued because server couldn't contact the primary execution host MoM                                                             |

### 10.9.1 Job Exit Status Between 0 and 128 (or 256)

This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the `.logout` script if the user has such a script (`cs`h).

### 10.9.2 Job Exit Status $\geq 128$ and $< 254$

This means the job was killed with a signal. The signal is given by  $X \text{ modulo } 128$  (or 256 on some systems). For example an exit value of 137 means the job's top process was killed with signal 9 ( $137 \% 128 = 9$ ).

The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see `wait(2)` or `waitpid(2)` for more information), are the value of the signal that killed the job.

To interpret (or "decode") the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a **SIGTERM** (e.g.  $143 - 128 = 15$ , signal 15 is **SIGTERM**). See the `kill(1)` manual page for a mapping of signal numbers to signal name on your operating system.

## Job Exit Status = 254

Exit status 254 indicates that the job experienced a failure during execution of a script or program by `execve()`. Possible causes:

- The file or a script interpreter is not a regular file
- Execute permission is denied for the file or a script
- The filesystem is mounted `noexec`
- The file pathname or a script or ELF interpreter does not exist

For more information, see the `execve()` man page.

### 10.9.3 Logging Job Exit Status

The exit status of jobs is recorded in the PBS server logs and the accounting logs.

### 10.9.4 Exit Status of Interactive Jobs

The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.

## 10.10 Rerunning or Requeueing a Job

You can re-run a job using the `qrerun` command. To re-run a job means to kill it, and requeue it in the execution queue from which it was run. See [“qrerun” on page 181 of the PBS Professional Reference Guide](#).

### 10.10.1 Requeueing a Job on a Dead Node

Before you requeue a job on a node you know to be dead, use `qmgr` to mark the node as *Down*. When the node is marked *Down*, `qrerun` the job.

### 10.10.2 Output from a Re-run Job

When you re-run a job, the job's existing standard output and error files are copied back to the server host and stored in `PBS_HOME/spool`. They are then sent with the job to MoM when the job is again run. The output of a job that is re-run is appended to the output from prior runs of the same job.

### 10.10.3 Requeueing Caveats

- When requeueing a job fails, for example because the queue does not exist, the job is deleted.
- If a job's `run_count` attribute is already at the limit (20), and you requeue the job, the job will be held the next time the scheduler tries to run it.

---

## 10.10.4 Caveats for Jobs Started by PBS

PBS attempts to run a job a certain number of times before placing a hold on the job. You cannot prevent a job from being held after this number of attempts. You must explicitly release the hold.

## 10.11 Job IDs

### 10.11.1 Format of Job IDs

Job Identifier

`<sequence number>[.<server name>][@<server>]`

Job Array Identifier

Job array identifiers are a sequence number followed by square brackets:

`<sequence number>[[.<server name>][@<server>]`

Example:

`1234[]`

Note that some shells require that you enclose a job array ID in double quotes.

### 10.11.2 Range of IDs

The largest allowed value for a job ID or job array ID is set in the [max\\_job\\_sequence\\_id](#) server attribute. Minimum allowed is 9999999. Maximum allowed is 999999999999. After this has been reached, job IDs start again at zero.

### 10.11.3 Job IDs and Moving Jobs

If a job is qmoved from one server to another, the job's ID does not change.

### 10.11.4 Job IDs and Requeueing and Checkpoint/Restart

If a job is requeued without being checkpointed, or checkpointed and requeued, it keeps its original job ID.

## 10.12 Where to Find Job Information

Information about jobs is found in `PBS_HOME/server_priv/jobs` and `PBS_HOME/mom_priv/jobs`.

### 10.12.1 Deleted Jobs

If PBS tries to requeue a job and cannot, for example when the queue doesn't exist, the job is deleted.

### 10.12.2 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.



---

### 10.12.3 Job Information When Server is Down

When the PBS server is down, you can use the `pbs_datservice` command to start the PBS data service by hand, and then run the `printjob` command at the server host. See [“pbs\\_datservice” on page 61 of the PBS Professional Reference Guide](#) and [“printjob” on page 128 of the PBS Professional Reference Guide](#).

### 10.12.4 Job Information on Execution Host

You can use the `printjob` command to look at job information on the execution host. See [“printjob” on page 128 of the PBS Professional Reference Guide](#).

## 10.13 Job Directories

PBS jobs use two kinds of directories:

- The job's *staging and execution directory*, into which input files are staged, and from which output files are staged. It is also the current working directory for the job script, for tasks started via the `pbs_tm( )` API, and for the epilogue.
- The job's *temporary directory*, where the job can create scratch files if necessary. The root of this directory is specified in the `$tmpdir` MoM configuration parameter. PBS creates the temporary directory, then sets the `TMPDIR` job environment variable to the path of the temporary directory. The job can then use this environment variable.

### 10.13.1 Staging and Execution Directories for Job

A job's *staging and execution directory* is the directory to which input files are staged, and from which output files are staged. It is also the current working directory for the job script, for tasks started via the `pbs_tm( )` API, and for the epilogue.

For multi-host jobs, PBS stages files to and from the primary execution host only. The job submitter specifies files and directories to be staged via the job's `stagein` and `stageout` attributes, which have this format:

*execution\_path@storage\_host:storage\_path*

The *execution\_path* is the path to the staging and execution directory. On `stagein`, *storage\_path* is the path where the input files normally reside, and on `stageout`, *storage\_path* is the path where output files will end up.

Make sure that each execution host can provide an area for staging and execution directories for jobs.

#### 10.13.1.1 Using Job-specific Staging and Execution Directories

Each PBS user may submit several jobs at once, and each job may need to have data files staged in or out. To prevent collisions, PBS can create a job-specific staging and execution directory for each job.

If all users on a host have home directories, PBS can create the staging and execution directories for each job in the job submitters' home directories. If users do not have home directories, you can designate a directory for the task by setting the `$jobdir_root` MoM parameter to that location.

Whether or not PBS creates job-specific staging and execution directories for a job is controlled by the job's `sandbox` attribute:

- If the job's `sandbox` attribute is set to *PRIVATE*, PBS creates a staging and execution directory for each job, in the location specified by the `$jobdir_root` MoM parameter. If the `$jobdir_root` parameter is unset, PBS creates job-specific staging and execution directories in the job submitter's home directory.
- If the job's `sandbox` attribute is set to *HOME* or is unset, PBS does not create job-specific staging and execution directories. Instead PBS uses the job submitter's home directory.

Using the server's `default_qsub_arguments` attribute, you can specify the default for the `sandbox` attribute for all jobs. By default, the `sandbox` attribute is not set.

The submitter can set the `sandbox` attribute via `qsub`, for example:

```
qsub -Wsandbox=PRIVATE
```

The `-Wsandbox` option to `qsub` overrides `default_qsub_arguments`. The job's `sandbox` attribute cannot be altered while the job is executing.

### 10.13.1.2 Using Shared Directories for Staging and Execution

Using a shared directory for job staging and execution is a little more complicated when nodes are released early from a job. Normally each MoM on a sister node that is being released cleans up its own files upon release. However, if the directory is shared, you need to prevent those sister MoM(s) from prematurely cleaning up job files before the job has finished. This is an issue whether or not the directory is the user home directory. You take care of this by specifying whether the directory is shared via the `$jobdir_root` MoM parameter:

- When staging and execution directories are to be created in a shared (e.g. NFS) directory specified in `$jobdir_root`, set the *shared* directive after the directory name:  
`$jobdir_root <directory name> shared`
- If job submitter home directories are shared, tell MoM:  
`$jobdir_root PBS_USER_HOME shared`

### 10.13.1.3 Examples of Setting Location for Creation of Staging and Execution Directories

To tell PBS to create job staging and execution directories created under `/r/shared`, so that each job gets `/r/shared/<job-specific directory>`, put the following line in MoM's configuration file:

```
$jobdir_root /r/shared
```

To tell PBS to use `/scratch` when it is a shared directory:

```
$jobdir_root /r/shared shared
```

To tell PBS to use shared submitter home directories:

```
$jobdir_root PBS_USER_HOME shared
```

To tell PBS to use non-shared submitter home directories, leave the `$jobdir_root` parameter blank.

### 10.13.1.4 Options, Attributes and Environment Variables Affecting Staging

PBS sets the environment variable `PBS_JOBDIR` to the pathname of the staging and execution directory on the primary execution host. `PBS_JOBDIR` is added to the job script process, any job tasks created by the `pbs_tm( )` API, the prologue and epilogue, and the MoM `$action` scripts.

The job's `jobdir` attribute is read-only, and is also set to the pathname of the staging and execution directory on the primary execution host. You can view the `jobdir` attribute via the `-f` option to `qstat`.

The following table lists the options, attributes, etc., affecting staging:

**Table 10-7: Options, Attributes, Environment Variables, etc., Affecting Staging**

| Option, Attribute, Environment Variable, etc. | Effect                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MoM's <code>\$jobdir_root</code> parameter    | <p>Directory under which PBS creates job-specific staging and execution directories. Defaults to user's home directory if unset. If <code>\$jobdir_root</code> is unset, the user's home directory must exist. If <code>\$jobdir_root</code> is set but does not exist when MoM starts, MoM will abort. If <code>\$jobdir_root</code> is set but does not exist when MoM tries to run a job, MoM will kill the job. Permissions on the directory specified in this option must be <code>1777</code>.</p> <p>When you set <code>\$jobdir_root</code> to a shared (e.g. NFS) directory, tell MoM it is shared by setting the <i>shared</i> directive after the directory name:</p> <pre><code>\$jobdir_root &lt;directory name&gt; shared</code></pre> <p>If the user home directories are shared, tell MoM they are shared:</p> <pre><code>\$jobdir_root PBS_USER_HOME shared</code></pre> <p>Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when sister nodes are released, those sister MoMs would normally clean up their own files upon release, but this could cause problems in a shared directory. So if <code>\$jobdir_root</code> or submitter home directories are shared, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.</p> <p>Example of using a shared non-submitter-home directory:</p> <pre><code>\$jobdir_root /r/shared shared</code></pre> <p>Example of using shared submitter home directories:</p> <pre><code>\$jobdir_root PBS_USER_HOME shared</code></pre> <p>Example of a non-shared directory:</p> <pre><code>\$jobdir_root /scratch/foo</code></pre> |
| MoM's <code>\$usecp</code> parameter          | Tells MoM where to look for files in a shared file system; also tells MoM that she can use the local copy agent for these files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Job's <code>sandbox</code> attribute          | Determines whether PBS creates staging and execution directories for this job. If value is <i>PRIVATE</i> , PBS creates directories under the location specified in the MoM <code>\$jobdir_root</code> configuration option or in the submitter's home directory. If value is <i>HOME</i> or is unset, PBS uses the user's home directory for staging and execution. User-settable per-job via <code>qsub -W</code> or through a PBS directive.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Job's <code>stagein</code> attribute          | <p>Sets list of files or directories to be staged in. User-settable per job via <code>qsub -W</code>. Format:</p> <pre><code>execution_path@storage_host:storage_path</code></pre> <p>The <i>execution_path</i> is the path to the staging and execution directory. On stagein, <i>storage_path</i> is the path where the input files normally reside.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

Table 10-7: Options, Attributes, Environment Variables, etc., Affecting Staging

| Option, Attribute, Environment Variable, etc. | Effect                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Job's stageout attribute                      | Sets list of files or directories to be staged out. User-settable per job via <code>qsub -W</code> .<br>Format:<br><i>execution_path@storage_host:storage_path</i><br>The <i>execution_path</i> is the path to the staging and execution directory. On stageout, <i>storage_path</i> is the path where output files will end up.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Job's jobdir attribute                        | Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via <code>qstat -f</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Job's Keep_Files attribute                    | Determines whether output and/or error files remain on execution host. User-settable per job via <code>qsub -k</code> or through a PBS directive. If the <b>Keep_Files</b> attribute is set to <i>o</i> and/or <i>e</i> (output and/or error files remain in the staging and execution directory) and the job's <b>sandbox</b> attribute is set to <i>PRIVATE</i> , standard out and/or error files are removed when the staging and execution directory is removed at job end along with its contents. If direct write for files is specified via the <i>-d</i> suboption to the <i>-k</i> argument, files are not removed. See <a href="#">"Keeping Output and Error Files on Execution Host", on page 46 of the PBS Professional User's Guide</a> . |
| Remove_Files attribute                        | Specifies whether standard output and/or standard error files are automatically removed (deleted) upon job completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Job's PBS_JOBDIR environment variable         | Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, <code>pbs_tm</code> job tasks, prologue and epilogue, and MoM \$action scripts.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Job's TMPDIR environment variable             | Location of job-specific scratch directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| PBS_RCP string in <code>pbs.conf</code>       | Location of <code>rcp</code> command                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| PBS_SCP string in <code>pbs.conf</code>       | Location of <code>scp</code> command; setting this parameter causes PBS to first try <code>scp</code> rather than <code>rcp</code> for file transport.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Server's default_qsub_arguments attribute     | Can contain a default for job's <b>sandbox</b> (and other) attributes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

### 10.13.1.5 Getting Information About the Job Staging and Execution Directory

You can check the value of the job's `jobdir` attribute via `qstat` or the equivalent API while a job is executing. The value of `jobdir` is not retained if a job is rerun; it is undefined whether `jobdir` is visible or not when the job is not executing.

### 10.13.1.6 Staging and Execution Directory Caveats

- If the user home directory is NFS mounted, and you want to use `sandbox=PRIVATE`, then root must be allowed write privilege on the NFS filesystem on which the users' home directories reside.
- You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution. The pathname to each directory on each node may be different, since each depends on the corresponding MoM's `$jobdir_root` or user home directory.
- The directory specified in MoM's `$jobdir_root` parameter must have permissions set to `1777`.
- Beware shared staging directories:

When you set `$jobdir_root` to a shared (e.g. NFS) directory, tell MoM it is shared by setting the *shared* directive after the directory name:

```
$jobdir_root <directory name> shared
```

If the user home directory is shared, tell MoM it is shared:

```
$jobdir_root PBS_USER_HOME shared
```

Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when sister nodes are released, those sister MoMs would normally clean up their own files upon release, but this could cause problems in a shared directory. So if `$jobdir_root` or submitter home directories are shared, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

## 10.14 The Job Lifecycle

### 10.14.1 Sequence of Events for Start of Job

This is the order in which events take place on an execution host at the start of a job:

1. Application licenses are checked out
2. Any job-specific staging and execution directories are created:
  - `PBS_JOBDIR` and job's `jobdir` attribute are set to pathname of staging and execution directory
  - Files are staged in
 

PBS evaluates `execution_path` and `storage_path` relative to the staging and execution directory given in `PBS_JOBDIR`. PBS stages files to the primary execution host only. Staging is done as the job owner.

PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See [section 9.7, “Setting File Transfer Mechanism”, on page 441](#).
3. Temporary scratch directories (TMPDIRs) are created
 

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for this job. The root of TMPDIR is set by MoM to the value of MoM's `$tmpdir` configuration parameter. PBS sets TMPDIR to the pathname of the job-specific temporary scratch directory. This directory is for the use of the job, not PBS. This directory and its contents are removed when the job is finished.

The recommended TMPDIR configuration is to have a separate, local directory on each host. If the temporary scratch directory cannot be created, the job is killed.
4. The job's cpusets are created
5. The prologue is executed

The MoM's prologue is run on the primary host as root, with the current working directory set to `PBS_HOME/mom_priv` and with `PBS_JOBDIR` set in its environment.

6. The job script is executed

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job via the `pbs_tm( )` API as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in their environment. The job attribute `jobdir` is set to the pathname of the staging and execution directory on the primary host.

## 10.14.2 Sequence of Events for End of Job

This is the order in which events generally take place at the end of a job:

7. The job script finishes

8. The epilogue is run

PBS runs MoM's epilogue script on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` set in its environment.

9. The obit is sent to the server

10. Any specified file staging out takes place, including `stdout` and `stderr`

When PBS stages files out, it evaluates `execution_path` and `storage_path` relative to `PBS_JOBDIR`. Files that cannot be staged out are saved in `PBS_HOME/undelivered`. PBS stages files out from the primary execution host only. Staging is done as the job owner.

PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See [section 9.7, “Setting File Transfer Mechanism”, on page 441](#).

When the job is done, PBS writes the final job accounting record and purges job information from the server's database.

- If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. If no errors are encountered during stageout and all stageouts are successful, the staging and execution directory and all of its contents are removed, on all execution hosts.
- Files to be staged out are deleted all together, only after successful stageout of all files. If any errors are encountered during stageout, no files are deleted on the primary execution host, and the execution directory is not removed.
- If PBS created job-specific staging and execution directories on secondary execution hosts, those directories and their contents are removed at the end of the job, regardless of stageout errors. If these directories are shared, only the MoM on the primary execution host does the cleanup. If the directories are not shared, those directories are cleaned up by the sister MoMs.
- If PBS did not create job-specific staging and execution directories, files that are successfully staged out are deleted immediately, without regard to files that were not successfully staged out.

11. Files staged in or out are removed

The job's `stdout` and `stderr` files are created directly in the job's staging and execution directory on the primary execution host. See ["Managing Output and Error Files", on page 42 of the PBS Professional User's Guide](#).

- If PBS creates job-specific staging and execution directories and the submitter uses `qsub -k` without the `-d` sub-option (direct write to final destination), the `stdout` and `stderr` files are **not** automatically copied out of the staging and execution directory at job end; they are deleted when the directory is automatically removed.
- If PBS does not create job-specific staging and execution directories and the submitter uses `qsub -k`, standard out and/or standard error files are retained in the submitter's home directory on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

12. PBS removes all temporary scratch directories (TMPDIRs), along with their contents.

13. Any job-specific staging and execution directories are removed

14. Job files are deleted

15. Application licenses are returned to pool

16. The job's cpusets are destroyed

## 10.15 Managing Job History

### 10.15.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

You can configure whether PBS preserves job history, and for how long, by setting values for the `job_history_enable` and `job_history_duration` server attributes.

### 10.15.2 Definitions

#### Moved jobs

Jobs which were moved to another server

#### Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

#### History jobs

Jobs which will no longer execute at this server:

- Moved jobs
- Finished jobs



---

### 10.15.3 Job History Information Preserved by PBS

PBS can keep all job attribute information, including the following kinds of job history information:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that are running at another server
- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

### 10.15.4 Period When PBS Preserves Job History

PBS preserves history for the specified history duration beginning from the time a job finishes or is deleted.

After the duration has expired, PBS deletes the job history information and it is no longer available.

### 10.15.5 Configuring Job History Management

To configure job history, you enable it and you set the job history duration. You configure PBS to manage job history using the following server attributes:

#### job\_history\_enable

Enables or disables job history management. Setting this attribute to *True* enables job history management.

Format: Boolean.

Default: *False*

#### job\_history\_duration

Specifies the length of time that PBS will keep each job's history.

Format: duration: [[hours:]minutes:]seconds[.milliseconds]

Default: *Two weeks* (336:00:00)

#### 10.15.5.1 Enabling Job History

To enable job history management, set the server's `job_history_enable` attribute to *True*:

```
Qmgr: set server job_history_enable=True
```

#### 10.15.5.2 Setting Job History Duration

To set the length of time that job history is preserved, set the server's `job_history_duration` attribute to the desired duration:

```
Qmgr: set server job_history_duration=<duration>
```

If the job history duration is set to *zero*, no history is preserved.

If job history is enabled and job history duration is unset, job history information is kept for the default 2 weeks.

## 10.15.6 Changing Job History Settings

### 10.15.6.1 Disabling Job History

If job history is being preserved, and you unset the `job_history_enable` server attribute, PBS deletes all job history information. This information is no longer available.

### 10.15.6.2 Enabling Job History

If job history is not being preserved, and you set the `job_history_enable` server attribute, PBS begins preserving job history information for any jobs that are queued or running.

### 10.15.6.3 Modifying Job History Duration

Every job's history duration is set to the current value of the `job_history_duration` server attribute.

Example 10-5: Reducing job history duration:

The value of `job_history_duration` was "00:10:00" when a job finished execution. After 2 minutes, you change the duration to "00:06:00". This job's history is kept for a total of 6 minutes.

Example 10-6: Increasing job history duration:

The value of `job_history_duration` was "00:10:00" when a job finished execution. After 8 minutes you change the duration to "00:30:00". This job's history is kept for a total of 30 minutes.

Example 10-7: Increasing job history duration:

The value of `job_history_duration` was "00:10:00" when a job finished execution. After 11 minutes you change the duration to "00:30:00". This job's history is kept for a total of 10 minutes. The job's history is deleted after it is kept for 10 minutes.

## 10.15.7 Backward Compatibility

To have PBS behave as it did before the job history management feature was introduced, disable job history management. Do one of the following:

- Set the server's `job_history_enable` attribute to *False*:  
`Qmgr: set server job_history_enable=False`
- Unset the server's `job_history_enable` attribute:  
`Qmgr: unset server job_history_enable`
- Set the value of the server's `job_history_duration` attribute to *zero*, by doing one of the following:  
`Qmgr: set server job_history_duration=0`  
`Qmgr: set server job_history_duration=00:00`  
`Qmgr: set server job_history_duration=00:00:00`

## 10.15.8 Logging Moved Jobs

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the `qmove` command
- Job was submitted to a routing queue, then routed to a destination queue at another server

When a job is moved, the server logs the event in the server log and the accounting log. The server log messages are logged at log level 0x0008.

Format for the server log file:

```
7/08/2008 16:17:38;0008;Server@serverhost1;Job; 97.serverhost1.domain.com;Job moved to
destination: workq@serverhost2
```

Format for the accounting log entry:

```
7/08/2008 16:17:38;M;97.serverhost1.domain.com;destination=workq@serverhost2
```

Record type: *M* (moved job)

## 10.15.9 Deleting Moved Jobs and Job Histories

You can use the `qdel -x` option to delete job histories. This option also deletes any specified jobs that are queued, running, held, suspended, finished, or moved. When you use this, you are deleting the job and its history in one step. If you use the `qdel` command without the `-x` option, you delete the job, but not the job history, and you cannot delete a moved or finished job. See [“qdel” on page 143 of the PBS Professional Reference Guide](#).

### 10.15.10 Job History Caveats

- Enabling job history requires additional memory for the server. When the server is keeping job history, it needs 8kb-12kb of memory per job, instead of the 5kb it needs without job history. Make sure you have enough memory: multiply the number of jobs being tracked by this much memory. For example, if you are starting 100 jobs per day, and tracking history for two weeks, you're tracking 1400 jobs at a time. On average, this will require 14.3M of memory.
- If the server is shut down abruptly, there is no loss of job information. However, the server will require longer to start up when keeping job history, because it must read in more information.

## 10.16 Environment Variables

The settings in `$PBS_HOME/pbs_environment` are available to user job scripts. You must HUP the MoM if you change the file. This file is useful for setting environment variables for `mpirun` etc. For a list of environment variables used by PBS, see [“PBS Environment Variables” on page 397 of the PBS Professional Reference Guide](#).

## 10.17 Adjusting Job Running Time

### 10.17.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the `min_walltime` and `max_walltime` resources. PBS chooses the actual `walltime`. Any job that requests `min_walltime` is a **shrink-to-fit** job.

For a complete description of using shrink-to-fit jobs, see [section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211](#).

## 10.18 Managing Number of Run Attempts

PBS has a built-in limit of 21 for the number of times the server can try to run a job or subjob. When the job or subjob goes over this limit, it gets a System hold. The number of tries is recorded in the job or subjob's `run_count` attribute. The `run_count` attribute starts at zero, and the job or subjob is held when `run_count` goes above 20. When a subjob's `run_count` attribute goes above 20, it and its parent job array get a System hold. You can use `qrls` on the parent array to release the parent array and indirectly release the subjobs. See [“qrls” on page 183 of the PBS Professional Reference Guide](#).

Job submitters can set a non-negative value for `run_count` on job submission, and can use `qalter` to raise the value of `run_count`. A PBS Manager or Operator can use `qalter` to raise or lower the value of `run_count`.

## 10.19 Managing Amount of Memory for Job Scripts

By default, starting with version 13.1, PBS limits the size of any single job script to 100MB. You can set a different limit using the `jobscript_max_size` server attribute. The format for this attribute is *size*, and the units default to bytes. You can specify the units. For example:

```
Qmgr: set server jobscript_max_size = 10mb
```

Job script size affects server memory footprint. If a job submitter wants to use a really big script, they can put it in shared storage and call it from a short script, or they can run a small job script that stages in the big script, then calls it.

## 10.20 Allowing Interactive Jobs on Windows

1. Make sure that file and printer sharing is enabled. This is off by default.
2. Make sure that the ephemeral port range in your firewall is open on both the submission and execution hosts. Check your OS documentation for the correct range.
3. Make sure that IPC\$ share is enabled. You should be able to run the following command from the submission host:

```
net use \\<execution_host>\IPC$
```

The output should look like this:

```
> net use \\myhost\IPC$
c:\Users\pbsuser>net use \\myhost\IPC$
Local name
Remote name \\myhost\IPC$
Resource type IPC
Status Disconnected
Opens 0
Connections 1
```

The command completed successfully.

### 10.20.1 Configuring PBS for Remote Viewer on Windows

Job submitters can run interactive GUI jobs so that the GUI is connected to the primary execution host for the job. The job submitter runs a GUI application over a remote viewer. On Windows, PBS supports any remote viewer, such as Remote Desktop or X.

You can specify the remote viewer that PBS will use by setting a `pbs.conf` parameter on each submission host. See [section 10.20.2, “Specifying Remote Viewer at Submission Hosts”, on page 485](#).

On an execution host that will launch a GUI application for an interactive job, MoM must run in a `LocalSystem` account. See [section 10.20.3, “Configuring MoM to Run in LocalSystem Account on Windows”, on page 485](#).

A password is usually required when a Remote Desktop client tries to connect to an execution host. However you can configure Single Sign-on for Remote Desktop using the current login at the client host. See [section 10.20.4, “Configuring Single Sign-on for Remote Desktop on Windows”, on page 486](#).

## 10.20.2 Specifying Remote Viewer at Submission Hosts

You can specify which remote viewer PBS should use when a job submitter runs a GUI job remotely. On each submission host, set the `PBS_REMOTE_VIEWER` parameter in `pbs.conf` to point to the remote viewer you want, or to a script that launches the desired remote viewer. If this parameter is unset, PBS uses the native Windows Remote Desktop client as the remote viewer. The line in `pbs.conf` should have this form:

```
PBS_REMOTE_VIEWER = <remote viewer client>
```

Example 10-8: Using the remote desktop client as the remote viewer:

```
PBS_REMOTE_VIEWER=mstsc /v
```

Example 10-9: Using the VNC viewer client as the remote viewer:

```
PBS_REMOTE_VIEWER=vncviewer.exe
```

Example 10-10: Launching a remote viewer via a script:

```
PBS_REMOTE_VIEWER=launch_remote_viewer.bat
```

## 10.20.3 Configuring MoM to Run in LocalSystem Account on Windows

On an execution host that will launch a GUI application for an interactive job, MoM must run in a `LocalSystem` account. To run MoM in a `LocalSystem` account, take the following steps:

1. Log in as administrator
2. Open `services.msc`
3. Right-click on the `pbs_mom` service and open "*properties*"
4. In the "*Log on*" tab, select "*Local System Account*"
5. Check "*Allow service to interact with desktop*"
6. Click *OK*

---

## 10.20.4 Configuring Single Sign-on for Remote Desktop on Windows

### 10.20.4.1 Configuring Submission Hosts for Single Sign-on

You can configure single sign-on using domain or local group policy. Follow these steps:

1. Log on to your local machine as an administrator
2. Start the Group Policy Editor:  
`gpedit.msc`
3. Navigate to "*Computer Configuration\Administrative Templates\System\Credentials Delegation*"
4. Double-click the "*Allow Delegating Default Credentials*" policy
5. Enable the policy
6. Click on the "*Show*" button to get to the list of servers
7. Add "TERMSRV/<server name>" to the server list.
8. You can add any number of server names to the list. A server name can be a hostname or an IP address. You can use one wildcard (\*) per name. To store credentials for everything, use just a wildcard.
9. Confirm your changes by clicking on the "*OK*" button until you get back to the main Group Policy Object Editor dialog.
10. Repeat steps 3 through 7 for the following policies:
  - a. "*Allow Delegating Default Credentials with NTLM-only Server Authentication*"
  - b. "*Allow Delegating Saved Credentials with NTLM-only Server Authentication*"
  - c. "*Allow Delegating Saved Credentials*"
11. In the Group Policy editor, navigate to *Computer Configuration -> Administrative Templates -> Windows Components -> Remote Desktop Services -> Remote Desktop Connection Client*
12. For the entry labeled "*Do not allow passwords to be saved*", change to *Disabled*
13. Force the policy to be refreshed immediately on the local machine. Run the following at a command prompt:  
`gpupdate /force`

### 10.20.4.2 Configuring Execution Hosts for Single Sign-on

The PBS execution host is the Remote Desktop server.

If the execution host is a Windows server, for example Windows Server 2008 R2, follow these steps:

1. Start Server Manager
2. Expand *Roles->Remote Desktop Services* and select *RD Session Host Configuration*
3. In the right pane in *Connections*, right-click *RDP-TCP Connection Name* and choose *Properties*
4. On the *Log on Settings* tab make sure "*Always prompt for password*" is unchecked
5. On the *General* tab choose the *Security layer: Negotiate* or *SSL (TLS 1.0)*
6. Click *OK*

If the execution host is not a Windows server, follow these steps:

1. Open the Group Policy Editor:  
`gpedit.msc`
2. Navigate to *Computer Configuration->Administrative Templates->Windows Components->Remote Desktop Services->Remote Desktop Session Host->Security*
3. Set "Always prompt for password upon connection" to "Disabled"

## 10.21 Releasing Unneeded Vnodes from Jobs

If you want to prevent unnecessary resource usage, you can release unneeded hosts or vnodes from jobs. You can use the `pbs_release_nodes` command or the `release_nodes_on_stageout` job attribute:

- You can use the `pbs_release_nodes` command at the command line, or submitters can use it or in their job scripts to release vnodes when the command is issued. You can use this command to release specific vnodes that are not on the primary execution host, or all vnodes that are not on the primary execution host. You can also use it to release all hosts or vnodes except for what you specify, which can be either a count of hosts to keep, or a select specification describing the vnodes to keep. You cannot use the command to release vnodes on the primary execution host. See [“pbs\\_release\\_nodes” on page 92 of the PBS Professional Reference Guide](#).
- You can set the job's `release_nodes_on_stageout` attribute to `True` so that PBS releases all of the job's vnodes that are not on the primary execution host when stageout begins. You must set the job's `stageout` attribute as well. See [“Job Attributes” on page 328 of the PBS Professional Reference Guide](#).
- You can use the `default_qsub_arguments` server attribute to specify that all jobs are submitted with `release_nodes_on_stageout` set by default.

For details, see [“Releasing Unneeded Vnodes from Your Job”, on page 129 of the PBS Professional User’s Guide](#).

### 10.21.1 Caveats and Restrictions for Releasing Vnodes

- The job must specify a stageout parameter in order to release vnodes on stageout. If the job does not specify stageout, `release_nodes_on_stageout` has no effect.
- You can release only vnodes that are not on the primary execution host. You cannot release vnodes on the primary execution host.
- The job must be running (in the *R* state).
- If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.
- If a vnode on a multi-vnode host is assigned exclusively to a job, and the vnode is released, the job will show that the vnode is released, but the vnode will still show as assigned to the job in `pbsnodes -av` until the other vnodes on that host have been released. If a vnode on a multi-vnode machine is not assigned exclusively to a job, and the vnode is released, it shows as released whether or not the other vnodes on that host are released.
- If you specify release of a vnode on which a job process is running, that process is terminated when the vnode is released.

## 10.22 Tolerating Vnode Faults

PBS lets you allocate extra vnodes to a job so that the job can successfully start and run even if some vnodes fail. See [section 8.5, “Vnode Fault Tolerance for Job Start and Run”, on page 403](#).



## 10.23 Managing Job Array Behavior

You can set a limit on the number of simultaneously running subjobs from any one array job by setting the job's `max_run_subjobs` attribute, either via `qalter -Wmax_run_subjobs=<new value> <job ID>`, or in a `queuejob` or `modifyjob` hook.

You can set a limit on the size of job arrays, either at the server, via the server `max_array_size` attribute, or at each queue, via the server `max_array_size` attribute.

Note that the `qrun` command overrides the limit on the number of simultaneously running subjobs for an array job set in the `max_run_subjobs` job attribute.

You can specify whether the `max_queued` limit attribute counts each array job as a single job, or counts each subjob as a single job via the `elim_on_subjobs` server attribute. Set this to *True* to have each subjob count as a single job.

## 10.24 Recommendations

We recommend that as much as possible, you avoid huge numbers of jobs and subjobs. We recommend consolidating jobs where possible.

# 11

## Security

This chapter describes the security features of PBS. These instructions are for the PBS administrator and Manager.

### 11.1 Configurable Features

This section gives an overview of the configurable security mechanisms provided by PBS, and gives links to information on how to configure each mechanism.

The following table lists configurable PBS security mechanisms and their configuration procedures.

**Table 11-1: Security Mechanisms and their Configuration Procedures**

| Security Mechanism                                                     | Configuration Procedure                                              |
|------------------------------------------------------------------------|----------------------------------------------------------------------|
| Authentication with daemons and users                                  | <a href="#">"Authentication for Daemons &amp; Users" on page 508</a> |
| Encrypting communication                                               | <a href="#">"Encrypting PBS Communication" on page 517</a>           |
| Access control for server, queues, reservations                        | <a href="#">"Using Access Control Lists" on page 492</a>             |
| Event logging for server, scheduler, MoMs                              | <a href="#">"Event Logging" on page 429</a>                          |
| File copy mechanism                                                    | <a href="#">"Setting File Transfer Mechanism" on page 441</a>        |
| Levels of privilege (user roles)                                       | <a href="#">"User Roles and Required Privilege" on page 489</a>      |
| Restricting access to execution hosts via <code>\$restrict_user</code> | <a href="#">"Restricting Execution Host Access" on page 521</a>      |

### 11.2 User Roles and Required Privilege

#### 11.2.1 Root Privilege

Root privilege is required to perform some operations in PBS involving writing to the server's private, protected data. Root privilege is required in order to do the following:

- Create hooks
- Alter MoM and scheduler configuration files
- Set scheduler priority formula
- Run certain commands, including the following:
  - `pbs_probe`
  - `pbs_mom`
  - `pbs_sched`
  - `pbs_server`
  - `pbsfs`
- Use the `tracejob` command to view accounting log information

---

There are some operations that root privilege alone does not allow. These operations require Manager privilege but not root privilege. Manager privilege, but not root privilege, is required in order to do the following:

- Set attributes
- Create or delete vnodes using the `qmgr` command

## 11.2.2 User Roles

PBS allows certain privileges based on what role a person has, and whether that person has root privilege. PBS recognizes only three roles, and all those using PBS must be assigned one of these roles. These roles are *Manager*, *Operator*, and *user*. Roles are assigned by PBS Managers only. No roles can be added, and roles cannot be modified; the function of roles is hardcoded in the server.

In addition to these roles, PBS requires a [PBS Administrator](#) to perform some downloading, installation, upgrading, configuration, and management functions. PBS does not recognize *PBS Administrator* as a PBS role; this term is used in PBS documentation to mean the person who performs these tasks.

PBS roles and PBS Administrators are described in the following sections:

### 11.2.2.1 User

#### 11.2.2.1.i Definition of User

Users are those who submit jobs to PBS.

Users have the lowest level of privilege. Users are referred to in the PBS documentation as "users". By default, users may operate only on their own jobs. They can do the following:

- Submit jobs
- Alter, delete, and hold their own jobs
- Status their own jobs, and those of others if permission has been given via the [query\\_other\\_jobs](#) server attribute. The `query_other_jobs` server attribute controls whether unprivileged users are allowed to select or query the status of jobs owned by other users.
- List and print some but not all server, queue, vnode, scheduler, and reservation attributes

#### 11.2.2.1.ii Defining List of Users

PBS allows you to define a list of users allowed or denied access to the PBS server, however this is done using the PBS access control list mechanism. Access control is described in [section 11.3, "Using Access Control Lists", on page 492](#).

### 11.2.2.2 Operator

#### 11.2.2.2.i Definition of Operator

A PBS Operator is a person who has an account that has been granted Operator privilege.

Operators have more privilege than users, and less privilege than Managers.

Operators can manage the non-security-related attributes of PBS such as setting and unsetting non-security attributes of vnodes, queues, and the server. Operators can also set queue ACLs.

---

Operators can do the following:

- All operations that users can perform
- Set non-security-related server, queue, and vnode attributes (Operators are not permitted to set server ACLs)
- Alter some job attributes
- Set or alter most resources on the server, queues, and vnodes
- Rerun, requeue, delete, and hold all jobs
- Run any command to act on a job

#### 11.2.2.2.ii Defining List of Operators

To define the list of Operators at a PBS complex, set the server's `operators` attribute to a list of usernames, where each username should be an Operator. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

It is important to grant Operator privilege to appropriate persons only, since Operators can control how user jobs run.

### 11.2.2.3 Manager

#### 11.2.2.3.i Definition of Manager

A Manager is a person who has an account that has been granted PBS Manager privilege.

Managers have more privilege than Operators. Managers can manage the security aspects of PBS such as server ACLs and assignment of User Roles.

Managers can do the following:

- All operations that Operators can perform
- Create or delete queues or vnodes
- Set all server, queue, and vnode attributes, including server ACLs

#### 11.2.2.3.ii Defining List of Managers

To define the list of Managers at a PBS complex, set the server's `managers` attribute to a list of usernames, where each username should be a Manager. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

If the server's `managers` attribute is not set or is unset, root on the server host is given Manager privilege.

It is important to grant Manager privilege to appropriate persons only, since Managers control much of PBS.

### 11.2.2.4 PBS Administrator

#### 11.2.2.4.i Definition of PBS Administrator

Linux: person with Manager privilege and root access.

Windows: person with Manager privilege who is a member of the local Administrators group.

A person who administers PBS, performing functions such as downloading, installing, upgrading, configuring, or managing PBS. PBS Administrators perform all the functions requiring root privilege, as described in [section 11.2.1, “Root Privilege”, on page 489](#).

*PBS Administrator* is distinguished from "site administrator", although often these are the same person.

---

## 11.3 Using Access Control Lists

### 11.3.1 Access Definitions

In this section we describe the meaning of access for each entity and object where the access of the entity to the object has an access control mechanism.

#### 11.3.1.1 Access to a PBS Object

Below are the definitions of what access to each of the following PBS objects means:

##### **Access to the server**

Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the server and queues.

##### **Access to a queue**

Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

##### **Access to a reservation**

Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.

#### 11.3.1.2 Access by a PBS Entity

Access can be granted at the server, queues, and reservations for each of the following entities:

##### **User access**

The specified user is allowed access.

##### **Group access**

A user in the specified group is allowed access

##### **Host access**

A user is allowed access from the specified host

### 11.3.2 Requirement for Access

In order to have access to a PBS object such as the server or a queue, a user must pass all enabled access control tests: the user must be allowed access, the user's group must be allowed access, and the host where the user is working must be allowed access.

In some cases, Manager or Operator privilege overrides access controls. For some kinds of access, there are no controls. See [section 11.3.10, “Operations Controlled by ACLs”, on page 504](#).

### 11.3.3 Managing Access via Lists

PBS uses access control lists (ACLs) to manage access to the server, queues, and reservations. There is a separate set of ACLs for the server, each queue, and each reservation. The server enforces the access control policy for User Roles supported by PBS. The policy is hardcoded within the server. ACLs can specify which entities are allowed access and which entities are denied access.

Each server and queue ACL can be individually enabled or disabled by a Manager. If an ACL is enabled, access is allowed or denied based on the contents of the ACL. If the ACL is disabled, access is allowed to all. The contents of each server or queue ACL can be set or altered by a Manager.

Reservation ACLs are enabled only by the reservation creator or the PBS Administrator. The server's `resv_enable` attribute controls whether reservations can be created. When this attribute is set to *False*, reservations cannot be created.

No default ACLs are shipped.

## 11.3.4 ACLs

An ACL, or Access Control List, is a list of zero or more entities (users, groups, or hosts from which users or groups may be attempting to gain access) allowed or denied access to parts of PBS such as the server, queues, or reservations. A server ACL applies to access to the server, and therefore all of PBS. A queue's ACL applies only to that particular queue. A reservation's ACL applies only to that particular reservation. The server, each queue, and each reservation has its own set of ACLs.

### 11.3.4.1 Format of ACLs

Entity access is controlled according to the list of entities allowed or denied access as specified in the object's `acl_<entity>` attribute. The object's access control attribute contains a list of entity names, where each entity name is marked with a plus sign ("+") if the entity is allowed access, and with a minus sign ("-") if the entity is denied access. For example, to allow `User1@host1.example.com`, and deny `User2@host1.example.com`:

```
+User1@host1.example.com, -User2@host1.example.com
```

### 11.3.4.2 Default ACL Behavior

If an entity name is included without either a plus or a minus sign, it is treated as if it has a plus sign, and allowed access.

If an entity name is not in the list, the default behavior is to deny access to the entity. Therefore, if the list is empty but enabled because the object's `acl_<entity>_enable` attribute is set to *True* (see [section 11.3.5, "Enabling Access Control", on page 495](#)), all entities are denied access.

### 11.3.4.3 Modifying ACL Behavior

You can specify how an ACL treats an unmatched entity by including special flags in the ACL itself. These are the plus and minus signs.

To allow access for all unmatched entities (the reverse of the default behavior), put a plus sign ("+") anywhere by itself in the list. For example:

```
+User1@host1.example.com, +, -User2@host1.example.com
```

To deny access for all unmatched entities (the default behavior), put a minus sign ("-") anywhere by itself in the list. For example:

```
+User1@host1.example.com, -, -User2@host1.example.com
```

If there are entries for both a plus and a minus sign, the last entry in the list (closest to the rightmost side of the list) will control the behavior of the ACL.

### 11.3.4.4 Contents of User ACLs

User ACLs contain a username and hostname combination. The subject's username and hostname combination is compared to the entries in the user ACL. Usernames take this form:

*User1@host.domain.com*

*User1@host.subdomain.domain.com*

Usernames can be wildcarded. See [section 11.3.4.7, “Wildcards In ACLs”, on page 494](#).

### 11.3.4.5 Contents of Group ACLs

Group ACLs contain names based on the user's groups, as defined by the operating system where the server is executing. All of the user's groups are included. The subject's group names on the server are compared to the entries in the Group ACL. Group names cannot be wildcarded.

### 11.3.4.6 Contents of Host ACLs

Host ACLs contain fully-qualified hostnames. The subject's host name is compared to the entries in the host ACL. To find the fully-qualified name of a host, use the `pbs_hostn` command. See [“pbs\\_hostn” on page 64 of the PBS Professional Reference Guide](#).

Hostnames can be wildcarded. See the following section.

### 11.3.4.7 Wildcards In ACLs

Usernames and hostnames can be wildcarded. The hostname portion of the username is wildcarded exactly the same way a hostname is wildcarded. The non-hostname portion of a username cannot be wildcarded.

The only character that can be used to wildcard entity names is the asterisk (“\*”). Wildcarding must follow these rules:

- The asterisk must be to the right of the at sign (“@”)
- There can be at most one asterisk per entity name
- The asterisk must be the leftmost label after the at sign

The following table shows how hostnames are wildcarded:

**Table 11-2: How Hostnames Are Wildcarded**

| Wildcard Use       | Meaning                                       |
|--------------------|-----------------------------------------------|
| *.test.example.com | Any host in the test subdomain in example.com |
| *.example.com      | Any host in example.com                       |
| *.com              | Any host in .com                              |
| *                  | Any host                                      |



The following examples show how wildcarding works in host ACLs:

Example 11-1: To limit host access to host myhost.test.example.com only:

```
myhost.test.example.com
```

Example 11-2: To limit host access to any host in the test.example.com subdomain:

```
*.test.example.com
```

Example 11-3: To limit host access to any host in example.com:

```
*.example.com
```

Example 11-4: To allow host access for all hosts:

```
*
```

The following examples show how wildcarding works in user ACLs:

Example 11-5: To limit user access to UserA requesting from host myhost.test.example.com only:

```
UserA@myhost.test.example.com
```

Example 11-6: To limit user access to UserA on any host in the test.example.com subdomain:

```
UserA@*.test.example.com
```

Example 11-7: To limit user access to UserA on any host in example.com:

```
UserA@*.example.com
```

Example 11-8: To limit user access to UserA from anywhere:

```
UserA@*
```

```
or
```

```
UserA
```

Listing a username without specifying the host or domain is the equivalent of listing the username followed by "@\*". This means that

```
User1
```

is the same as

```
User1@*
```

### 11.3.4.8 Restrictions on ACL Contents

All access control lists are traversed from left to right, and the first match found is used. It is important to make sure that entries appear in the correct order.

To single out a few, specify those few first, to the left of the other entries.

Example 11-9: To allow all users in your domain except User1 access, the list should look like this:

```
-User1@example.com, +*@example.com
```

Example 11-10: To deny access to all users in your domain except User1, the list should look like this:

```
+User1@example.com, -*@example.com
```

## 11.3.5 Enabling Access Control

Each server and queue ACL is controlled by a Boolean switch whose default value is *False*, meaning that access control is turned off. When access control is turned off, all entities have access to the server and to each queue. When access control is turned on, access is allowed only to those entities specifically granted access.

To use access control, first set the contents of the ACL, then enable it by setting its switch to *True*.

Reservation ACLs are enabled when the reservation creator sets their contents. Reservation ACLs do not have switches. Reservations use queues, which are regular queues whose ACL values have been copied from the reservation. These queues are not intended to be operated on directly. See [section 11.3.8, “Reservation Access”, on page 501](#).

### 11.3.5.1 Table of ACLs and Switches

The following table lists the ACLs and their switches, with defaults, for the server, queues, and reservations.

**Table 11-3: ACLs and Their Switches**

|                   |        | User<br>(Default Value)             | Group<br>(Default Value)                    | Host<br>(Default Value)                 |
|-------------------|--------|-------------------------------------|---------------------------------------------|-----------------------------------------|
| Server            | Switch | acl_user_enable<br>( <i>False</i> ) | None                                        | acl_host_enable<br>( <i>False</i> )     |
|                   | List   | acl_users<br>(all users allowed)    | None                                        | acl_hosts<br>(all hosts allowed)        |
| Queue             | Switch | acl_user_enable<br>( <i>False</i> ) | acl_group_enable<br>( <i>False</i> )        | acl_host_enable<br>( <i>False</i> )     |
|                   | List   | acl_users<br>(all users allowed)    | acl_groups<br>(all groups allowed)          | acl_hosts<br>(all hosts allowed)        |
| Reservation       | Switch | None                                | None                                        | None                                    |
|                   | List   | Authorized_Users<br>(creator only)  | Authorized_Groups<br>(creator's group only) | Authorized_Hosts<br>(all hosts allowed) |
| Reservation queue | Switch | acl_user_enable<br>( <i>True</i> )  | acl_group_enable<br>( <i>False</i> )        | acl_host_enable<br>( <i>False</i> )     |
|                   | List   | acl_users<br>(creator only)         | acl_groups<br>(all groups allowed)          | acl_hosts<br>(all hosts allowed)        |

### 11.3.6 Creating and Modifying ACLs

Server and queue ACLs follow the same rules for creation and modification. Reservation queue ACLs behave the same way regular queue ACLs do. Reservation ACLs can only be created and modified by the reservation creator and the administrator. See [section 11.3.8, “Reservation Access”, on page 501](#).

### 11.3.6.1 Rules for Creating and Modifying Server and Queue ACLs

- Server and queue ACLs are created and modified using the `qmgr` command.
- An ACL is a list of entries. When you operate on the list, the first match found, searching from left to right, is used. If there is more than one match for the entity you wish to control, ensure that the first match gives the behavior you want.
- When you create or add to an ACL, you can use the `+` or `-` operators to specify whether or not an entity is allowed access. Omitting the operator is equivalent to adding a `+` operator.
- When you re-create an existing ACL, this is equivalent to unsetting the old ACL and creating a new one.
- When you add to an ACL, the new entry is appended to the end of the ACL, on the right-hand side.
- When you remove an entity from an ACL, you cannot use `+` or `-` operators to specify which entity to remove, even if there are multiple entries for an entity and each entry has a different operator preceding it, for example `"-bob, +bob"`.
- When you remove an entity, only the first match found is removed.

### 11.3.6.2 Examples of Creating and Modifying Server and Queue ACLs

The following examples show the server's user ACL being set. Queue ACLs work the same way as server ACLs, and the equivalent `qmgr` command can be used for queues. So, where we use the following for the server:

```
Qmgr: set server acl_users ...
```

the same effect can be achieved at the queue using this:

```
Qmgr: set queue <queue name> acl_users ...
```

If the queue name is `Q1`, the `qmgr` command looks like this:

```
Qmgr: set queue Q1 acl_users ...
```

Example 11-11: To create a server or queue ACL:

```
Qmgr: set <object> <ACL> = <entity list>
```

Example:

```
Qmgr: set server acl_users = "-User1@*.example.com,+User2@*.example.com"
```

ACL looks like this:

```
-User1@*.example.com, +User2@*.example.com
```

Example 11-12: To add to a server or queue ACL:

```
Qmgr: set <object> <ACL> += <entity list>
```

Example:

```
Qmgr: set server acl_users += -User3@*.example.com
```

ACL looks like this:

```
-User1@*.example.com, +User2@*.example.com, -User3@*.example.com
```

Example 11-13: To remove an entry from an ACL:

```
Qmgr: set <object> <ACL> -= <entity>
```

Example:

```
Qmgr: set server acl_users -= User2@*.example.com
```

ACL looks like this:

```
-User1@*.example.com, -User3@*.example.com
```

Example 11-14: To remove two entries for the same entity from an ACL:

```
Qmgr: set <object> <ACL> -= <entity1, entity1>
```

Example: If ACL contains +A, +B, -C, -A, +D, +A

```
Qmgr: set server acl_users -= "A, A"
```

ACL looks like this:

```
+B, -C, +D, +A
```

Example 11-15: To remove multiple entities from an ACL:

```
Qmgr: set <object> <ACL> -= <entity list>
```

Example: If ACL contains +B, -C, +D, +A

```
Qmgr: set server acl_users -= "B, D"
```

ACL looks like this:

```
-C, +A
```

### 11.3.6.3 Who Can Create, Modify, Enable, or Disable ACLs

The following table summarizes who can create, modify, enable, or disable ACLs and their associated switches:

**Table 11-4: Who Can Create, Modify, Enable, Disable ACLs**

| ACLs and Switches        |         | Manager                     | Operator                    | User                      |
|--------------------------|---------|-----------------------------|-----------------------------|---------------------------|
| Server ACLs and Switches | Create  | Yes                         | No                          | No                        |
|                          | Modify  | Yes                         | No                          | No                        |
|                          | Enable  | Yes                         | No                          | No                        |
|                          | Disable | Yes                         | No                          | No                        |
| Queue ACLs and Switches  | Create  | Yes                         | Yes                         | No                        |
|                          | Modify  | Yes                         | Yes                         | No                        |
|                          | Enable  | Yes                         | Yes                         | No                        |
|                          | Disable | Yes                         | Yes                         | No                        |
| Reservation ACLs         | Create  | Only if reservation creator | Only if reservation creator | When creating reservation |
|                          | Modify  | Yes, if administrator       | No                          | Yes                       |
|                          | Enable  | Creator and administrator   | No                          | When creating reservation |
|                          | Disable | No                          | No                          | No                        |

**Table 11-4: Who Can Create, Modify, Enable, Disable ACLs**

| ACLs and Switches                   |         | Manager | Operator | User                                                                                                                        |
|-------------------------------------|---------|---------|----------|-----------------------------------------------------------------------------------------------------------------------------|
| Reservation Queue ACLs and Switches | Create  | Yes     | Yes      | Indirectly when creating reservation                                                                                        |
|                                     | Modify  | Yes     | Yes      | No                                                                                                                          |
|                                     | Enable  | Yes     | Yes      | Indirectly when creating reservation                                                                                        |
|                                     | Disable | Yes     | Yes      | Group and host ACLs can be indirectly disabled by user during reservation creation.<br>User ACL cannot be disabled by user. |

### 11.3.6.4 Who Can Operate on Server ACLs

PBS Managers only can create or modify server ACLs and the Boolean switches that enable them.

### 11.3.6.5 Who Can Operate on Queue ACLs

PBS Managers and Operators, but not users, can create and modify queue ACLs and their Boolean switches.

### 11.3.6.6 Who Can Operate on Reservation ACLs

When creating a reservation, the reservation creator cannot disable the user ACL, but can choose to enable or disable the group and host ACLs implicitly via the command line, and can specify the contents of all three ACLs. Reservation ACLs can be modified via [pbs\\_ralter](#) or disabled.

### 11.3.6.7 Who Can Operate on Reservation Queue ACLs

Unprivileged users cannot directly create, modify, enable, or disable reservation queue ACLs or the associated switches. The reservation creator can indirectly create and enable the reservation queue's ACLs during reservation creation. If a user wants to modify a reservation queue's ACLs, they can do so indirectly by deleting the reservation and creating a new one with the desired ACLs.

PBS Managers and Operators can modify, enable, or disable a reservation queue's ACLs.

A reservation queue's user ACL is always enabled unless explicitly disabled after creation by a Manager or Operator.

## 11.3.7 Server and Queue ACLs

Access control for an entity such as a user, group, or host is enabled by setting the attribute enabling that entity's ACL to *True*. When this attribute is *True*, entity access is controlled according to the list of entities allowed or denied access as specified in the ACL for that entity. The default value for each ACL's switch attribute is *False*, meaning that entity access is not controlled.

### 11.3.7.1 Server ACLs

The server has a host ACL and a user ACL.

Server access is controlled by these attributes:

- User access: `acl_user_enable` and `acl_users`
- Host access: `acl_host_enable` and `acl_hosts`

### 11.3.7.2 Queue ACLs

Each queue has three ACLs: a host ACL, a user ACL, and a group ACL.

Queue access is controlled by these attributes:

- User access: `acl_user_enable` and `acl_users`
- Group access (queue only): `acl_group_enable` and `acl_groups`
- Host access: `acl_host_enable` and `acl_hosts`

### 11.3.7.3 Access to Server for MoMs

You can specify whether all MoMs should have the same privilege when contacting the server as hosts listed in the `acl_hosts` server attribute using the `acl_host_moms_enable` server attribute. If you set this to *True*, all MoMs are allowed privileged access to the server, and you don't need to explicitly add their hosts to the ACL. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#).

### 11.3.7.4 Examples of Setting Server and Queue Access

To restrict access to the server or queue, first set the contents of the ACL, then enable the ACL by setting its switch to *True*.

Example 11-16: To allow server access for all users in your domain except User1, and to allow server access for User2 in another domain:

Set the server's `acl_users` attribute:

```
Qmgr: set server acl_users = "-User1@example.com, +*@example.com,
+User2@otherdomain.com"
```

Enable user access control by setting the server's `acl_user_enable` attribute to *True*:

```
Qmgr: set server acl_user_enable = True
```

Example 11-17: To require that users of a queue be in Group1 only:

Set the queue's `acl_groups` attribute:

```
Qmgr: set queue Queue1 acl_groups = +Group1
```

Enable group access control by setting the queue's `acl_group_enable` attribute to *True*:

```
Qmgr: set queue Queue1 acl_group_enable = True
```

Example 11-18: To allow access to Queue1 from Host1 only:

Set the queue's `acl_hosts` attribute:

```
Qmgr: set q Queue1 acl_hosts = +Host1@example.com
```

Enable host access control by setting the queue's `acl_host_enable` attribute to *True*:

```
Qmgr: set q Queue1 acl_host_enable = True
```

## 11.3.8 Reservation Access

Advance, job-specific, and standing reservations are intended to be created by job submitters, although managers and operators can create them as well. Maintenance reservations can be created only by managers and operators. The administrator controls whether reservations can be created via the server's `resv_enable` attribute. When this attribute is set to *True*, reservations can be created.

Reservation ACLs allow or deny access based on group names, usernames, and hostnames. Each reservation has its own access control attributes that can be used to specify which users and groups have access to the reservation, and the hosts from which these users and groups are allowed access. The creator of the reservation sets the lists of users, groups and hosts that have access to the reservation (the reservation ACLs). This is done while creating the reservation, using options to the `pbs_rsub` command.

When you create a reservation ACL, it is automatically enabled; you do not have to explicitly enable it. The reservation's list of authorized users is always enabled during reservation creation. The reservation's lists of authorized groups and authorized hosts are only enabled if explicitly set by the reservation creator. PBS checks for membership in authorized lists only when that ACL is enabled. So for example, if you create a reservation and do not specify a list of authorized groups, no groups are added to the reservation's ACL, but you can submit jobs to the reservation because PBS does not check for group membership.

While you will see that each reservation has its own queue, do not attempt to manipulate reservation queue attributes directly. You operate on the reservation attributes, and PBS manages the queue's attributes, making them mirror those of the reservation. Set or modify reservation attributes using [pbs\\_rsub](#) and [pbs\\_ralter](#).

### 11.3.8.1 Meaning of Reservation Access

Access to a reservation via the reservation's ACLs is required for the following actions:

- Submitting a job into the reservation
- Moving a job into the reservation

A job owner can perform the following actions on their own jobs, regardless of ACLs:

- Delete their job
- Hold their job
- Move their job out of the reservation

For example, if an Operator `qmoves` User1's job into a reservation to which User1 is denied access, User1 can still perform operations on the job such as deleting or holding the job, and User1 can `qmove` the job out of the reservation.

### 11.3.8.2 Reservation Access Attributes

Reservation access is controlled by the following reservation attributes:

- User access: `Authorized_Users`
  - Default: the reservation creator only is allowed access
  - This ACL is always enabled
- Group access: `Authorized_Groups`
  - Default: no groups are allowed access
  - This ACL is enabled only when you specify a list of groups
- Host access: `Authorized_Hosts`
  - Default: all hosts are allowed access
  - This ACL is enabled only when you specify a list of hosts



### 11.3.8.3 Setting and Changing Reservation Access

The reservation creator uses options to the [pbs\\_rsub](#) command to set reservation access attributes:

#### -U <authorized user list>

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's `Authorized_Users` attribute to *auth user list*.

This list becomes the `acl_users` attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the `Authorized_Users` and `Authorized_Groups` reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the `Authorized_Users` reservation attribute in [section 6.8, "Reservation Attributes", on page 304](#).

Syntax:

```
[+|-]<username>[@<hostname>][,+|-]<username>[@<hostname>]...
```

Default: Job owner only

#### -G <authorized group list>

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's `Authorized_Groups` attribute to *auth group list*.

This list becomes the `acl_groups` list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the `Authorized_Users` and `Authorized_Groups` reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the `Authorized_Groups` reservation attribute in [section 6.8, "Reservation Attributes", on page 304](#).

Syntax:

```
[+|-]<group name>[,+|-]<group name> ...]
```

Default: No groups are authorized to submit jobs

#### -H <authorized host list>

Comma-separated list of hosts from which jobs can and cannot be submitted to this reservation. This list becomes the `acl_hosts` list for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. If the reservation creator specifies this list, the creator's host is not automatically added to the list.

See the `Authorized_Hosts` reservation attribute in [section 6.8, "Reservation Attributes", on page 304](#).

Format: `[+|-]<hostname>[,+|-]<hostname> ...]`

Default: All hosts are authorized to submit jobs

Use the [pbs\\_ralter](#) command to modify existing advance, job-specific, or standing reservations:

**-U <authorized user list>**

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's `Authorized_Users` attribute to *auth user list*.

This list becomes the `acl_users` attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the `Authorized_Users` and `Authorized_Groups` reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the `Authorized_Users` reservation attribute in [section 6.8, “Reservation Attributes”, on page 304](#).

Syntax:

```
[+|-]<username>[@<hostname>][,+|-]<username>[@<hostname>]...
```

**-G <authorized group list>**

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's `Authorized_Groups` attribute to *auth group list*.

This list becomes the `acl_groups` list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the `Authorized_Users` and `Authorized_Groups` reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the `Authorized_Groups` reservation attribute in [section 6.8, “Reservation Attributes”, on page 304](#).

Syntax:

```
[+|-]<group name>[,+|-]<group name> ...]
```

Default: no default

**11.3.8.3.i Examples of Setting and Changing Reservation Access**

Example 11-19: To disallow access for User1 and allow access for all other users at your domain:

Set reservation's `Authorized_Users` attribute using the `-U` option to `pbs_rsub`:

```
pbs_rsub ... -U "-User1@example.com, +*example.com"
```

Example 11-20: To allow access for Group1 and Group2 only:

Set reservation's `Authorized_Groups` attribute using the `-G` option to `pbs_rsub`:

```
pbs_rsub ... -G "+Group1, +Group2"
```

Note that any users in Group1 and Group2 to whom you wish to grant access must be explicitly granted access in the `Authorized_Users` list.

Example 11-21: To allow access from Host1 and Host2 only:

Set reservation's `Authorized_Hosts` attribute using the `-H` option to `pbs_rsub`:

```
pbs_rsub ... -H "+Host1.example.com, +Host2.example.com, -*.example.com"
```

Example 11-22: To allow User2 and User3 access to the the reservation:

Use `pbs_ralter -U` to add User2 and User3 to the the reservation's `Authorized_Users` attribute:

```
pbs_ralter ... -U "+User2@example.com,+User3@example.com"
```

Example 11-23: To disallow Group3 access to the the reservation:

Use `pbs_ralter -G` to remove Group3 from the the reservation's `Authorized_Groups` attribute:

```
pbs_ralter ... -G "-Group3@example.com"
```

### 11.3.8.4 Reservation Queues

While you will see that each reservation has its own queue, do not attempt to manipulate reservation queue attributes directly. You operate on the reservation attributes, and PBS manages the queue's attributes, making them mirror those of the reservation. Set or modify reservation attributes using [pbs\\_rsub](#) and [pbs\\_ralter](#).

You can move jobs into or out of reservation queues.

#### 11.3.8.4.i Reservation Queue ACLs

If the group or host reservation ACL is specified by the reservation creator, the associated Boolean switch for the reservation queue ACL is set to *True*.

`Authorized_Users` is always set to the creator and copied to the queue's `acl_users` attribute, and `acl_user_enable` is always set to *True*.

If `Authorized_Groups` is specified by the creator, it is copied to the queue's `acl_groups` attribute and `acl_group_enable` is set to *True*. If the reservation creator does not specify a value for `Authorized_Groups`, nothing is copied to the queue's `acl_groups`, and `acl_group_enable` remains at its default value of *False*.

If `Authorized_Hosts` is specified by the creator, it is copied to the queue's `acl_hosts` attribute and `acl_host_enable` is set to *True*. If the reservation creator does not specify a value for `Authorized_Hosts`, nothing is copied to the queue's `acl_hosts`, and `acl_host_enable` remains at its default value of *False*.

The following table shows the relationships between reservation ACLs and reservation queue ACLs:

**Table 11-5: Relationship Between Reservation ACLs and Reservation Queue ACLs**

| Entity | Reservation ACL                | Reservation Queue ACL   | Reservation Queue ACL Switch  |
|--------|--------------------------------|-------------------------|-------------------------------|
| Users  | <code>Authorized_Users</code>  | <code>acl_users</code>  | <code>acl_user_enable</code>  |
| Groups | <code>Authorized_Groups</code> | <code>acl_groups</code> | <code>acl_group_enable</code> |
| Hosts  | <code>Authorized_Hosts</code>  | <code>acl_hosts</code>  | <code>acl_host_enable</code>  |

## 11.3.9 Scope of Access Control

Queue-level ACLs provide different security functionality from that provided by server-level ACLs. Access to PBS commands is controlled by server-level ACLs. For example, access to the `qstat` and `qselect` operations are controlled only at the server level. For unprivileged users, access to a specific queue is controlled through that queue's ACLs.

The users allowed access to a queue or reservation are a subset of the users allowed access to the server. Therefore, if you wish to allow a user access to a queue, that user must also be allowed access to the server. The hosts from which a user may run commands at a queue are a subset of the hosts from which a user may run commands at the server. See [“Server Attributes” on page 281 of the PBS Professional Reference Guide](#), [“Queue Attributes” on page 312 of the PBS Professional Reference Guide](#), and [“Reservation Attributes” on page 304 of the PBS Professional Reference Guide](#).

## 11.3.10 Operations Controlled by ACLs

ACLs control some operations in PBS, but not others. Manager and Operator privileges override some ACL restrictions.

### 11.3.10.1 Server Operations Controlled by ACLs

#### 11.3.10.1.i Server Host ACL

If it is enabled, the server host ACL is checked for and controls all server operations, and is honored regardless of privilege. Any request coming from a disallowed host is denied.

#### 11.3.10.1.ii Server User ACL

If it is enabled, the server's user ACL is checked for and controls all server operations, but is overridden by Manager or Operator privilege. This means that the server's user ACL applies only to users, not to Managers or Operators. Even if explicitly denied access in the server's user ACL, a PBS Manager or Operator is allowed access to the server. Note that queue access is controlled separately by queue ACLs; even if Managers or Operators are explicitly denied access in the server's user ACL, if a queue's ACLs are not enabled, Managers and Operators have access to the queue. The same is true for reservations.

### 11.3.10.2 Queue Operations Controlled by ACLs

If enabled, queue ACLs are applied only when an entity is attempting to enqueue a job. Enqueueing a job can happen in any of three ways:

- Moving a job into the queue
- Submitting a job to the queue
- Routing a job into the queue

Queue ACLs are not applied for non-enqueueing operations, for example:

- Moving a job out of the queue
- Holding a job
- Deleting a job
- Signaling a job
- Getting job status

#### 11.3.10.2.i Queue Host ACL

If a queue's host ACL is enabled, it is checked when an entity attempts to enqueue a job. The host ACL is always honored, regardless of privilege.

#### 11.3.10.2.ii Queue User and Group ACLs

If a queue's user or group ACL is enabled, it is applied when an entity attempts to enqueue a job. Manager and Operator privileges override queue user and group ACLs when an entity attempts to move a job into a queue. This means that a PBS Manager or Operator who is explicitly denied access by the user or group ACL for queue Q1 can still use the `qmove` command to move a job into Q1, as long as other ACLs allow the operation (the server's user and host ACLs must both allow this).

A queue user or group ACL is applied in the following way:

**Table 11-6: How Queue User and Group ACLs Are Applied**

| Operation                          | Applied to Users | Applied to Managers/Operators |
|------------------------------------|------------------|-------------------------------|
| Moving a job into the queue        | Yes              | No                            |
| Submitting a job to the queue      | Yes              | Yes                           |
| Having a job routed into the queue | Yes              | Yes                           |

### 11.3.10.3 Reservation Operations Controlled by ACLs

Access to a reservation's queue is controlled through its queue's ACLs. A reservation's queue behaves exactly the same way as a regular queue.

### 11.3.10.4 Table of Operations Controlled by ACLs and Overrides

The following table lists which operations are and are not controlled by server and queue ACLs, and which controls are overridden.

**Table 11-7: Operations Controlled by ACLs, and ACL Overrides**

| Operation                  | Server ACLs |                  |                   |         |                  |                   | Queue ACLs |                  |                   |         |                  |                   |         |                  |                   |
|----------------------------|-------------|------------------|-------------------|---------|------------------|-------------------|------------|------------------|-------------------|---------|------------------|-------------------|---------|------------------|-------------------|
|                            | Host        |                  |                   | User    |                  |                   | Host       |                  |                   | User    |                  |                   | Group   |                  |                   |
|                            | Applied     | Manager Override | Operator Override | Applied | Manager Override | Operator Override | Applied    | Manager Override | Operator Override | Applied | Manager Override | Operator Override | Applied | Manager Override | Operator Override |
| Move job into queue        | Y           | N                | N                 | Y       | Y                | Y                 | Y          | N                | N                 | Y       | Y                | Y                 | Y       | Y                | Y                 |
| Move job out of queue      | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Submit job to queue        | Y           | N                | N                 | Y       | Y                | Y                 | Y          | N                | N                 | Y       | N                | N                 | Y       | N                | N                 |
| Have job routed into queue | Y           | N                | N                 | Y       | Y                | Y                 | Y          | N                | N                 | Y       | N                | N                 | Y       | N                | N                 |
| Delete job                 | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Hold job                   | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Release job                | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Signal job                 | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Status job                 | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Status server              | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |
| Status queue               | Y           | N                | N                 | Y       | Y                | Y                 | N          | -                | -                 | N       | -                | -                 | N       | -                | -                 |

## 11.3.11 Avoiding Problems

### 11.3.11.1 Using Group Lists

When a user specifies a group list, each and every group in which that user might execute a job must have a group name and an entry in the groups database, for example, `/etc/group`.

## 11.3.12 Flatuid and Access

The server's `flatuid` attribute affects both when users can operate on jobs and whether users without accounts on the server host can submit jobs.

### 11.3.12.1 How flatuid Controls When Users Can Operate On Jobs

This section describes how the server's `flatuid` attribute affects the circumstances under which users can operate on jobs.

This attribute specifies whether, for each user, the username at the submission host must be the same as the one at the server host. The username at the server host must always be the same as the username at the execution host. When `flatuid` is set to *True*, the server assumes that `UserA@host1` is the same as `UserA@host2`. Therefore, if `flatuid` is *True*, `UserA@host2` can operate on `UserA@host1`'s job.

The value of `flatuid` also affects whether `.rhosts` and `hosts.equiv` are checked. If `flatuid` is *True*, `.rhosts` and `hosts.equiv` are not queried, and for any users at `host2`, only `UserA` is treated as `UserA@host1`. If `flatuid` is *False*, `.rhosts` and `hosts.equiv` are queried.

That is, when `flatuid` is *True*, even if `UserB@host2` is in `UserA@host1`'s `.rhosts`, `UserB@host2` cannot operate on `UserA`'s job(s). If `flatuid` is *False*, and `UserB@host2` is in `UserA@host1`'s `.rhosts`, `UserB@host2` is allowed to operate on `UserA`'s job(s).

Example:

`UserA@host1` has a job

`UserB@host2` is in `UserA@host1`'s `.rhosts`

- a. `flatuid = True`: `UserB@host2` cannot operate on `UserA`'s job
- b. `flatuid = False`: `UserB@host2` can operate on `UserA`'s job

The following table shows how access is affected by both the value of the server's `flatuid` attribute and whether `UserB@host2` is in `UserA@host1`'s `.rhosts`:

**Table 11-8: Effect of `flatuid` Value on Access**

|                                                                   | <b>flatuid = True</b> |           | <b>flatuid = False</b> |           |
|-------------------------------------------------------------------|-----------------------|-----------|------------------------|-----------|
|                                                                   | <b>Yes</b>            | <b>No</b> | <b>Yes</b>             | <b>No</b> |
| <b>UserB@host2 in UserA@host1's .rhosts</b>                       |                       |           |                        |           |
| Is <code>UserA@host1</code> treated as <code>UserA@host2</code> ? | Yes                   | Yes       | No                     | No        |
| Is <code>.rhosts</code> queried?                                  | No                    | No        | Yes                    | Yes       |
| Can <code>UserB</code> operate on <code>UserA</code> 's jobs?     | No                    | No        | Yes                    | No        |

### 11.3.12.2 How `flatuid` Affects Users Without Server Accounts

This section describes how the server's `flatuid` attribute affects users who have no account on the server host.

#### 11.3.12.2.i Linux and `flatuid`

- If `flatuid` is set to *False*, users who have no account at the server host cannot submit jobs to PBS.
- If `flatuid` is set to *True*, these users can submit jobs. However, the job will only run if it is sent to execution hosts where the user does have an account. If the job is sent to execution hosts where the user does not have an account, the job will not run, and the MoM will log an error message.

#### 11.3.12.2.ii Windows and `flatuid`

Regardless of the value of `flatuid`, users who have no account at the server host cannot submit jobs to PBS. Users must have an account at the server, and it must have the same password.

## 11.4 Authentication for Daemons & Users

PBS uses a client-server model for authentication. The following table shows the authentication method used for each communication pair:

**Table 11-9: Authentication Method Selection**

| Sender                                                    | Recipient  | Authentication Method Specified At... |
|-----------------------------------------------------------|------------|---------------------------------------|
| PBS server                                                | Comm       | Server (in this case, PBS server)     |
| MoM                                                       | Comm       | Client (in this case, MoM)            |
| Comm A                                                    | Comm B     | Client (in this case, comm A)         |
| PBS commands, e.g. <code>qsub</code> , <code>qstat</code> | PBS server | Client (in this case, the command)    |

Communication between MoM and comm or PBS server and comm is initiated by MoM or PBS server, not comm.

By default, PBS on Linux uses reserved ports for authentication of daemons and users. On Windows, PBS uses `pwd`. You can use other methods such as MUNGE. We use MUNGE for mixed-mode operation (mixed Linux and Windows complexes).

For server-to-scheduler communication, PBS always uses reserved ports for authentication; this is not configurable.

Authentication is independent of encryption. For encryption, see [section 11.5, “Encrypting PBS Communication”, on page 517](#).

### 11.4.1 Specifying Allowed Authentication Methods

PBS can use more than one authentication method at the same time. You specify which authentication methods are to be allowed by listing them in the `PBS_SUPPORTED_AUTH_METHODS` parameter in `pbs.conf` on all PBS hosts. If you leave this field blank, it defaults to "resvport" (reserved ports). If you specify any value, for example "munge", that is the only allowed method. So if you want both reserved ports and MUNGE, use "munge,resvport" (without quotes). This value is used only by the authenticating server, and is ignored by the client.

#### 11.4.1.1 Supported Authentication Methods

You can use any of the following authentication methods/libraries:

MUNGE

resvport (reserved port)

pwd (password, used on Windows)

If you do not configure a method, PBS uses resvport.

### 11.4.2 Specifying Authentication Method Used by Authentication Client

To specify the default method to be used by an authentication client at a given host, set the `PBS_AUTH_METHOD` parameter in `pbs.conf` on that host to the desired library/method, for example, "munge". This parameter is case-insensitive. The `PBS_AUTH_METHOD` parameter in `pbs.conf` is used only by the authentication client.



Make sure that the authentication method you choose for the authentication client is listed in the `PBS_SUPPORTED_AUTH_METHODS` parameter in `pbs.conf` on the server host. This parameter is case-insensitive.

### 11.4.3 Authentication via Reserved Ports

When using reserved ports, PBS commands and daemons can call the `pbs_iff` command to authenticate a user or daemon. The `pbs_iff` command runs as a privileged user, binds to a reserved port, and sends a request from the client to the server.

### 11.4.4 Authentication via MUNGE

You can use the MUNGE authentication daemon to create and validate credentials within a PBS complex, so that communication for PBS commands and daemons is validated via MUNGE. Using MUNGE, the hosts in the PBS complex form a security realm and share a cryptographic key. PBS Professional uses the MUNGE authentication service to authenticate the UID and GID of PBS processes, and to create and validate credentials.

The client machines in the complex can create and validate credentials without using root privilege, reserved ports, or methods requiring a specific platform. All PBS daemons are authenticated via MUNGE when they try to connect to `pbs_comm`. MUNGE uses the key in `/etc/munge/munge.key`.

#### 11.4.4.1 Steps to Integrate MUNGE with PBS

1. Download and install a supported version of MUNGE on all machines in the PBS complex. This includes server, execution, comm, submission hosts, and cloud nodes (install MUNGE on the instance that you will use to burst cloud nodes; see the individual cloud provider instructions). You can get MUNGE either via your Linux distribution package repositories or from the MUNGE project directly.

Follow the MUNGE installation instructions at <https://github.com/dun/munge/wiki/Installation-Guide>.

2. On the PBS server host, generate the `munge.key` file using the `create-munge-key` command.
3. On every host, if the library name is not exactly "libmunge.so", for example "libmunge.so.2", add a soft link to it. For example:

```
ln -s /lib64/libmunge.so.2 /lib64/libmunge.so
```

4. Copy `/etc/munge/munge.key` on the PBS server host to `/etc/munge/munge.key` on all hosts in the complex.
5. Start MUNGE:

```
systemctl start munge
```

6. On the server host, edit the PBS configuration file (`/etc/pbs.conf`) and add these lines:

```
PBS_AUTH_METHOD=MUNGE
PBS_SUPPORTED_AUTH_METHODS="pwd,munge"
```

7. On client and execution hosts, edit the PBS configuration file (`/etc/pbs.conf`) and add this line:

```
PBS_AUTH_METHOD=MUNGE
```

8. Restart the PBS daemons:

```
systemctl restart pbs
```

---

## 11.4.5 Configuring SSSD

### 11.4.5.1 Configuring SSSD on RHEL 7 and CentOS 7

We show an example of configuring SSSD on RHEL 7 and CentOS 7, using the following steps:

1. Install the required packages:

- a. Install required packages for sssd:

```
yum install realmd oddjob oddjob-mkhomedir sssd adcli openldap-clients policycoreutils-python
samba-common samba-common-tools krb5-workstation
```

- b. Check whether libpam is already installed on the system. If not, install libpam.
- c. The pam library name may be libpam.so.<version>. If so, you may need to create a soft link:

```
ln -s /usr/lib64/libpam.so.0.83.1 /usr/lib64/libpam.so
```

2. Find out whether we are in a domain:

```
realm list
```

3. Discover the Active Directory domain for your Windows hosts:

```
realm discover <domain controller hostname>.<domain to join>.com
<domain to join>.com
type: kerberos
realm-name: <domain to join>.COM
domain-name: <domain to join>.com
configured: no
server-software: active-directory
client-software: sssd
required-package: oddjob
required-package: oddjob-mkhomedir
required-package: sssd
required-package: adcli
required-package: samba-common-tools
```

4. Add the Linux host to Active Directory:

```
realm join --user=Administrator@<domain to join>.com <domain controller hostname>.<domain to
join>.com
```

5. If no errors are encountered, users should be able to see the domain information:

```
realm list
type: kerberos
realm-name: <domain to join>.COM
domain-name: <domain to join>.com
configured: kerberos-member
server-software: active-directory
client-software: sssd
required-package: oddjob
required-package: oddjob-mkhomedir
required-package: sssd
required-package: adcli
```

```
required-package: samba-common-tools
login-formats: %U@<domain to join>.com
login-policy: allow-realm-logins
```

6. Verify that the Kerberos configuration file `/etc/krb5.conf` and `sssd` configuration file `/etc/sss/sss.conf` have the correct domain name specified where required.

7. Set the appropriate permissions for `sss.conf`:

```
chown root:root /etc/sss/sss.conf
chmod 0600 /etc/sss/sss.conf
restorecon /etc/sss/sss.conf
authconfig --enablesss --enablesssdauth --enablemkhomedir --update
systemctl start sssd
```

8. In the file `/etc/sss/sss.conf`, set `use_fully_qualified_names` to *False*:

```
use_fully_qualified_names = False
```

9. Restart the `sss` service:

```
systemctl restart sssd
```

### 11.4.5.2 Configuring SSSD on RHEL8

Example 11-24: Configuring SSSD on RHEL8:

1. Follow the steps in [section 11.4.5.1, “Configuring SSSD on RHEL 7 and CentOS 7”, on page 510](#)
2. Make sure that the following lines are in the `/etc/pam.d/passwd` file; add them if necessary:

```
auth include system-auth
account include system-auth
```

3. Add the following line to `/etc/sss/sss.conf`:

```
access_provider = permit
```

### 11.4.5.3 Configuring SSSD on Ubuntu 16

Example 11-25: Configuring SSSD on Ubuntu 16:

1. Follow the steps in [section 11.4.5.1, “Configuring SSSD on RHEL 7 and CentOS 7”, on page 510](#)
2. Make sure that the following lines are in the `/etc/pam.d/passwd` file; add them if necessary:

```
auth include common-auth
account include common-auth
```

3. Add the following line to `/etc/sss/sss.conf`:

```
access_provider = permit
```

---

### 11.4.5.4 Configuring SSSD on Ubuntu 18

Configuring SSSD on Ubuntu 16:

1. Update and install:

```
sudo apt -y update
sudo apt-get install -y packagekit
sudo apt install sssd-ad sssd-tools realmd adcli
```

2. In /etc/hosts, add an entry for the host where AD is configured

3. Discover and join the domain:

```
sudo realm -v discover ad1.example.com
sudo realm join --user <administrator>@<domain to join>.COM <domain controller hostname>.<domain to join>.com
```

4. List the newly joined domain:

```
realm list
sudo pam-auth-update --enable mkhomedir
```

5. In the /etc/sss/sss.conf file, set the following:

```
use_fully_qualified_names = False
```

6. Restart sssd:

```
service sssd restart
```

7. Check whether sssd works:

```
id <username>
su - <username>
```

For more information, see <https://ubuntu.com/server/docs/service-sss>.

### 11.4.5.5 Configuring SSSD on SUSE 15

Example 11-26: Configuring SSSD on SUSE 15 in order to connect to a Windows server host:

We use these settings for our example:

- Windows Domain = WINAUTHTEST.COM
- Windows Server Name = advmsetup
- Windows Server IP Address = 10.79.102.6
- AD Administrator user = loginuser
- Test User on AD = servacc

1. Install required packages and their dependencies:

```
zypper ref
zypper in krb5-client samba-client sssd sssd-ad
```

2. Edit /etc/krb5.conf so that it contains the following:

```
includedir /etc/krb5.conf.d

[libdefaults]
"dns_canonicalize_hostname" and "rdns" are better set to false for improved security.
If set to true, the canonicalization mechanism performed by Kerberos client may
allow service impersonification, the consequence is similar to conducting TLS certificate
verification without checking host name.
If left unspecified, the two parameters will have default value true, which is less secure.
dns_canonicalize_hostname = false
rdns = false
default_realm = WINAUTHTEST.COM
dns_lookup_realm = false

[realms]
WINAUTHTEST.COM = {
 kdc = advmsetup.winauthtest.com
 master_kdc = advmsetup.winauthtest.com
 admin_server = advmsetup.winauthtest.com
}

[logging]
kdc = FILE:/var/log/krb5/krb5kdc.log
admin_server = FILE:/var/log/krb5/kadmind.log
default = SYSLOG:NOTICE:DAEMON

[domain_realm]
.winauthtest.com = WINAUTHTEST.COM
winauthtest.com = WINAUTHTEST.COM
```

3. Edit /etc/samba/smb.conf. Add the following lines to the global section:

```
[global]
workgroup = WINAUTHTEST
passdb backend = tdbsam
```

```
printing = cups
printcap name = cups
printcap cache time = 750
cups options = raw
map to guest = Bad User
logon path = \\%L\profiles\.msprofile
logon home = \\%L%\U\.9xprofile
logon drive = P:
usershare allow guests = Yes
realm = WINAUTHTEST.COM
security = ADS
template shell = /bin/bash
winbind refresh tickets = yes
winbind use default domain = yes
kerberos method = secrets and keytab
client signing = yes
client use spnego = yes
```

4. Edit `/etc/hosts`. Add the Windows server IP address and hostname:

```
10.79.102.6 advmsetup.winauthtest.com advmsetup winauthtest.com
10.79.102.22 sles15server2 sles15server2.winauthtest.com
```

5. Add the SLES 15 server to the AD domain:

- a. Run the `kinit` command as administrator:

```
pbsadmin@sles15server:~> kinit loginuser
```

Password for loginuser@WINAUTHTEST.COM:

Warning: Your password will expire in 3 days on Sunday 14 June 2022 02:16:13 PM UTC

- b. Join the AD domain:

```
pbsadmin@sles15server:~> sudo net ads join -U loginuser -S advmsetup.winauthtest.com
```

Enter loginuser's password:

Using short domain name -- WINAUTHTEST

Joined 'SLES15SERVER' to dns domain 'winauthtest.com'

No DNS domain configured for sles15server. Unable to perform DNS Update.

DNS update failed: NT STATUS\_INVALID\_PARAMETER

(Ignore the warning you get here for DNS.)

- Verify that the user is authenticated to the SLES server using `ldapsearch`. This gives the AD attributes:

```
pbsadmin@sles15server:~> /usr/bin/ldapsearch -H ldap://adwmsetup.winauthtest.com/ -Y GSSAPI -N -b "dc=winauthtest,dc=com" "(&(objectClass=user)(sAMAccountName=servacc))"
```

SASL/GSSAPI authentication started

SASL username: loginuser@WINAUTHTEST.COM

SASL SSF: 56

SASL data security layer installed.

```
extended LDIF
```

#

# LDAPv3

```
base <dc=winauthtest,dc=com> with scope subtree
```

```
filter: (&(objectClass=user)(sAMAccountName=servacc))
requesting: ALL
#

servacc, Users, winauthtest.com
dn: CN=servacc,CN=Users,DC=winauthtest,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: servacc
givenName: servacc
distinguishedName: CN=servacc,CN=Users,DC=winauthtest,DC=com
instanceType: 4
whenCreated: 20200403032934.0Z
whenChanged: 20200608034248.0Z
displayName: servacc
uSNCreated: 69750
uSNChanged: 335995
name: servacc
objectGUID:: micBdtED4UKRiEl4r2bFCg==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 132361895928844770
lastLogoff: 0
lastLogon: 132363412329813043
pwdLastSet: 132325361902386414
primaryGroupID: 513
objectSid:: AQUAAAAAAAAUVAAAacGLL19eyqDLBPfLvhsAAA==
accountExpires: 9223372036854775807
logonCount: 6695
sAMAccountName: servacc
sAMAccountType: 805306368
userPrincipalName: servacc@winauthtest.com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=winauthtest,DC=com
dsCorePropagationData: 20200403033300.0Z
dsCorePropagationData: 16010101000001.0Z
lastLogonTimestamp: 132360613688616817

search reference
ref: ldap://ForestDnsZones.winauthtest.com/DC=ForestDnsZones,DC=winauthtest,DC=com

search reference
```



```
ref: ldap://DomainDnsZones.winauthtest.com/DC=DomainDnsZones,DC=winauthtest,DC=com
```

```
search reference
```

```
ref: ldap://winauthtest.com/CN=Configuration,DC=winauthtest,DC=com
```

```
search result
```

```
search: 4
```

```
result: 0 Success
```

```
numResponses: 5
```

```
numEntries: 1
```

```
numReferences: 3
```

7. Edit `/etc/sss/sss.conf`. You add the domain in the *domain* section. Add the following lines in the *sss*, *nss*, and *domain* sections:

```
[sss]
config_file_version = 2
services = nss, pam
SSSD will not start if you do not configure any domains.
Add new domain configurations as [domain/<NAME>] sections, and
then add the list of domains (in the order you want them to be
queried) to the "domains" attribute below and uncomment it.
; domains = LDAP
domains = winauthtest.com
```

```
[nss]
```

```
filter_users = root
```

```
filter_groups = root
```

```
[pam]
```

```
[domain/winauthtest.com]
debug_level = 6
id_provider = ad
auth_provider = ad
ad_domain = winauthtest.com
ad_server = advmsetup.winauthtest.com
ad_hostname = advmsetup.winauthtest.com
ldap_id_mapping = True
override_homedir = /home/%u
ldap_schema = ad
default_shell = /bin/bash
use_fully_qualified_names = False
```

8. Edit `/etc/nsswitch.conf`. NSS is used for name resolution. Since we are adding another service to resolve names, that service needs to be added to `/etc/nsswitch.conf`. Add 'sss' to the *passwd* and *group* fields:

```
sles15server:/home/pbsadmin # vi /etc/nsswitch.conf
```

```
passwd: compat sss
group: compat sss
shadow: compat
```

9. Edit `/etc/nscd.conf`.

Prevent `nscd` from caching so that it doesn't interfere with the `sss` password and group caching. If both `nscd` and `sss` are caching passwords and groups, the daemons could conflict and AD users would not be resolved. Find the `enable-cache` option for the password and group and set it to `'no'`:

```
enable-cache passwd no
enable-cache group no
```

10. Restart the `nscd` service:

```
sudo systemctl restart nscd
```

11. Start the `sss` service:

```
sudo systemctl start sssd
```

You can verify whether `sss` is started:

```
ps -eaf | grep sssd]
```

12. Enable authentication through SSSD to AD. Add the `sss` `pam` module:

```
pam-config --add --sss
pam-config --add --mkhomedir
```

13. Test user resolution and authentication.

- a. Run the `id` command to check username resolution:

```
id servacc
```

- b. Switch user to see whether the home directory is created:

```
su - servacc
```

For more information on configuring `sss`, see <https://www.suse.com/support/kb/doc/?id=000019039>.

## 11.5 Encrypting PBS Communication

PBS can encrypt communication sent via commands and between daemons, providing end-to-end encryption. To encrypt your PBS communication, provide the encryption mechanism, and set the `PBS_ENCRYPT_METHOD` parameter in `pbs.conf` on all PBS hosts to the method that clients will use. For end-to-end encryption, set it on all PBS hosts.

You may want to use encryption especially for cloud hosts.

TLS encryption is required on Windows.

### 11.5.1 Supported Encryption Methods

PBS supports TLS for encryption.

---

## 11.5.2 Using Transport Layer Security (TLS) for Client-Server Communication

You can use transport layer security (TLS) encryption for a PBS complex that has both Windows and Linux execution hosts, or when you want an extra layer of security. TLS encryption will provide greater security for your client-server connections when one PBS daemon sends a request to another daemon.

Encryption is independent of authentication. For authentication information, see [section 11.4, “Authentication for Daemons & Users”, on page 508](#).

### 11.5.2.1 Overview of Configuring PBS for TLS Encryption

We walk you through the steps to configure PBS for TLS encryption, and we provide example steps here. To summarize:

1. Get or create a CA certificate (the public certificate)
2. Get or create a self-signed TLS certificate
3. Copy the TLS certificate into the appropriate location
4. Generate a private key
5. Edit `pbs.conf` and set TLS as your encryption method
6. Restart PBS

For additional information, see <https://www.openssl.org/docs/man1.1.1/man1/openssl-ca.html>.

### 11.5.2.2 Example of Configuring PBS for TLS Encryption

The following steps show an example of configuring PBS for TLS encryption.

1. Log in as root or administrator.  
Perform all of the following steps as root on Linux or Administrator on Windows.
2. Create a configuration file for a certificate using X509v3 extensions.

For this step, make sure you choose options and configuration parameters that meet your requirements. See the OpenSSL documentation for help. In our example, the file is named "my.conf" and the current working directory is /root/certs. Contents of my.conf:

```
[cacert]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:TRUE
keyUsage = critical, digitalSignature, cRLSign, keyCertSign, keyEncipherment
extendedKeyUsage = clientAuth, serverAuth, emailProtection
nsCertType = server, client, email
nsComment = "CA Certificate Generated By OpenSSL for PBSPro"

[usrcert]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
basicConstraints = critical, CA:FALSE
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, serverAuth, emailProtection
nsCertType = server, client, email
nsComment = "User Certificate Generated By OpenSSL for PBSPro"
```

3. Generate your root certificate authority:

```
openssl genrsa -out rootca.key.pem 4096
openssl req -new -key rootca.key.pem -out rootca.csr.pem -subj "/O=PBSPro/OU=PBSPro/CN=RootCA/"
openssl x509 -req -signkey ./rootca.key.pem -extfile ./my.conf -extensions cacert -days 12775
 -in rootca.csr.pem -out rootca.cert.pem
```

4. Generate your intermediate certificate authority:

```
openssl genrsa -out intca.key.pem 4096
openssl req -new -key intca.key.pem -out intca.csr.pem -subj "/O=PBSPro/OU=PBSPro/CN=IntCA/"
openssl x509 -req -CAkey ./rootca.key.pem -CA ./rootca.cert.pem -CAcreateserial -CAserial
 ./serials.txt -extfile ./my.conf -extensions cacert -days 9125 -in intca.csr.pem -out
 intca.cert.pem
```

5. Generate your CA certificate:

```
cat rootca.cert.pem intca.cert.pem > ca.cert.pem
```

6. Generate certificates for PBS server and communication daemons:

```
openssl genrsa -out pbspro.key.pem 2048
openssl req -new -key pbspro.key.pem -out pbspro.csr.pem -subj
 "/O=PBSPro/OU=PBSPro/CN=PBSProServices/"
openssl x509 -req -CAkey ./intca.key.pem -CA ./intca.cert.pem -CAcreateserial -CAserial
 ./serials.txt -extfile ./my.conf -extensions usrcert -days 1825 -in pbspro.csr.pem -out
 pbspro.cert.pem
openssl verify -CAfile ca.cert.pem pbspro.cert.pem
```

7. Edit PBS configuration files:

On each PBS host (server, scheduler, MoM, comm, client), edit `pbs.conf`, and set the `PBS_ENCRYPT_METHOD` parameter to "tls" (don't include the quotes). The `PBS_ENCRYPT_METHOD` parameter is case-insensitive:

```
PBS_ENCRYPT_METHOD=tls
```

8. Make it so we can use the value of `PBS_HOME` in `pbs.conf`:

```
source /etc/pbs.conf
```

9. Create certificate directory:

The PBS server and comms use a certificate key pair stored in a certificate directory.

On each host running a PBS server or comm, create `PBS_HOME/certs`:

```
mkdir ${PBS_HOME}/certs
```

10. Copy files into certificate directory:

- a. Copy your `pbspro.cert.pem` file to `${PBS_HOME}/certs/cert.pem`:

```
cp /root/certs/pbspro.cert.pem ${PBS_HOME}/certs/cert.pem
```

- b. Copy your `pbspro.key.pem` file to `${PBS_HOME}/certs/key.pem`:

```
cp /root/certs/pbspro.key.pem ${PBS_HOME}/certs/key.pem
```

11. Set permissions and ownership for certificate directory:

- a. Make sure that permissions for the certificate directory and its contents are `0600`:

```
chmod -R 0600 ${PBS_HOME}/certs
```

- b. Make sure the owner is root on Linux or Administrator on Windows:

```
chown -R root: ${PBS_HOME}/certs
```

12. Install CA certificate file:

On each host running a PBS server or comm, install the file in `/etc/pbs_ca.pem` (Linux), or `<PBS installation directory>/pbs_ca.pem` (Windows). For example, if PBS is installed on Windows in `C:\Program Files (x86)\PBS`, you'd put the file there.

Copy your `/root/certs/ca.cert.pem` file to `/etc/pbs_ca.pem`:

```
cp /root/certs/ca.cert.pem /etc/pbs_ca.pem
```

13. Set permissions and ownership for CA certificate file:

- a. Make sure the permissions for the file are `0644`:

```
chmod -R 0644 /etc/pbs_ca.pem
```

- b. Make sure the owner is root on Linux or Administrator on Windows:

```
chown -R root: /etc/pbs_ca.pem
```

14. Restart PBS daemons:

- On every Linux host in the complex:

```
<path to start/stop script>/pbs restart
```

or

```
systemctl start pbs
```

- On every Windows execution host in the complex:

```
net stop pbs_mom
```

```
net start pbs_mom
```

## 11.6 Restricting Execution Host Access

You can configure each PBS execution host so that the only users who have access to the machine are those who are running jobs on the machine. You can specify this by adding the `$restrict_user` parameter to the MoM configuration file `PBS_HOME/mom_priv/config`. This parameter is a Boolean, which if set to `True`, prevents any user not running a job from running any process on the machine for more than 10 seconds. The interval between when PBS applies restrictions depends upon MoM's other activities, but can be no more than 10 seconds.

You can specify which users are exempt from this restriction by adding the `$restrict_user_exceptions` parameter to the same file. See the description of the parameter in the next section.

You can allow system processes to run by specifying the maximum numeric user ID allowed access to the machine when not running a job. You do this by adding the `$restrict_user_maxsysid` parameter to the MoM configuration file. PBS automatically tries to allow system processes to run: if `$restrict_user` is enabled and `$restrict_user_maxsysid` is unset, PBS looks in `/etc/login.defs` for `SYSTEM_UID_MAX` for the value to use. If there is no maximum ID set there, it looks for `SYSTEM_MIN_UID`, and uses that value minus 1. Otherwise PBS uses the default value of 999. See [section 9.5.7, “Restricting User Access to Execution Hosts”, on page 438](#) and [“\\$restrict\\_user {True | False}” on page 249 of the PBS Professional Reference Guide](#).

Access to `pbs_mom` is controlled through a list of hosts specified in the `$clienthost` parameter in the `pbs_mom`'s configuration file. By default, only "localhost", the name returned by `gethostname(2)`, and the host named by `PBS_SERVER` from `/etc/pbs.conf` are allowed. See [“MoM Parameters” on page 243 of the PBS Professional Reference Guide](#) for more information on the configuration file.

### 11.6.1 MoM Access Configuration Parameters

These are the configuration parameters in `PBS_HOME/mom_priv/config` that can be set to restrict and specify access to each execution host. Each execution host has its own configuration file.

#### `$clienthost`

List of hosts which are allowed to connect to MoM as long as they are using a privileged port. For example, this allows the hosts "fred" and "wilma" to connect to MoM:

```
$clienthost fred
$clienthost wilma
```

The following hostnames are added to `$clienthost` automatically: the server, the localhost, and if configured, the secondary server. The server sends each MoM a list of the hosts in the nodes file, and these are added internally to `$clienthost`. None of these hostnames need to be listed in the configuration file.

Two hostnames are always allowed to connect to `pbs_mom`, "localhost" and the name returned to MoM by the system call `gethostname()`. These hostnames do not need to be added to the MoM configuration file.

The hosts listed as "clienthosts" make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

#### `$restrict_user <value>`

Controls whether users not submitting jobs have access to this machine. When `True`, only those users running jobs are allowed access.

Format: Boolean

Default: off

#### `$restrict_user_exceptions <user_list>`

List of users who are exempt from access restrictions applied by `$restrict_user`. Maximum number of names in list is 10.

Format: Comma-separated list of usernames; space allowed after comma

**\$restrict\_user\_maxsysid <value>**

Allows system processes to run when \$restrict\_user is enabled. Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by \$restrict\_user.

Format: Integer

Default: 999

## 11.6.2 Examples of Restricting Access

To restrict user access to those running jobs, add:

```
$restrict_user True
```

To specify the users who are allowed access whether or not they are running jobs, add:

```
$restrict_user_exceptions <user list>
```

For example:

```
$restrict_user_exceptions User1, User2
```

To allow system processes to run, specify the maximum numeric user ID by adding:

```
$restrict_user_maxsysid <user ID>
```

For example:

```
$restrict_user_maxsysid 999
```

## 11.7 Changing the PBS Service Account Password

Normally, the password for the PBS service account on Windows should not be changed. But if it is necessary to change it, perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account's password on one machine in a command prompt from an admin-type of account by typing:

Domain environments:

```
net user <name of PBS service account> * /domain
```

Non-domain environment:

```
net user <name of PBS service account> *
```

2. Provide the Service Control Manager (SCM) with the new password given above. Do this either using the GUI-based Services application which is one of the Administrative Tools, or by unregistering and re-registering the PBS services with the password. See [“pbs\\_account” on page 54 of the PBS Professional Reference Guide](#).

To unregister:

```
pbs_account --unreg "%Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"
```

To re-register:

```
pbs_account --reg "%Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"
```

When re-registering, you can give an additional -p password argument to the pbs\_account command, to specify the password on the command line.



---

## 11.8 Paths and Environment Variables

A significant effort has been made to ensure the various PBS components themselves cannot be a target of opportunity in an attack on the system. The two major parts of this effort are the security of files used by PBS and the security of the environment. Any file used by PBS, especially files that specify configuration or other programs to be run, must be secure. The files must be owned by root and in general cannot be writable by anyone other than root.

A corrupted environment is another source of attack on a system. To prevent this type of attack, each component resets its environment when it starts. If it does not already exist, the `environment` file is created during the install process. As built by the install process, it will contain a very basic path and, if found in root's environment, the following variables:

- TZ
- LANG
- LC\_ALL
- LC\_COLLATE
- LC\_CTYPE
- LC\_MONETARY
- LC\_NUMERIC
- LC\_TIME

The `environment` file may be edited to include the other variables required on your system.

The entries in the `PBS_ENVIRONMENT` file can take two possible forms:

```
variable_name=value
variable_name
```

In the latter case, the value for the variable is obtained before the environment is reset.

### 11.8.1 Path Caveats

Note that `PATH` must be included. This value of `PATH` will be passed on to batch jobs. To maintain security, it is important that `PATH` be restricted to known, safe directories. Do NOT include "." in `PATH`. Another variable which can be dangerous and should not be set is `IFS`.

## 11.9 File and Directory Permissions

Each parent directory above `PBS_HOME` must be owned by root and writable by root only. All files and directories used by PBS should be writable by root only. Permissions should allow read access for all files and directories except those that are private to the daemons. The following should not be writable by any but root:

```
PBS_HOME/mom_priv
<sched_priv directory>
PBS_HOME/server_priv
```

The `PBS_HOME` directory must be readable and writable from server hosts by root (Administrator) on Linux.

On Windows, `PBS_HOME` must have Full Control permissions for the local "Administrators" group on the local host.

PBS checks permissions for certain files and directories. The following error message is printed for certain files and directories (e.g. `/etc/pbs.conf`, `/var/spool/PBS/mom_priv/config`, etc.) if their permissions present a security risk:

```
<command>: Not owner (1) in chk_file_sec, Security violation "<directory>" resolves to
"<directory>"
<command>: Unable to configure temporary directory.
```

## 11.10 Root-owned Jobs

The server will reject any job which would execute under the UID of zero unless the owner of the job, typically root, is listed in the server attribute `acl_roots`.

In order to submit a job from a root account on the local host, be sure to set `acl_roots`. For instance, if user foo has root privilege, you need to set:

```
Qmgr: set server acl_roots += foo
```

in order to submit jobs and not get a "bad UID for job execution" message.

Windows Administrators are not considered to have root access, so a Windows Administrator can run a job without being listed in `acl_roots`.

### 11.10.1 Caveats for Root-owned Jobs

Allowing root jobs means that they can run on a configured host under the same account which could also be a privileged account on that host.

## 11.11 Passwords

PBS has different password requirements dictated by the Linux and Windows operating systems. Jobs submitted on Linux systems do not require passwords. Jobs on Windows MoM systems require passwords.

See the PBS Professional 2024.1 release notes for a list of supported architectures.

### 11.11.1 Windows User Passwords

Windows execution host systems require a password for PBS to run a process as the user, so users on these systems must supply a password. Users cache their passwords via the [pbs\\_login](#) command. Job submitters run the [pbs\\_login](#) command once per submission host, initially and for each password change.

## 11.11.2 Changing the PBS Service Account Password

Normally, the PBS service account password should not be changed. But if it is necessary to change it perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account's password on a machine in a command prompt from an admin-type of account by typing:

```
net user <name of PBS service account> * /domain
```

2. Provide the Service Control Manager (SCM) with the new password specified above. This can be done via the GUI-based Services application found as one of the Administrative Tools, or by unregistering and re-registering the PBS MoM with the new password.

```
pbs_account --unreg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"
```

```
pbs_account --reg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"
```

The register form (last line above) can take an additional argument `-p password` so that you can specify the password on the command line directly.

3. Run the `pbs_login` command:

```
pbs_login -m <PBS service account password>
```

4. Restart MoM:

```
net stop pbs_mom
```

```
net start pbs_mom
```

### 11.11.2.1 Caveats for Changing Service Account Password

Using `pbs_account --unreg` and `pbs_account --reg` stops and restarts MoM, which can kill jobs.

## 11.12 Windows Firewall

Under Windows, the Windows Firewall may have been turned on by default. If so, it will block incoming network connections to all services including PBS. Therefore after installing PBS Professional, to allow `pbs_mom` to accept incoming connections:

Access *Settings->Control Panel->Security Center->Windows Firewall*, and verify that the Windows Firewall has been set to "ON" to block incoming network connections.

From this panel, you can either turn Windows Firewall "off", or click on the *Exceptions* tab and add the following to the list:

```
[INSTALL_PATH]\exec\sbin\pbs_mom.exe
```

## 11.13 Logging Security Events

Each PBS daemon logs security-related events, at event class 32 (0x0020) or at event class 128 (0x0080). For information about daemon logfiles, see [section 9.4, "Event Logging", on page 429](#).

### 11.13.1 Events Logged at Event Class 32 (0x0020)

The following security-related events are logged at decimal event class 32 (0x0020):

- When an execution host has access restrictions in place via the `$restrict_user` configuration parameter, and MoM detects that a user who is not exempt from access restriction is running a process on the execution host, MoM kills that user's processes and writes a log message:

```
01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
```

See [section 11.6, “Restricting Execution Host Access”, on page 521](#).

- If for some reason the access permissions on the PBS file tree are changed from their default settings, a daemon may detect this as a security violation, refuse to execute, and write an error message in the corresponding log file. The following are examples of each daemon's log entry:

```
Server@<host>: Permission denied (13) in chk_file_sec, Security violation
"/var/spool/pbs/server_priv/jobs/" resolves to "/var/spool/pbs"
pbs_mom: Permission denied (13) in chk_file_sec, Security violation
"/var/spool/pbs/mom_priv/jobs/" resolves to "/var/spool/pbs"
pbs_sched: Permission denied (13) in chk_file_sec, Security violation "/var/spool/pbs/sched_priv"
resolves to "/var/spool/pbs"
```

A Manager can run `pbs_probe` (on Linux) or `pbs_mkdirs` (on Windows) to check and optionally correct any directory permission or ownership problems.

- When a user without a password entry (an account) on the server attempts to submit a job, the server logs this event. The following is an example log entry:
 

```
8/21/2009 15:28:30;0080;Server@capella;Req;req_reject;Reject reply code=15023, aux=0, type=1,
from User1@host1.example.com
```
- If a daemon detects that a file or directory in the PBS hierarchy is a symbolic link pointing to a non-secure location, this is written to the daemon's log. The resulting log message is the same as for a permission violation:
 

```
Server@<host>: Permission denied (13) in chk_file_sec, Security violation
"/var/spool/pbs/server_priv/jobs/" resolves to "/var/spool/pbs"
pbs_mom: Permission denied (13) in chk_file_sec, Security violation
"/var/spool/pbs/mom_priv/jobs/" resolves to "/var/spool/pbs"
pbs_sched: Permission denied (13) in chk_file_sec, Security violation "/var/spool/pbs/sched_priv"
resolves to "/var/spool/pbs"
```
- If an `$action` script is to be executed for a job belonging to a user who does not have an account on an execution host, the execution host's MoM logs this event. The following is an example log entry:
 

```
08/21/2009 16:06:49;0028;pbs_mom;Job;2.host1;No Password Entry for User User1
```
- When a job triggers an action script for which the environment cannot be set up, perhaps due to a system error, the MoM attempting to run the action script logs the event. The log message contains the following:
 

```
:<job ID>;failed to setup dependent environment!
```
- When the scheduler attempts to run a job on an execution host where the job's owner does not have an account, the MoM on the execution host logs this event. The following is an example log entry:
 

```
08/21/2009 15:51:14;0028;pbs_mom;Job;1.host1;No Password Entry for User User1
```
- When the scheduler attempts to run a job on an execution host where the job's owner does not have a home directory, and when the job would use that home directory, the execution host's MoM logs this event. The log message contains the following:
 

```
Access from host not allowed, or unknown host: <numeric IP address>
```

See [“pbs\\_mom” on page 71 of the PBS Professional Reference Guide](#).

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).

### 11.13.1.1 Events Logged at Event Class 128 (0x0080)

The following security-related event is logged at event class 128 (0x0080):

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).
- If a user or Operator tries to set an attribute that can be set by Managers only, or attempts to create or delete vnodes:

The `qmgr` command returns this error message:

```
qmgr obj=<object> svr=default: Unauthorized Request
qmgr: Error (15007) returned from server
```

The server logs the following message:

```
Req;req_reject;Reject reply code=15007, aux=0, type=9, from <username>
```

- When a user is denied access to the server because of the contents of the `acl_users` server attribute, the server logs the following:

```
Req;req_reject;Reject reply code=15007, aux=0, type=21, from username@host.domain.com
```

### 11.13.1.2 Events Logged at Event Class 1

- When an attempt is made to contact MoM from a non-privileged port for a request requiring a privileged port, MoM logs the following:

```
pbs_mom;Svr;pbs_mom;Unknown error: 0 (0) in rm_request, bad attempt to connect message refused
from port 61558 addr 127.0.0.1
```

### 11.13.1.3 Events Not Logged

The following events are not logged:

- When an attempt is made to connect to a host in the PBS complex from a disallowed host
- When an ACL check denies an entity access to a PBS object
- A user tries to query other users' jobs when the server's `query_other_jobs` attribute is set to *False*
- When an Operator or Manager overrides the server's user ACL

## 11.14 Securing Containers

- Use the security enhancement named "pbs\_container"; see [section 7.4.4, “Configure Security Enhancement for Docker”, on page 365](#).
- Make sure that when you are configuring the container hook, if you whitelist any container arguments in the `container_args_allowed` hook configuration parameter, do not whitelist "--group-add". This would allow job submitters to add themselves to any groups inside the container. Instead, set the `enable_group_add_arg` hook parameter to *True* so the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs. See [section 7.4.2, “Configure PBS Container Hook”, on page 361](#).



# 12

# Accounting

## 12.1 The Accounting Log File

The PBS server automatically maintains an accounting log file on the server host only.

### 12.1.1 Name and Location of Accounting Log File

Accounting log files are written on the server host only.

The accounting log filename defaults to `PBS_HOME/server_priv/accounting/ccyyymmdd` where `ccyyymmdd` is the date.

You can place the accounting log files elsewhere by specifying the `-A` option on the `pbs_server` command line.

The argument to the `-A` option is the absolute path to the file to be written. If you specify a null string, the accounting log is not opened and no accounting records are recorded. For example, the following produces no accounting log:

```
pbs_server -A ""
```

### 12.1.2 Managing the Accounting Log File

If you use the default filename including the date, the server closes the file and opens a new file every day on the first write to the file after midnight.

If you use either the default file or a file named with the `-A` option, the server closes the accounting log upon daemon/service shutdown and reopens it upon daemon/service startup.

The server closes and reopens the account log file when it receives a `SIGHUP` signal. This allows you to rename the old log and start recording again on an empty file. For example, if the current date is February 9, 2015 the server will be writing in the file `20150209`. The following actions cause the current accounting file to be renamed `feb9` and the server to close the file and start writing a new `20150209`.

```
cd $PBS_HOME/server_priv/accounting
mv 20150209 feb9
kill -HUP <server PID>
```

### 12.1.3 Permissions for Accounting Log

The `PBS_HOME/server_priv/accounting` directory is owned by root, and has the following permissions:

```
drwxr-xr-x
```



---

## 12.2 Viewing Accounting Information

To see accounting information, you can do any of the following:

- Use the `tracejob` command to print out all accounting information for a job
- Use the `pbs-report` command to generate reports of accounting statistics from accounting files
- Use the PBS Works front-end tool called PBS Analytics
- Look at the accounting files using your favorite editor or viewer

### 12.2.1 Using the `tracejob` Command

You can use the `tracejob` command to extract the accounting log messages for a specific job and print the messages in chronological order. The `tracejob` command looks at all log files, so if you want to see accounting information only, use the `-l`, `-m`, and `-s` options to filter out scheduler, MoM, and server log messages. You can use `tracejob` to see information about a job that is running or has finished. For accounting information, use the `tracejob` command at the server host.

```
tracejob <job ID>
```

See [“`tracejob`” on page 238 of the PBS Professional Reference Guide](#) for details about using the `tracejob` command.

#### 12.2.1.1 Permissions for the `tracejob` Command

Root privilege is required when using `tracejob` to see accounting information.

## 12.3 Format of Accounting Log Messages

The PBS accounting log is a text file with each entry terminated by a newline. There is no limit to the size of an entry.

### 12.3.1 Log Entry Format

The format of a message is:

```
<logfile date and time>;<record type>;<ID string>;<message text>
```

where

*logfile date and time*

Date and time stamp in the format:

*mm/dd/yyyy hh:mm:ss*

*record type*

A single character indicating the type of record

*ID string*

The job or reservation identifier

*message text*

Message text format is blank-separated *keyword=value* fields.

Message text is ASCII text.

Content depends on the record type.

There is no dependable ordering of the content of each message.

There is no limit to the size of an entry.

## 12.3.2 Space Characters in String Entries

String entries in the accounting log may contain spaces. Under Linux, you must enclose any strings containing spaces with quotes.

Example 12-1: If the value of the `Account_Name` attribute is "*Power Users*", the accounting entry should look like this, either because you added the quotes or PBS did:

```
user=pbstest group=None account="Power Users"
```

### 12.3.2.1 Replacing Space Characters in String Entries

You can specify a replacement for the space character in any accounting string via the `-s` option to the `pbs_server` command by doing the following:

1. Bring up the *Services* dialog box
2. Select *PBS\_SERVER*
3. Stop the server
4. In the start parameters, use the `-s` option to specify the replacement
5. Start the server

Example 12-2: To replace space characters with "%20", bring up the server with "`-s %20`".

In this example, PBS replaces space characters in string entries with "%20":

```
user=pbstest group=None account=Power%20Users
```

If the first character of the replacement string argument to the `-s` option appears in the data string itself, PBS replaces that character with its hex representation prefixed by `%`.

Example 12-3: Given a percent sign in one of our string entries:

```
account=Po%wer Users
```

Since `%` appears in the data string and our replacement string is "%20", PBS replaces `%` with its hex representation (`%25`):

```
account="Po%25wer%20Users"
```

## 12.4 Types of Accounting Log Records

Accounting records for job arrays and subjobs are the same as for jobs, except that subjobs do not have Q (job entered queue) records. PBS writes different types of accounting records for different events. We list the record types, and describe the triggering event and the contents for each record type.

### A

Job was aborted by the server. The *message text* contains the explanation for why the job was aborted.

### a

Job was altered via `qalter` or a server hook. The message text is a list of <job attribute>=<new value> pairs, separated by spaces. This record shows only changes to a job attribute made via `qalter` or a server hook. This record does not show changes made by the server, a scheduler, or when MoM changes `resources_used`.

### B

Reservation record, written at the beginning of a reservation period, for all types of reservations. This record is written when the start time of a confirmed reservation is reached. Possible information includes the following:

**Table 12-1: B Record: Reservation Information**

| Entry                                | Explanation                                                                                                                                                                      |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Authorized_Groups= <groups>          | Groups who are and are not authorized to submit jobs to the reservation.                                                                                                         |
| Authorized_Hosts= <hosts>            | Hosts from which jobs may and may not be submitted to the reservation.                                                                                                           |
| Authorized_Users = <users>           | Users who are and are not authorized to submit jobs to the reservation.                                                                                                          |
| ctime= <creation time>               | Timestamp; time at which the reservation was created, in seconds since the epoch.                                                                                                |
| duration = <reservation duration>    | The duration specified or computed for the reservation, in seconds.                                                                                                              |
| end= <end of period>                 | Time at which the reservation period is to end, in seconds since the epoch.                                                                                                      |
| name= <reservation name>             | Reservation name, if reservation creator supplied a name string for the reservation.                                                                                             |
| nodes= <vnodes>                      | Contents of <code>resv_nodes</code> reservation attribute                                                                                                                        |
| owner= <reservation owner>           | Name of party who submitted the reservation request.                                                                                                                             |
| queue= <queue name>                  | Name of the reservation queue.                                                                                                                                                   |
| Resource_List= <requested resources> | List of resources requested by the reservation. Resources are listed individually, for example: <code>Resource_List.ncpus = 16</code> <code>Resource_List.mem = 1048676kb</code> |
| start= <start of period>             | Time at which the reservation period is to start, in seconds since the epoch.                                                                                                    |

### C

Job was checkpointed and requeued. Not written for a snapshot checkpoint, where the job continues to run.

When a job is checkpointed and requeued, PBS writes a C record in the accounting log. The C record contains the job's exit status. If a job is checkpointed and requeued, the exit status recorded in the accounting record is -12.

The C record is written for the following:

- Using the `qhold` command
- Checkpointing and aborting a job

**c**

After vnode release, the **c** record shows job information for the upcoming phase. See also the **u** record, which shows the phase that just finished, and the **e** record, which shows the final phase. The **c** record's *message text* field contains the following:

**Table 12-2: c Record: Upcoming Phase, After Vnode Release**

| Entry                                             | Explanation                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ctime=<creation time>                             | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                                                                                                                                                                                                |
| etime=<time when job became eligible to run>      | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.                                                                                                                                                                            |
| exec_host=<list of hosts and resources>           | List of job hosts with host-level, consumable resources allocated from each host. Format: <i>exec_host</i> =<host A>/<index>*<CPUs> [+<host B>/<index>*<CPUs>] where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted.                                                                                              |
| exec_vnode=<vnode_list>                           | List of job vnodes with vnode-level, consumable resources from each vnode. Format: (<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)<br><i>resource amount</i> is the amount of that resource allocated from that vnode.<br>Parentheses may be missing if the <i>exec_vnode</i> string was entered without them while executing <code>qrun -H</code> . |
| group= <group name>                               | Group name under which the job will execute.                                                                                                                                                                                                                                                                                                                                                             |
| jobname= <job name>                               | Name of the job.                                                                                                                                                                                                                                                                                                                                                                                         |
| project= <project name>                           | Job's project name at the start of the job.                                                                                                                                                                                                                                                                                                                                                              |
| qtime=<time>                                      | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                      |
| queue=<queue name>                                | The name of the queue in which the job resides.                                                                                                                                                                                                                                                                                                                                                          |
| resources_used.<resource name> = <resource value> | List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is <code>qrerun</code> , and only in the case where the job did start running.                                                                                                                                                                                            |
| Resource_List.<resource name>= <resource value>   | List of resources requested by the job. Resources are listed individually, for example:<br><code>Resource_List.ncpus =16</code><br><code>Resource_List.mem =1048676kb</code>                                                                                                                                                                                                                             |
| run_count=<count>                                 | The number of times the job has been executed.                                                                                                                                                                                                                                                                                                                                                           |
| session=<job session ID>                          | Session number of job.                                                                                                                                                                                                                                                                                                                                                                                   |
| start=<start time>                                | Time when job execution started, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                 |
| user=<username>                                   | The username under which the job will execute.                                                                                                                                                                                                                                                                                                                                                           |

**D**

Job or subjob was deleted by request. If a running job is discarded by PBS, PBS writes a D record, but not an E record. The *message text* contains `requestor=<username>@<hostname>` to identify who deleted the job. The D record is written for the following actions:

**Table 12-3: D Record Triggers**

| Action                                                                                                           | Result                                                 |
|------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| qdel a non-running job                                                                                           | Write record immediately                               |
| qdel -W force a job                                                                                              | Kill job, then write record after job is deleted       |
| qdel -W force a provisioning job                                                                                 | Kill job, then write record after job is deleted       |
| qdel a running job                                                                                               | Kill job, then write record when MoM gets back to us   |
| qrerun a job                                                                                                     | Kill job, then write record after job is queued        |
| qdel a subjob                                                                                                    | Kill subjob, then write record after subjob is deleted |
| PBS discards a running job, for example due to hardware failure, and <code>node_fail_requeue</code> is triggered | Write record and requeue job                           |

**E**

Job or subjob ended (terminated execution). If a running job is discarded by PBS, PBS writes a D record, but not an E record. The end-of-job accounting record is not written until all of the job's resources are freed. The E record is written for the following actions:

- When a job array finishes (all subjobs are in the X state). For example, [1] and [2] finish, but we delete [3] while it's running, so the job array [] gets an E record.
- When MoM sends an obit to the server

The E record can include the following:

**Table 12-4: E Record: Job End**

| Entry                                                            | Explanation                                                                                                                                                                                                                                |
|------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>account= &lt;account name&gt;</code>                       | Written if job has a value for its <code>Account_Name</code> attribute                                                                                                                                                                     |
| <code>accounting_id= &lt;JID value&gt;</code>                    | CSA JID, job container ID; value of job's <code>accounting_id</code> attribute                                                                                                                                                             |
| <code>alt_id= &lt;alternate job ID&gt;</code>                    | Optional alternate job identifier.                                                                                                                                                                                                         |
| <code>array_indices=&lt;array indices&gt;</code>                 | Array indices job array was submitted with, if this is a job array. Not reported for subjobs.                                                                                                                                              |
| <code>ctime= &lt;creation time&gt;</code>                        | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                                  |
| <code>eligible_time= &lt;eligible time&gt;</code>                | Amount of time job has waited while blocked on resources, in seconds.                                                                                                                                                                      |
| <code>end= &lt;job end time&gt;</code>                           | Time in seconds since epoch when this accounting record was written. Includes time to stage out files, delete files, and free job resources. The time for these actions is not included in the <code>walltime</code> recorded for the job. |
| <code>etime= &lt;time when job became eligible to run&gt;</code> | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.              |

Table 12-4: E Record: Job End

| Entry                                            | Explanation                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| exec_host= <list of hosts and resources>         | List of job hosts with host-level, consumable resources allocated from each host. Format: <i>exec_host</i> =<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>] where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted.                                                                                            |
| exec_vnode= <vnodes>                             | List of job vnodes with vnode-level, consumable resources from each vnode. Format: (<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)<br><i>resource amount</i> is the amount of that resource allocated from that vnode.<br>Parentheses may be missing if the <i>exec_vnode</i> string was entered without them while executing <code>qrun -H</code> . |
| Exit_status= <exit status>                       | The exit status of the job or subjob. See <a href="#">"Job Exit Status Codes" on page 470 in the PBS Professional Administrator's Guide</a> and <a href="#">"Job Array Exit Status", on page 160 of the PBS Professional User's Guide</a> .<br>The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.                                               |
| group= <group name>                              | Group name under which the job executed. This is the job's <code>egroup</code> attribute.                                                                                                                                                                                                                                                                                                                |
| jobname= <job name>                              | Name of the job                                                                                                                                                                                                                                                                                                                                                                                          |
| jobobit= <obit time>                             | Timestamp; time when server receives job or subjob obit from MoM on primary execution host, in seconds since epoch                                                                                                                                                                                                                                                                                       |
| pcap_accelerator                                 | Power cap for an accelerator. Corresponds to Cray <code>capmc set_power_cap --accel</code> setting.                                                                                                                                                                                                                                                                                                      |
| pcap_node                                        | Power cap for a node. Corresponds to Cray <code>capmc set_power_cap --node</code> setting.                                                                                                                                                                                                                                                                                                               |
| pgov                                             | Cray ALPS reservation setting for CPU throttling corresponding to p-governor.                                                                                                                                                                                                                                                                                                                            |
| project= <project name>                          | Job's project name when this record is written                                                                                                                                                                                                                                                                                                                                                           |
| qtime=<time>                                     | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                      |
| queue= <queue name>                              | Name of the queue from which the job executed                                                                                                                                                                                                                                                                                                                                                            |
| resources_used.<resource name>= <resource value> | Resources used by the job as reported by MoM. Typically includes <code>ncpus</code> , <code>mem</code> , <code>vmem</code> , <code>cput</code> , <code>walltime</code> , <code>cpupercent</code> . <code>walltime</code> does not include suspended time. Some resources get special reporting; see <a href="#">section 12.6.2.2, "Reporting Resources Used by Job", on page 554</a> .                   |
| Resource_List.<resource name>= <resource value>  | List of resources requested by the job. Resources are listed individually, for example: <code>Resource_List.ncpus =16</code> <code>Resource_List.mem =1048676kb</code>                                                                                                                                                                                                                                   |
| resvID=<reservation ID>                          | ID of reservation job is in, if any                                                                                                                                                                                                                                                                                                                                                                      |
| resvname=<reservation name>                      | Name of reservation job is in, if any                                                                                                                                                                                                                                                                                                                                                                    |
| run_count=<count>                                | The number of times the job has been executed.                                                                                                                                                                                                                                                                                                                                                           |
| session=<session ID>                             | Session number of job.                                                                                                                                                                                                                                                                                                                                                                                   |
| start=<start time>                               | Time when job execution started, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                 |
| user=<username>                                  | The username under which the job executed. This is the job's <code>euser</code> attribute.                                                                                                                                                                                                                                                                                                               |

**e**

For a job whose vnodes were released, the **e** record shows information for the final phase of the job, after vnodes were released. See also the **c** record, which shows info for an upcoming phase, and the **u** record, which shows info for the just-finished phase. The **e** record's *message text* field contains the following:

**Table 12-5: e Record: Final Phase After Vnode Release**

| Entry                                             | Explanation                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ctime=<creation time>                             | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                                                                                                                                                                                                |
| etime=<time when job became eligible to run>      | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.                                                                                                                                                                            |
| exec_host=<list of hosts and resources>           | List of job hosts with host-level, consumable resources allocated from each host. Format: <i>exec_host</i> =<host A>/<index>*<CPUs> [+<host B>/<index>*<CPUs>] where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted.                                                                                              |
| exec_vnode=<vnode_list>                           | List of job vnodes with vnode-level, consumable resources from each vnode. Format: (<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)<br><i>resource amount</i> is the amount of that resource allocated from that vnode.<br>Parentheses may be missing if the <i>exec_vnode</i> string was entered without them while executing <code>qrun -H</code> . |
| group= <group name>                               | Group name under which the job will execute.                                                                                                                                                                                                                                                                                                                                                             |
| jobname= <job name>                               | Name of the job.                                                                                                                                                                                                                                                                                                                                                                                         |
| project= <project name>                           | Job's project name at the start of the job.                                                                                                                                                                                                                                                                                                                                                              |
| qtime=<time>                                      | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                      |
| queue=<queue name>                                | The name of the queue in which the job resides.                                                                                                                                                                                                                                                                                                                                                          |
| resources_used.<resource name> = <resource value> | List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is <code>qrerun</code> , and only in the case where the job did start running.                                                                                                                                                                                            |
| Resource_List.<resource name>= <resource value>   | List of resources requested by the job. Resources are listed individually, for example:<br><code>Resource_List.ncpus =16</code><br><code>Resource_List.mem =1048676kb</code>                                                                                                                                                                                                                             |
| run_count=<count>                                 | The number of times the job has been executed.                                                                                                                                                                                                                                                                                                                                                           |
| session=<job session ID>                          | Session number of job.                                                                                                                                                                                                                                                                                                                                                                                   |
| start=<start time>                                | Time when job execution started, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                 |
| user=<username>                                   | The username under which the job will execute.                                                                                                                                                                                                                                                                                                                                                           |

**K**

Reservation deleted at request of scheduler or server. The *message text* field contains `requestor=Server@<hostname>` or `requestor=Scheduler@<hostname>` to identify who deleted the resource reservation.



**k**

Reservation terminated by owner issuing a `pbs_rdel` command. Written for all types of reservations. The *message text* field contains `requestor=<username>@<hostname>` to identify who deleted the reservation.

**L**

Information about node or socket licenses. Written when the server periodically reports license information from the license server. This line in the log has the following fields:

`<Log date>; <record type>; <keyword>; <specification for license>; <hour>; <day>; <month>; <max>`

The following table describes each field:

**Table 12-6: Licensing Information in Accounting Log**

| Field                            | Description                                                             |
|----------------------------------|-------------------------------------------------------------------------|
| <i>Log date</i>                  | Date of event                                                           |
| <i>record type</i>               | Indicates license info                                                  |
| <i>keyword</i>                   | license                                                                 |
| <i>specification for license</i> | Indicates that this is license info                                     |
| <i>hour</i>                      | Number of licenses used in the last hour                                |
| <i>day</i>                       | Number of licenses used in the last day                                 |
| <i>month</i>                     | Number of licenses used in the last month                               |
| <i>max</i>                       | Maximum number of licenses ever used. Not dependent on server restarts. |

**M**

Job move record. When a job or job array is moved to another server, PBS writes an M record containing the date, time, record type, job ID, and destination.

Example of an accounting log entry:

```
7/08/2008 16:17:38; M; 97.serverhost1.domain.com; destination=workq@serverhost2
```

When a job array has been moved from one server to another, the subjob accounting records are split between the two servers.

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the `qmove` command
- Job was submitted to a routing queue, then routed to a destination queue at another server

**P**

Provisioning starts for job or reservation.

Format: <Date and time>;<record type>;<job or reservation ID>; <message text>, where *message text* contains:

- Username
- Group name
- Job or reservation name
- Queue
- List of vnodes that were provisioned, with the AOE that was provisioned
- Provision event (START)
- Start time in seconds since epoch

Example:

```
"01/15/2009 12:34:15;P;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=START start_time=1231928746"
```

### P

Provisioning ends for job or reservation. Provisioning can end due to either a successful finish or failure to provision.

Format: <Date and time>;<record type>;<job or reservation ID>; <message text>, where *message text* contains:

- Username
- Group name
- Job or reservation name
- Queue
- List of vnodes that were provisioned, with the AOE that was provisioned
- Provision event (END)
- Provision status (SUCCESS or FAILURE)
- End time in seconds since epoch

Example printed when job stops provisioning:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=SUCCESS
end_time=1231928812"
```

Example printed when provisioning for job failed:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=FAILURE
end_time=1231928812"
```

### Q

Job entered a queue. Not written for subjobs. PBS writes a new Q record each time the job is routed or moved to a new queue or to the same queue.

The Q record can include the following:

**Table 12-7: Q Record: Job Queued**

| Entry                         | Explanation                                                                                   |
|-------------------------------|-----------------------------------------------------------------------------------------------|
| account= <account name>       | Written if job has a value for its Account_Name attribute                                     |
| accounting_id= <JID value>    | CSA JID, job container ID; value of job's accounting_id attribute                             |
| array_indices=<array indices> | Array indices job array was submitted with, if this is a job array. Not reported for subjobs. |

Table 12-7: Q Record: Job Queued

| Entry                                                        | Explanation                                                                                                                                                                                                                   |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ctime= <i>&lt;creation time&gt;</i>                          | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                     |
| depend= <i>&lt;dependencies&gt;</i>                          | The job's dependencies, if any                                                                                                                                                                                                |
| etime= <i>&lt;time when job became eligible to run&gt;</i>   | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering. |
| group= <i>&lt;group name&gt;</i>                             | Group name under which the job executed. This is the job's egroup attribute.                                                                                                                                                  |
| jobname= <i>&lt;job name&gt;</i>                             | Name of the job                                                                                                                                                                                                               |
| pcap_accelerator                                             | Power cap for an accelerator. Corresponds to Cray capmc set_power_cap --accel setting.                                                                                                                                        |
| pcap_node                                                    | Power cap for a node. Corresponds to Cray capmc set_power_cap --node setting.                                                                                                                                                 |
| pgov                                                         | Cray ALPS reservation setting for CPU throttling corresponding to p-governor.                                                                                                                                                 |
| project= <i>&lt;project name&gt;</i>                         | Job's project name when this record is written                                                                                                                                                                                |
| qtime= <i>&lt;time&gt;</i>                                   | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                           |
| queue= <i>&lt;queue name&gt;</i>                             | Name of the queue from which the job executed                                                                                                                                                                                 |
| Resource_List.<resource name>= <i>&lt;resource value&gt;</i> | List of resources requested by the job. Resources are listed individually, for example: Resource_List.ncpus =16 Resource_List.mem =1048676kb                                                                                  |
| resvID= <i>&lt;reservation ID&gt;</i>                        | ID of reservation job is in, if any                                                                                                                                                                                           |
| resvname= <i>&lt;reservation name&gt;</i>                    | Name of reservation job is in, if any                                                                                                                                                                                         |
| user= <i>&lt;username&gt;</i>                                | The username under which the job executed. This is the job's euser attribute.                                                                                                                                                 |

**R**

This job information written when:

- A job is rerun via qrerun or node\_fail\_requeue action
- MoM is restarted without the -p or -r options

Not written when job fails to start because the prologue rejects the job.

Possible information includes:

Table 12-8: R Record: Job Rerun

| Entry                                       | Explanation                                                                                      |
|---------------------------------------------|--------------------------------------------------------------------------------------------------|
| account= <i>&lt;account name&gt;</i>        | Written if job has an "account name" string                                                      |
| accounting_id= <i>&lt;JID value&gt;</i>     | CSA JID, job container ID                                                                        |
| alt_id= <i>&lt;alternate job ID&gt;</i>     | Optional alternate job identifier.                                                               |
| ctime= <i>&lt;creation time&gt;</i>         | Timestamp; time at which the job was created, in seconds since the epoch.                        |
| eligible_time= <i>&lt;eligible time&gt;</i> | Amount of time job has waited while blocked on resources, starting at creation time, in seconds. |

Table 12-8: R Record: Job Rerun

| Entry                                             | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| end= <time>                                       | Time in seconds since epoch when this accounting record was written. Includes time to delete files and free resources.                                                                                                                                                                                                                                                                                                                                          |
| etime= <time job became eligible to run>          | Timestamp; time in seconds since epoch when job most recently became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or when a job is held then released. Not affected by qaltering.                                                                                                                                                                                                          |
| exec_host= <list of hosts and resources>          | List of job hosts with host-level, consumable resources allocated from each host. Format: <code>exec_host=&lt;host A&gt;/&lt;index&gt;*&lt;CPUs&gt; [+&lt;host B&gt;/&lt;index&gt;*&lt;CPUs&gt;]</code> where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted.                                                                                                            |
| exec_vnode= <vnode_list>                          | List of job vnodes with vnode-level, consumable resources from each vnode. Format: <code>(&lt;vnode A&gt;:&lt;resource name&gt;=&lt;resource amount&gt;:...)+(&lt;vnode B&gt;:&lt;resource name&gt;=&lt;resource amount&gt;:...)</code><br><i>resource amount</i> is the amount of that resource allocated from that vnode.<br>Parentheses may be missing if the <code>exec_vnode</code> string was entered without them while executing <code>qrun -H</code> . |
| Exit_status= <exit status>                        | The exit status of the previous start of the job. See <a href="#">"Job Exit Status Codes" on page 470 in the PBS Professional Administrator's Guide</a> .<br>The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.                                                                                                                                                                                        |
| group=<group name>                                | The group name under which the job executed.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| jobname=<job name>                                | The name of the job.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| jobobit= <obit time>                              | Timestamp; time when server receives job or subjob obit from MoM on primary execution host, in seconds since epoch                                                                                                                                                                                                                                                                                                                                              |
| project=<project name>                            | The job's project name.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| qtime=<time job was queued>                       | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                                             |
| queue=<queue name>                                | The name of the queue in which the job is queued.                                                                                                                                                                                                                                                                                                                                                                                                               |
| Resource_List.<resource name>= <resource value>   | List of resources requested by the job. Resources are listed individually, for example: <code>Resource_List.ncpus =16 Resource_List.mem =1048676kb</code>                                                                                                                                                                                                                                                                                                       |
| resources_used.<resource name> = <resource value> | List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is <code>qrerun</code> , and only in the case where the job did start running.                                                                                                                                                                                                                                                   |
| run_count=<count>                                 | The number of times the job has been executed.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| session=<session ID>                              | Session ID of job.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| start=<start time>                                | Time when job execution started most recently, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                                                          |
| user=<username>                                   | The username under which the job executed.                                                                                                                                                                                                                                                                                                                                                                                                                      |

**r**

Job has been resumed. Format:

<logfile date and time>;<record type>;<job ID string>;

**S**

Job execution started. The *message text* field contains the following:

**Table 12-9: S Record: Job Start**

| Entry                                                | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| accounting_id= <accounting string>                   | An identifier that is associated with system-generated accounting data. In the case where accounting is CSA, <i>accounting string</i> is a job container identifier or JID created for the PBS job.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| array_indices=<array indices>                        | Array indices job array was submitted with, if this is a job array. Not reported for subjobs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| ctime= <creation time>                               | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| etime= <time when job became eligible to run>        | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| exec_host= <list of hosts and resources>             | List of job hosts with host-level, consumable resources allocated from each host. Format: <i>exec_host</i> =<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>] where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| group= <group name>                                  | Group name under which the job will execute.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| jobname= <job name>                                  | Name of the job.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| pcap_accelerator                                     | Power cap for an accelerator. Corresponds to Cray capmc set_power_cap --accel setting.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| pcap_node                                            | Power cap for a node. Corresponds to Cray capmc set_power_cap --node setting.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| pgov                                                 | Cray ALPS reservation setting for CPU throttling corresponding to p-governor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| project= <project name>                              | Job's project name at the start of the job.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| qtime= <time>                                        | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| queue= <queue name>                                  | The name of the queue in which the job resides.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| resources_assigned.<resource name>= <resource value> | <p><b>Not</b> a job attribute; simply a label for reporting job resource assignment. One resource per entry.</p> <p>Includes all allocated consumable resources. Consumable resources include <i>ncpus</i>, <i>mem</i> and <i>vmem</i> by default, and any custom resource defined with the <i>-n</i> or <i>-f</i> flags. A resource is not listed if the job does not request it directly or inherit it by default from queue or server settings.</p> <p>Actual amount of each resource assigned to the job by PBS. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs.</p> |
| Resource_List.<resource name>= <resource value>      | <p>List of resources requested by the job. Resources are listed individually, for example:</p> <pre>Resource_List.ncpus =16 Resource_List.mem =1048676kb</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| resvID=<reservation ID>                              | ID of reservation job is in, if any                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

Table 12-9: S Record: Job Start

| Entry                       | Explanation                                              |
|-----------------------------|----------------------------------------------------------|
| resvname=<reservation name> | Name of reservation job is in, if any                    |
| session= <job session ID>   | Session number of job.                                   |
| start=<start time>          | Time when job execution started, in seconds since epoch. |
| user= <username>            | The username under which the job will execute.           |

**S**

A job's vnode request was trimmed via `release_nodes()` in an `execjob_prologue` or `execjob_launch` hook. The job's `tolerate_node_failures` attribute must be set to *all* or *job\_start*. The s record shows the new trimmed vnode request. Format: <date> <time>;s;<job ID>;user=<job owner> group=<group> project=<project> jobname=<job name> queue=<queue name> ctime=<ctime> qtime=<qtime> etime=<etime> start=<start time> exec\_host=<exec\_host> exec\_vnode=<exec\_vnode> Resource\_List.<resource>=<value> Resource\_List.<resource>=<value>...Resource\_List.<resource>=<value>

**T**

Job was restarted from a checkpoint file.

**U**

Unconfirmed advance, standing, job-specific ASAP, or maintenance reservation requested.

- For an advance, job-specific ASAP, or maintenance reservation, the *message text* field has this format:

U:<reservation ID> requestor=<username>@<hostname>

- For a standing reservation, the *message text* field has this format:

U:<reservation ID>;requestor=<username>@<hostname> recurrence\_rule=<recurrence rule> timezone=<time-zone>

Example: "U;S56.exampleserver;requestor=pbsuser@exampleserver recurrence\_rule=FREQ=HOURLY;COUNT=2 timezone=Asia/Kolkata".

**u**

After vnode release, the u record shows job information for the phase that just finished. See also the [c](#) record, which shows the upcoming phase, and the [e](#) record, which shows the final phase. The u record's *message text* field contains the following:

Table 12-10: u Record: Phase Just Finished, After Vnode Release

| Entry                                        | Explanation                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ctime=<creation time>                        | Timestamp; time at which the job was created, in seconds since the epoch.                                                                                                                                                                                                                                                                            |
| etime=<time when job became eligible to run> | Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.                                                                                                                        |
| exec_host=<list of hosts and resources>      | List of job hosts with host-level, consumable resources allocated from each host. Format: <code>exec_host=&lt;host A&gt;/&lt;index&gt;*&lt;CPUs&gt; [+&lt;host B&gt;/&lt;index&gt;*&lt;CPUs&gt;]</code> where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted. |

**Table 12-10: u Record: Phase Just Finished, After Vnode Release**

| Entry                                             | Explanation                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| exec_vnode=<vnode_list>                           | List of job vnodes with vnode-level, consumable resources from each vnode. Format: (<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)<br><i>resource amount</i> is the amount of that resource allocated from that vnode.<br>Parentheses may be missing if the <code>exec_vnode</code> string was entered without them while executing <code>qrun -H</code> . |
| group= <group name>                               | Group name under which the job will execute.                                                                                                                                                                                                                                                                                                                                                                   |
| jobname= <job name>                               | Name of the job.                                                                                                                                                                                                                                                                                                                                                                                               |
| project= <project name>                           | Job's project name at the start of the job.                                                                                                                                                                                                                                                                                                                                                                    |
| qtime=<time>                                      | Timestamp; the time that the job entered the current queue, in seconds since epoch.                                                                                                                                                                                                                                                                                                                            |
| queue=<queue name>                                | The name of the queue in which the job resides.                                                                                                                                                                                                                                                                                                                                                                |
| resources_used.<resource name> = <resource value> | List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is <code>qrerun</code> , and only in the case where the job did start running.                                                                                                                                                                                                  |
| Resource_List.<resource name>= <resource value>   | List of resources requested by the job. Resources are listed individually, for example:<br><code>Resource_List.ncpus =16</code><br><code>Resource_List.mem =1048676kb</code>                                                                                                                                                                                                                                   |
| run_count=<count>                                 | The number of times the job has been executed.                                                                                                                                                                                                                                                                                                                                                                 |
| session=<job session ID>                          | Session number of job.                                                                                                                                                                                                                                                                                                                                                                                         |
| start=<start time>                                | Time when job execution started, in seconds since epoch.                                                                                                                                                                                                                                                                                                                                                       |
| user=<username>                                   | The username under which the job will execute.                                                                                                                                                                                                                                                                                                                                                                 |

**Y**

Reservation confirmed or reconfirmed. Written for all types of reservations.

- Advance, job-specific, or maintenance reservations:

When an advance, job-specific, or maintenance reservation is confirmed for the first time, the Y record has this format:

*Y; <reservation ID> requestor=<requestor>@<server> start=<requested start time> end=<requested end time> vnodes=(<allotted vnodes>)*

Example: "Y; R123.server requestor=Scheduler@svr start=1497264531 end=1497264651 nodes=(node1.ncpus=3)"

The Y record is written when an advance, job-specific, or maintenance reservation alter request is confirmed. The Y record has the same format as for the first time the reservation is confirmed, but the requested field(s) are updated with new value(s):

*Y; <reservation ID> requestor=<requestor>@<server> start=<(new/original) start time> end=<(new/original) end time> nodes=(<allotted vnodes>)*

Example: "Y; R123.server requestor=root@hostname start=1497264471 end=1497264651 nodes=(node1.ncpus=3)"

- Standing reservations:

When a standing reservation is confirmed for the first time, the Y record has this format:



---

*Y; <reservation ID> requestor=<requestor>@<server> start=<requested start time> end=<requested end time> nodes=(<allotted vnodes>) count=<count>*

The *nodes* field is specific to the first occurrence.

Example: "Y; R123.server requestor=Scheduler@svr start=1497264531 end=1497264651 nodes=(node1:ncpus=3) count=3"

The Y record is written when a standing reservation alter request is confirmed (the reservation is reconfirmed). The Y record has the same format as as for the first time a standing reservation is confirmed, but the requested field(s) are updated with the new value(s), and the index of the next occurrence is appended. The *nodes* field is specific to the occurrence altered:

*Y; <reservation ID> requestor=<requestor>@<server> start=<(new/original) start time> end=<(new/original) end time> nodes=(<allotted vnodes>) count=<count> index=<index of the altered occurrence>*

Example: "Y; R123.server requestor=root@hostname start=1497264471 end=1497264651 nodes=(node1:ncpus=3) count=3 index=1"

The *allotted vnodes* is the value of the `resv_nodes` reservation attribute.

The *count* is the value of the `reserve_count` reservation attribute.

## **Z**

Job has been suspended. The *message string* contains the following:

- Values for the job's `resources_used` attribute
- Values for the job's `resources_released` attribute. This attribute is populated only when the server's `restrict_res_to_release_on_suspend` attribute is set; see [section 5.9.6.2, "Job Suspension and Resource Usage", on page 247](#).

## **12.4.1 Accounting Records for Job Arrays**

Accounting records for job arrays and subjobs are the same as for jobs, except that subjobs do not have Q (job entered queue) records.

When a job array has been moved from one server to another, the subjob accounting records are split between the two servers.

When a job array goes from *Queued* to *Begin* state, we write one S for the whole job array.

The E record is written when a job array finishes. For example, [1] and [2] finish, but we delete [3] while it's running, so the job array gets an E record.

A job array is finished when all subjobs are in the X state.

## 12.5 Timeline for Accounting Messages

### 12.5.1 Timeline for Job Accounting Messages

The following is a timeline that shows when job and job array attributes and resources are recorded, and when they are written:

**Table 12-11: Timeline for Job Accounting Messages**

| Job/Job Array Lifecycle                                                                             | Accounting Record |
|-----------------------------------------------------------------------------------------------------|-------------------|
| Job is queued                                                                                       | Q                 |
| Job is moved                                                                                        | M                 |
| Application licenses are checked out                                                                |                   |
| Any required job-specific staging and execution directories are created                             |                   |
| PBS_JOBDIR and job's jobdir attribute are set to pathname of staging and execution directory        |                   |
| If necessary, MoM creates job execution directory                                                   |                   |
| MoM creates temporary directory                                                                     |                   |
| Files are staged in                                                                                 |                   |
| Just after job is sent to MoM                                                                       | S<br>T            |
| primary execution host MoM tells sister MoMs they will run job task(s)                              |                   |
| MoM sets TMPDIR, JOBDIR, and other environment variables in job's environment                       |                   |
| MoM performs hardware-dependent setup: The job's cpusets are created, ALPS reservations are created |                   |
| The job script starts                                                                               |                   |
| MoM runs user program                                                                               |                   |
| Job starts an MPI process on sister vnode                                                           |                   |
| Job is suspended                                                                                    | z                 |
| Job is resumed                                                                                      | r                 |
| The job script finishes                                                                             |                   |
| The obit is sent to the server                                                                      | E, e              |
| Any specified file staging out takes place, including stdout and stderr                             |                   |
| Files staged in or out are deleted                                                                  |                   |
| Any job-specific staging and execution directories are removed                                      |                   |
| The job's cpusets are destroyed                                                                     |                   |
| Job files are deleted                                                                               | E                 |
| Application licenses are returned to pool                                                           |                   |

**Table 12-11: Timeline for Job Accounting Messages**

| Job/Job Array Lifecycle                                                               | Accounting Record |
|---------------------------------------------------------------------------------------|-------------------|
| Job is aborted by server                                                              | A                 |
| Job is checkpointed and held                                                          | C                 |
| Job is deleted                                                                        | D                 |
| Periodic license information                                                          | L                 |
| Job is rerun via <code>qrerun</code> or <code>node_fail_requeue</code>                | R                 |
| Job is restarted from a checkpoint file                                               | T                 |
| <code>qdel</code> a subjob                                                            | D                 |
| <code>qrerun</code> a job                                                             | D                 |
| <code>qdel</code> a provisioning job with force                                       | D                 |
| <code>qdel</code> any job with force                                                  | D                 |
| <code>qdel</code> a running job: kill job, then write record when MoM gets back to us | D                 |
| <code>qdel</code> a non-running job: write record now                                 | D                 |
| Job array finishes                                                                    | E                 |
| Job or subjob obit is received by server                                              | E, R              |
| After a job is discarded after nodes go down on multi-vnode job                       | E                 |
| Job vnodes are released                                                               | C, U              |

## 12.5.2 Where Job Attributes are Recorded

Some accounting entries for job attributes have different names from their attributes. The following table shows the record(s) in which each job attribute is recorded and the entry name:

**Table 12-12: Job Attributes in Accounting Records**

| Job Attribute                        | Record     | Accounting Entry Name      |
|--------------------------------------|------------|----------------------------|
| <code>accounting_id</code>           | E, Q, R, S | <code>accounting_id</code> |
| <code>Account_Name</code>            | E, Q, R    | <code>account</code>       |
| <code>accrue_type</code>             |            |                            |
| <code>alt_id</code>                  | E, R       | <code>alt_id</code>        |
| <code>argument_list</code>           |            |                            |
| <code>array</code>                   |            |                            |
| <code>array_id</code>                |            |                            |
| <code>array_index</code>             |            |                            |
| <code>array_indices_remaining</code> |            |                            |
| <code>array_indices_submitted</code> | E, Q, S    | <code>array_indices</code> |

**Table 12-12: Job Attributes in Accounting Records**

| <b>Job Attribute</b> | <b>Record</b>       | <b>Accounting Entry Name</b> |
|----------------------|---------------------|------------------------------|
| array_state_count    |                     |                              |
| block                |                     |                              |
| Checkpoint           |                     |                              |
| comment              |                     |                              |
| create_resv_from_job |                     |                              |
| ctime                | c, E, e, Q, R, S, u | ctime                        |
| depend               | Q                   | depend                       |
| egroup               | c, E, e, Q, R, S, u | group                        |
| eligible_time        | E, R                | eligible_time                |
| Error_Path           |                     |                              |
| estimated            |                     |                              |
| etime                | c, E, e, Q, R, S, u | etime                        |
| euser                | c, E, e, Q, R, S, u | user                         |
| executable           |                     |                              |
| Execution_Time       |                     |                              |
| exec_host            | c, E, e, R, S, u    | exec_host                    |
| exec_vnode           | c, E, e, R, u       | exec_vnode                   |
| Exit_status          | E, R                | Exit_status                  |
| forward_x11_cookie   |                     |                              |
| forward_x11_port     |                     |                              |
| group_list           |                     |                              |
| hashname             |                     |                              |
| Hold_Types           |                     |                              |
| interactive          |                     |                              |
| jobdir               |                     |                              |
| Job_Name             | c, E, e, Q, R, S, u | jobname                      |
| jobobit              | E, R                | jobobit                      |
| Job_Owner            |                     |                              |
| job_state            |                     |                              |
| Join_Path            |                     |                              |
| Keep_Files           |                     |                              |
| Mail_Points          |                     |                              |
| Mail_Users           |                     |                              |

Table 12-12: Job Attributes in Accounting Records

| Job Attribute             | Record              | Accounting Entry Name |
|---------------------------|---------------------|-----------------------|
| mtime                     |                     |                       |
| no_stdio_sockets          |                     |                       |
| obittime                  |                     |                       |
| Output_Path               |                     |                       |
| pcap_accelerator          | E, Q, S             |                       |
| pcap_node                 | E, Q, S             |                       |
| pgov                      | E, Q, S             |                       |
| Priority                  |                     |                       |
| project                   | c, E, e, Q, R, S, u | project               |
| pstate                    |                     |                       |
| qtime                     | c, E, e, Q, R, S, u | qtime                 |
| queue                     | c, E, e, Q, R, S, u | queue                 |
| queue_rank                |                     |                       |
| queue_type                |                     |                       |
| release_nodes_on_stageout |                     |                       |
| Remove_Files              |                     |                       |
| Rerunable                 |                     |                       |
| resources_released        | z                   | resources_released    |
| resources_used            | c, E, e, R, u, z    | resources_used        |
| Resource_List             | c, E, e, Q, R, S, u | Resource_List         |
| resource_released_list    |                     |                       |
| run_count                 | c, E, e, R, u       | run_count             |
| run_version               |                     |                       |
| sandbox                   |                     |                       |
| schedselect               |                     |                       |
| sched_hint                |                     |                       |
| server                    |                     |                       |
| session_id                | c, E, e, R, S, u    | session               |
| Shell_Path_List           |                     |                       |
| stagein                   |                     |                       |
| stageout                  |                     |                       |
| Stageout_status           |                     |                       |
| stime                     | c, E, e, R, S, u    | start                 |

**Table 12-12: Job Attributes in Accounting Records**

| Job Attribute          | Record | Accounting Entry Name |
|------------------------|--------|-----------------------|
| Submit_arguments       |        |                       |
| substate               |        |                       |
| sw_index               |        |                       |
| tolerate_node_failures |        |                       |
| topjob_ineligible      |        |                       |
| umask                  |        |                       |
| User_List              |        |                       |
| Variable_List          |        |                       |

### 12.5.3 Timeline for Reservation Accounting Messages

The following table shows the timeline for when reservation accounting messages are recorded:

**Table 12-13: Timeline for Reservation Accounting Messages**

| Reservation Lifecycle                                    | Accounting Record |
|----------------------------------------------------------|-------------------|
| Unconfirmed reservation is created                       | U                 |
| Reservation is confirmed                                 | Y                 |
| Reservation period begins                                | B, Y              |
| Provisioning for reservation begins                      | P                 |
| Provisioning for reservation ends                        | p                 |
| Reservation period ends                                  | F, Y              |
| Reservation is altered                                   | Y                 |
| Reservation is deleted by scheduler or server            | K                 |
| Reservation is deleted by user via <code>pbs_rdel</code> | k                 |

### 12.5.4 Where Reservation Attributes and Info are Recorded

Some accounting entries for reservation attributes have names that are different from their attributes. The following table shows the record(s) in which each reservation attribute is recorded and the entry name:

**Table 12-14: Reservation Attributes/Info in Accounting Records**

| Reservation Attribute/Info | Record | Accounting Entry Name |
|----------------------------|--------|-----------------------|
| Account_Name               |        |                       |
| Authorized_Groups          | B      | Authorized_Groups     |
| Authorized_Hosts           | B      | Authorized_Hosts      |

**Table 12-14: Reservation Attributes/Info in Accounting Records**

| <b>Reservation Attribute/Info</b> | <b>Record</b>             | <b>Accounting Entry Name</b> |
|-----------------------------------|---------------------------|------------------------------|
| Authorized_Users                  | B                         | Authorized_Users             |
| ctime                             | B                         | ctime                        |
| delete_idle_time                  |                           |                              |
| group_list                        |                           |                              |
| hashname                          |                           |                              |
| interactive                       |                           |                              |
| Mail_Points                       |                           |                              |
| Mail_Users                        |                           |                              |
| mtime                             |                           |                              |
| Priority                          |                           |                              |
| queue                             | B                         | queue                        |
| reservation requestor             | K, k, U, Y                | requestor                    |
| reserve_count                     | Y                         |                              |
| reserve_duration                  | B                         | duration                     |
| reserve_end                       | B, Y                      | end                          |
| reserve_ID                        | Q                         | resvID                       |
| reserve_index                     |                           |                              |
| reserve_job                       |                           |                              |
| Reserve_Name                      | B,<br>Q                   | name<br>resvname             |
| Reserve_Owner                     | B                         | owner                        |
| reserve_retry                     |                           |                              |
| reserve_rrule                     | U (standing reservations) | recurrence_rule              |
| reserve_start                     | B, Y                      | start                        |
| reserve_state                     |                           |                              |
| reserve_substate                  |                           |                              |
| reserve_type                      |                           |                              |
| Resource_List                     | B                         | Resource_List                |
| resv_nodes                        | B, Y                      | nodes                        |
| server                            |                           |                              |
| timezone                          | U (standing reservation)  | timezone                     |
| User_List                         |                           |                              |
| Variable_List                     |                           |                              |



### 12.5.4.1 Jobs in Reservations

- The job's queue is recorded in its c, E, e, Q, R, S, and u records.
- The job's reservation name and reservation ID are written in its E, Q, and S records.

## 12.5.5 How MoM Polling Affects Accounting

MoM periodically polls for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

Example 12-4: MoM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "`qsub -me`" was used in job submission) contains usage through T20.

The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.

If a job ends between MoM poll cycles, `resources_used.<resource name>` numbers will be slightly lower than they are in reality. For long-running jobs, the error percentage will be minor.

See [section 3.1.2, “Configuring MoM Polling Cycle”, on page 38](#) for details about MoM's polling cycle.

## 12.6 Resource Accounting

Job resources are recorded in the Q (job queued; `Resource_List` only), S (job start), R (job rerun), E (job end), c (upcoming phase when job vnodes are released), u (just-finished phase when job vnodes are released), s (vnodes trimmed), e (usage during post-release phase), and z (job suspension) records.

Reservation resources are recorded in the B (reservation start) record.

### 12.6.1 Accounting Log Resource Entry Formats

When reporting resources in the accounting B, c, E, e, R, S, or u records, there is one entry per resource. Each resource is reported on a separate line.

Values for requested resources are written in the same units as those in the resource requests. Values for `resources_used` and `resources_assigned` are written in *kb*. A suffix is always written unless the quantity is measured in bytes.

## 12.6.2 Job Resource Accounting

The following table shows which job and reservation resources are recorded in the accounting log, and lists the records where they are recorded:

**Table 12-15: Job Resources in Accounting Log**

| Resources          | Record                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Resource_List      | c<br>E<br>e<br>Q<br>R<br>S<br>u | List of resources requested by the job. Resources are listed individually, for example:<br><br>Resource_List.ncpus =16<br>Resource_List.mem =1048676kb                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| resources_assigned | S                               | <b>Not</b> a job attribute; simply a label for reporting job resource assignment. One resource per entry.<br><br>Includes all allocated consumable resources. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the <i>-n</i> or <i>-f</i> flags. A resource is not listed if the job does not request it directly or inherit it by default from queue or server settings.<br><br>Actual amount of each resource assigned to the job by PBS. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs. |
| resources_released | z                               | Listed by vnode, consumable resources that were released when the job was suspended.<br><br>Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Table 12-15: Job Resources in Accounting Log

| Resources      | Record                                   | Description                                                                                                                                                                                                                                                                                                                                     |
|----------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| resources_used | c<br>E<br>e<br>R (when qrerun)<br>u<br>z | Resources used by the job as reported by MoM. Typically includes ncpus, mem, vmem, cput, walltime, cpupercent. walltime does not include suspended time.                                                                                                                                                                                        |
| exec_host      | c<br>E<br>e<br>R<br>S<br>u               | List of job hosts with host-level, consumable resources allocated from each host. Format: <i>exec_host=&lt;host A&gt;/&lt;index&gt;*&lt;CPUs&gt; [+&lt;host B&gt;/&lt;index&gt;* &lt;CPUs&gt;]</i> where <i>index</i> is task slot number starting at 0, on that host, and <i>CPUs</i> is the number of CPUs assigned to the job, 1 if omitted. |
| exec_vnode     | c<br>E<br>e<br>R<br>u                    | List of job vnodes with vnode-level, consumable resources from each vnode. Format: ( <i>&lt;vnode A&gt;:&lt;resource name&gt;=&lt;resource amount&gt;:...</i> )+(<vnode B>:<resource name>=<resource amount>:...) <i>resource amount</i> is the amount of that resource allocated from that vnode.                                              |

### 12.6.2.0.i Accounting Log Entries for min\_walltime and max\_walltime

The Resource\_List job attribute contains values for min\_walltime and max\_walltime. For example, if the following job is submitted:

```
qsub -l min_walltime="00:01:00",max_walltime="05:00:00" -l select=2:ncpus=1 job.sh
```

This is the resulting accounting record:

```
...S..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18 resources_assigned.ncpus=2
...R..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18
...E..... Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18.....
```

### 12.6.2.1 Reporting Resources Assigned to Job

The value reported in the resources\_assigned accounting entry is the amount assigned to a job or that a job prevents other jobs from using, which is different from the amount the job requested or used. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs. In this example, resources\_assigned reports 4 CPUs, and resources\_used reports 1 CPU.

The resources\_assigned accounting entry is reported in the S record.

### 12.6.2.2 Reporting Resources Used by Job

Consumable job resources actually used by the job are recorded in the job's `resources_used` attribute. Values for `resources_used` are reported in the `c`, `E`, `e`, `u`, and `z` records, and the `R` record if the job is rerun, but not when the server loses contact with the primary execution host MoM.

You can use hooks to set values for a job's `resources_used` attribute for custom resources. These custom resources will appear in the accounting log, along with custom resources that are created or set in hooks. Other custom resources will not appear in the accounting log. See ["Setting Job Resources in Hooks" on page 50 in the PBS Professional Hooks Guide](#).

PBS reports `resources_used` values for string resources that are created or set in a hook as JSON strings in the `E` record.

- If MoM returns a JSON object (a Python dictionary), PBS reports it in the `E` record in single quotes:  
`resources_used.<resource_name> = '{ <MoM JSON item value>, <MoM JSON item value>, <MoM JSON item value>, ..}'`

Example: MoM returns `{ "a":1, "b":2, "c":1,"d": 4}` for `resources_used.foo_str`. We get:

```
resources_used.foo_str='{ "a": 1, "b": 2, "c":1,"d": 4}'
```

- If MoM returns a value that is not a JSON object, it is reported verbatim in the `E` record.

Example: MoM returns "hello" for `resources_used.foo_str`. We get:

```
resources_used.foo_str="hello"
```

### 12.6.2.3 Freeing Resources

The resources allocated to a job from vnodes are not released until all of those resources have been freed by all MoMs running the job. The end of job accounting record is not written until all of the resources have been freed. The `end` entry in the job `E` record includes the time to stage out files, delete files, and free the resources. This does not change the recorded `walltime` for the job.

### 12.6.2.4 Releasing Vnodes

When a job's vnodes are released via `pbs_release_nodes`, PBS writes the `c` and `u` records, and at the end of the job, PBS writes the `e` record.

If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.

## 12.6.3 Reservation Resource Accounting

The following table shows which reservation resources are recorded in the accounting log, and lists the records where they are recorded:

**Table 12-16: Reservation Resources in Accounting Log**

| Resources     | Record | Description                                                                                                                                                         |
|---------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| exec_host     | B      | List of vnodes allocated to the reservation                                                                                                                         |
| Resource_List | B      | List of resources requested by the reservation. Resources are listed individually as, for example:<br><br>Resource_List.ncpus = 16<br>Resource_List.mem = 1048676kb |
| resv_nodes    | B      | Contents of resv_nodes reservation attribute                                                                                                                        |

## 12.6.4 Changing Resource Values Reported in Accounting Logs

You can use an execution hook to set a value for `resources_used`, and this value is then recorded in the accounting log. Bear in mind that by the time an `execjob_end` hook runs, it's too late to change the accounting log; it's already written.

## 12.7 Options, Attributes, and Parameters Affecting Accounting

### 12.7.1 Options to `pbs_server` Command

**-A <accounting file>**

Specifies an absolute path name for the file to use as the accounting file. If not specified, the file is named for the current date in the `PBS_HOME/server_priv/accounting` directory.

Format: *String*

### 12.7.2 Options to `qsub` Command

**-A <accounting string>**

Accounting string associated with the job. Used for labeling accounting data and/or fairshare. Sets job's `Account_Name` attribute to <accounting string>.

Format: *String*

**-W release\_nodes\_on\_stageout=<value>**

When set to *True*, all of the job's vnodes are released when stageout begins.

When the `cgroups` hook is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a `cgroup` are not released until the entire `cgroup` is released.

The job's `stageout` attribute must be set for the `release_nodes_on_stageout` attribute to take effect.

## 12.7.3 Options to `qalter` Command

### `-A <new accounting string>`

Replaces the accounting string associated with the job. Used for labeling accounting data and/or fairshare. Sets job's `Account_Name` attribute to `<new accounting string>`. This attribute cannot be altered once the job has begun execution.

Format: *String*

### `-W release_nodes_on_stageout=<value>`

When set to *True*, all of the job's vnodes are released when stageout begins.

When cgroups is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

The job's `stageout` attribute must be set for the `release_nodes_on_stageout` attribute to take effect.

## 12.7.4 Job Attributes

### `Account_Name`

PBS jobs have an attribute called `Account_Name`. You can use it however you want; PBS does not interpret it.

PBS does not use this accounting string by default. However, you can tell PBS to use the job's accounting string as the owner of the job for fairshare purposes. See [section 4.9.19, “Using Fairshare”, on page 139](#). PBS accepts the string passed by the shell as is.

Any character is allowed if the string is enclosed in single or double quotes. When you specify this string on the command line to a PBS utility or in a directive in a PBS job script, escape any embedded white space by enclosing the string in quotes.

You can set the initial value of a job's `Account_Name` attribute via the `-A <account string>` option to `qsub`. You can change the value of a job's `Account_Name` attribute via the `-A <new account string>` option to `qalter`.

Can be read and set by user, Operator, Manager.

Format: String that can contain any character

Default value: none.

Python attribute value type: *str*

### `accounting_id`

Accounting ID for tracking accounting data not produced by PBS. May be used for a CSA job ID or job container ID.

Can be read by User, Operator, Manager.

No default value.

Format: *String*

Python attribute value type: *str*

**alt\_id**

For a few systems, the session ID is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it is recorded in the end-of-job accounting record. May contain white spaces.

On Windows, holds PBS home directory.

Can be read by User, Operator, Manager.

No default value.

Format: *String*

Python attribute value type: *str*

**release\_nodes\_on\_stageout**

When set to *True*, all of the job's vnodes are released when stageout begins.

When the cgroups hook is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

## 12.7.5 MoM Parameters

**\$logevent <mask>**

Sets the mask that determines which event types are logged by `pbs_mom`. To include all debug events, use `0xffffffff`. See [“Log Levels” on page 429 of the PBS Professional Reference Guide](#).

## 12.8 Accounting Caveats and Advice

If you use the cgroups hook to manage subsystems and create child vnodes, you get accurate accounting. If not, accuracy depends on whether or not your MPI is integrated with PBS.

### 12.8.1 Integrate MPIs for Accurate Accounting

PBS Professional is integrated with several implementations of MPI. When PBS is integrated with an MPI, PBS can track resource usage, control jobs, clean up job processes, and perform accounting for all of the tasks run under the MPI.

When PBS is not integrated with an MPI, PBS can track resource usage, clean up processes, and perform accounting only for processes running on the primary host. This means that accounting and tracking of CPU time and memory aren't accurate, and job processes on sister hosts cannot be signaled.

Follow the steps in [section 13.1, “Integration with MPI”, on page 559](#).

### 12.8.2 MPI Integration under Windows

Under Windows, some MPIs such as MPICH are not integrated with PBS. With non-integrated MPIs, PBS is limited to tracking resources, signaling jobs, and performing accounting only for job processes on the primary vnode.

### 12.8.3 Using Hooks for Accounting

#### 12.8.3.1 Use Hooks to Record Job Information

For each job, you can use `execjob_prologue`, `execjob_epilogue`, or `exechost_periodic` hooks to set `resources_used` values for custom resources, which are then recorded in the job's E record.



---

### 12.8.3.2 Use Hooks to Manage Job Accounting String

You can use a hook to assign the correct value for each job's `Account_Name` attribute. This can be useful both for your accounting records and if you use the job's `Account_Name` attribute as the job's owner for fairshare. See the PBS Professional Hooks Guide and [section 4.9.19, “Using Fairshare”, on page 139](#).

# 13

# Using MPI with PBS

## 13.1 Integration with MPI

PBS Professional is integrated with several implementations of MPI. When PBS is integrated with an MPI, PBS can track resource usage, control jobs, clean up job processes, perform accounting for all of the tasks run under the MPI, and create TMPDIR on each of the job's hosts.

When PBS is not integrated with an MPI, PBS can track resource usage, clean up processes, and perform accounting only for processes running on the primary host. This means that accounting and tracking of CPU time and memory aren't accurate, and job processes on sister hosts cannot be signaled.

## 13.2 Prerequisites

Before you integrate an MPI with PBS, the MPI must be working by itself. For example, you must make sure that all required environment variables are set correctly for the MPI to function.

## 13.3 Types of Integration

PBS provides support for integration for many MPIs. You can integrate MPIs with PBS using the following methods:

- Intel MPI 4.0.3 on Linux uses `pbs_tmsh` when it sees certain environment variables set. No other steps are required. See [section 13.5, “Integrating Intel MPI 4.0.3 On Linux Using Environment Variables”, on page 561](#).
- Wrapping the MPI with a PBS-supplied script which uses the TM (task manager) interface to manage job processes. PBS supplies a master script to wrap any of several MPIs. See [section 13.10, “Integration by Wrapping”, on page 563](#).
- PBS supplies wrapper scripts for some MPIs, for wrapping those MPIs by hand. See [section 13.13, “Integration By Hand”, on page 568](#).
- For non-integrated MPIs, job scripts can integrate the MPIs on the fly using the `pbs_tmsh` command. Note that a PBS job script that uses `mpirun` with `pbs_tmsh` cannot be used outside of PBS. See [section 13.9, “Integration on the Fly using the pbs\\_tmsh Command”, on page 562](#) and [“Integrating an MPI on the Fly”, on page 85 of the PBS Professional User’s Guide](#).
- Some MPIs can be compiled to use the TM interface. See [section 13.8, “Integration Using the TM Interface”, on page 562](#).
- Some MPIs require users to call `pbs_attach`. See [section 13.13.5, “Integrating HPE MPI”, on page 571](#).
- Altair support can help integrate your MPI with PBS so that the MPI always calls `pbs_attach` when it calls `ssh`. If you would like to use this method, contact Altair support at [www.pbsworks.com](http://www.pbsworks.com).

To integrate an MPI with PBS, you use just one of the methods above. The method you choose depends on the MPI. The following table lists the supported MPIs, how to integrate them, and gives links to the steps involved and any special notes about that MPI:

**Table 13-1: List of Supported MPIs**

| MPI Name                   | Versions                                                         | Method                                               | Integration Steps                                                                  | MPI-specific Notes                                    |
|----------------------------|------------------------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------------------------|-------------------------------------------------------|
| HP MPI                     | 1.08.03<br>2.0.0                                                 | Use <code>pbs_mpihp</code>                           | <a href="#">"Steps to Integrate HP MPI or Platform MPI"</a>                        | <a href="#">"Integrating HP MPI and Platform MPI"</a> |
| Intel MPI                  | 4.0.3 on Linux                                                   | Set environment variables                            | <a href="#">"Integrating Intel MPI 4.0.3 On Linux Using Environment Variables"</a> | None                                                  |
| Intel MPI                  | 4.0.3 on Windows                                                 | Use wrapper script                                   | <a href="#">"Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script"</a>      | None                                                  |
| Intel MPI                  | 2.0.022<br>3<br>4                                                | Use <code>pbsrun_wrap</code> (wrapper is deprecated) | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |
| MPICH-P4                   | 1.2.5<br>1.2.6<br>1.2.7<br><b>Support for all is deprecated.</b> | Use <code>pbs_mpirun</code>                          | <a href="#">"Steps to Integrate MPICH-P4"</a>                                      | <a href="#">"Integrating MPICH-P4"</a>                |
| MPICH-GM (MPICH 1.2.6.14b) | <b>Support for wrapper is deprecated.</b>                        | Use <code>pbsrun_wrap</code>                         | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |
| MPICH-MX                   | <b>Support for wrapper is deprecated.</b>                        | Use <code>pbsrun_wrap</code>                         | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |
| MPICH2                     | 1.0.3<br>1.0.5<br>1.0.7<br>on Linux                              | Use <code>pbsrun_wrap</code>                         | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |
| MPICH2                     | 1.4.1p1 on Windows                                               | Use wrapper script                                   | <a href="#">"Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script"</a>       | None                                                  |
| MVAPICH                    | 1.2<br><b>Support for wrapper is deprecated.</b>                 | Use <code>pbsrun_wrap</code>                         | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |
| MVAPICH2                   | 1.8                                                              | Use <code>pbsrun_wrap</code>                         | <a href="#">"Wrapping an MPI Using the pbsrun_wrap Script"</a>                     | <a href="#">"Integration by Wrapping"</a>             |

Table 13-1: List of Supported MPIs

| MPI Name     | Versions | Method                                                                                                        | Integration Steps                                           | MPI-specific Notes                                    |
|--------------|----------|---------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------|
| Open MPI     | 1.4.x    | Compile with TM                                                                                               | <a href="#">"Integration Using the TM Interface"</a>        | <a href="#">"Integrating Open MPI"</a>                |
| Platform MPI | 8.0      | Use <code>pbs_mpihp</code>                                                                                    | <a href="#">"Steps to Integrate HP MPI or Platform MPI"</a> | <a href="#">"Integrating HP MPI and Platform MPI"</a> |
| HPE MPI      | Any      | Optional: Use <code>mpiexec</code> , or users put <code>pbs_attach</code> in <code>mpirun</code> command line | <a href="#">"Steps to Integrate HPE MPI"</a>                | <a href="#">"Integrating HPE MPI"</a>                 |

## 13.4 Transparency to the User

Many MPIs can be integrated with PBS in a way that is transparent to the job submitter. This means that a job submitter can use the same MPI command line inside and outside of PBS. All of the MPIs listed above can be made to be transparent.

## 13.5 Integrating Intel MPI 4.0.3 On Linux Using Environment Variables

You can allow Intel MPI 4.0.3 to automatically detect when it runs inside a PBS job and use `pbs_tmsh` to integrate with PBS. When it has detected that it is running in a PBS job, it uses the hosts allocated to the job.

On hosts running Intel MPI 4.0.3 that have `PBS_EXEC/bin` in the default PATH, set the following environment variables in `PBS_HOME/pbs_environment`:

```
I_MPI_HYDRA_BOOTSTRAP=rsh
I_MPI_HYDRA_BOOTSTRAP_EXEC=pbs_tmsh
```

On hosts running Intel MPI 4.0.3 that do not have `PBS_EXEC/bin` in their default PATH, use the full path to `pbs_tmsh`. For example:

```
I_MPI_HYDRA_BOOTSTRAP_EXEC=/opt/pbs/bin/pbs_tmsh
```

The default process manager for Intel MPI 4.0.3 is Hydra.

### 13.5.1 Restrictions for Intel MPI 4.0.3

The unwrapped version of Intel MPI 4.0.3 `mpirun` on Linux does not support MPD.

## 13.6 Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script

This version of PBS provides a wrapper script for Intel MPI 4.0.3 on Windows. The wrapper script is named `pbs_intelmpi_mpirun.bat`, and it is located in `$PBS_EXEC/bin`. This script uses `pbs_attach` to attach MPI tasks to a PBS job. You do not need to take any steps to integrate Intel MPI on Windows; job submitters must call the wrapper script inside their job scripts.

## 13.7 Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script

This version of PBS provides a wrapper script for MPICH2 1.4.1p1 on Windows. The wrapper script is named `pbs_mpich2_mpirun.bat`, and it is located in `$PBS_EXEC/bin`. This script uses `pbs_attach` to attach MPI tasks to a PBS job. You do not need to take any steps to integrate Intel MPI on Windows; job submitters must call the wrapper script inside their job scripts.

## 13.8 Integration Using the TM Interface

PBS provides an API to the PBS task manager, or TM, interface. You can configure an MPI to use the PBS TM interface directly.

When a job process is started on a sister host using the TM interface, the sister host's MoM starts the process and the primary host's MoM has access to job process information.

An MPI that we know can be compiled with the TM interface is Open MPI.

## 13.9 Integration on the Fly using the `pbs_tmrsh` Command

If using a non-integrated MPI, job submitters can integrate an MPI on the fly by using the `pbs_tmrsh` command. This command emulates `rsh`, but uses the TM interface to talk directly to `pbs_mom` on sister hosts. The `pbs_tmrsh` command informs the primary and sister MoMs about job processes on sister hosts. PBS can track resource usage for all job processes.

Job submitters use this command by setting the appropriate environment variable to `pbs_tmrsh`. For example, to integrate MPICH, set `P4_RSHCOMMAND` to `pbs_tmrsh`. For details, see ["Integrating an MPI on the Fly", on page 85 of the PBS Professional User's Guide](#).

The following figure illustrates how the `pbs_tmsh` command can be used to integrate an MPI on the fly:

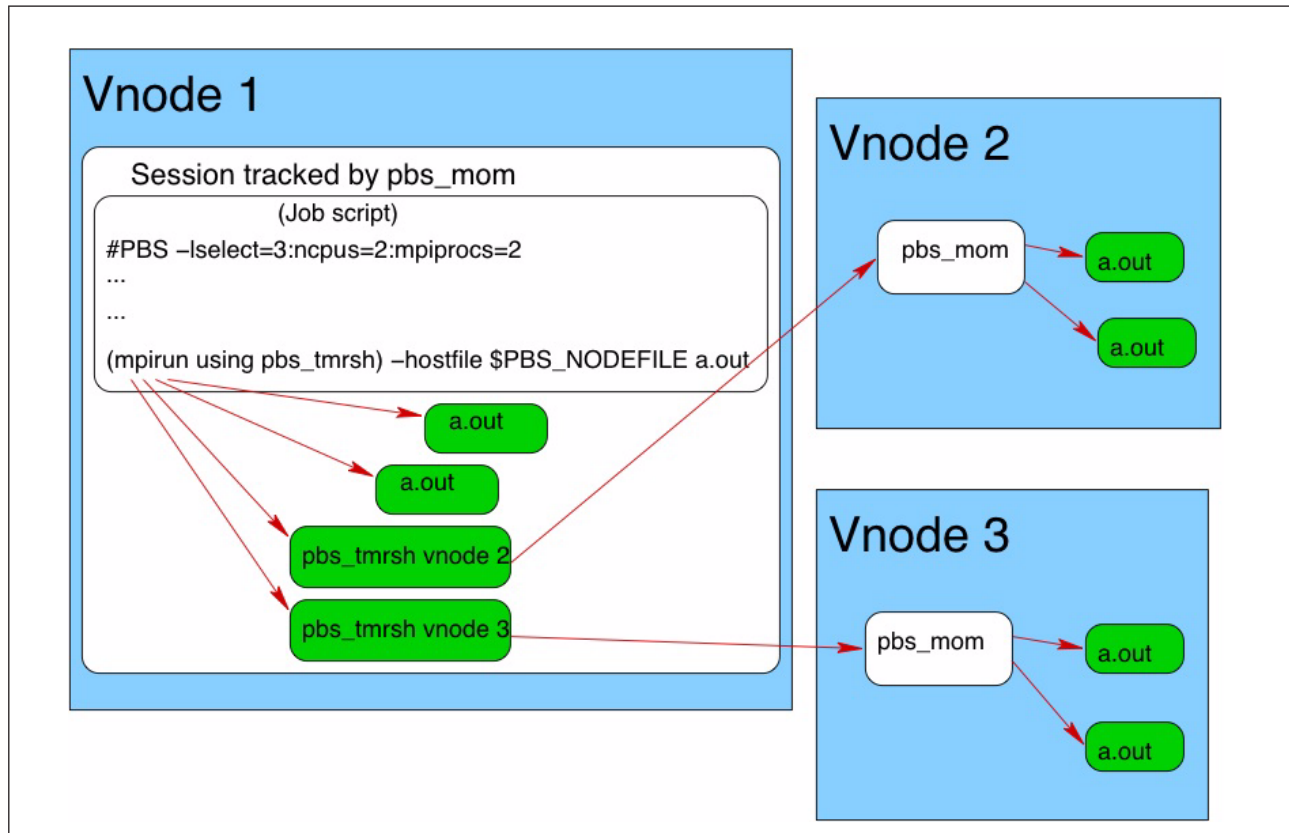


Figure 13-1: PBS knows about processes on vnodes 2 and 3, because `pbs_tmsh` talks directly to `pbs_mom`, and `pbs_mom` starts the processes on vnodes 2 and 3

### 13.9.1 Caveats for the `pbs_tmsh` Command

- This command cannot be used outside of a PBS job; if used outside a PBS job, this command will fail.
- The `pbs_tmsh` command does not perform exactly like `rsh`. For example, you cannot pipe output from `pbs_tmsh`; this will fail.

## 13.10 Integration by Wrapping

Wrapping an MPI means replacing its `mpirun` or `mpiexec` with a script which calls the original executable and, indirectly, `pbs_attach`. Job processes are started by `rsh` or `ssh`, but the `pbs_attach` command informs the primary and sister MoMs about the processes, so that PBS has control of the job processes. See [“pbs\\_attach” on page 56 of the PBS Professional Reference Guide](#).

PBS provides a master script called `pbsrun_wrap` that you use to wrap many MPIs. PBS supplies special wrapper scripts so that you can wrap other MPIs by hand.

The following figure shows how a wrapped `mpirun` call works:

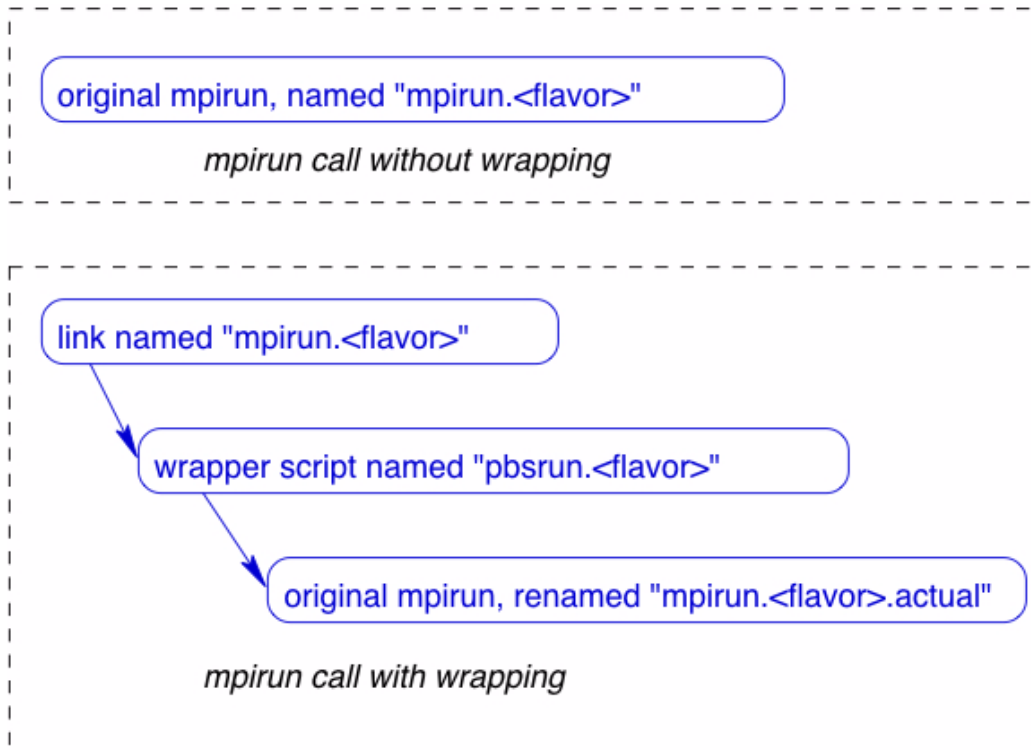
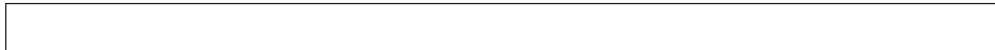


Figure 13-2: The job script calls the link that has the name of the original `mpirun`

### 13.10.1 Wrap the Correct Instance

Figure 13-3: The job script calls the link that has the name of the original `mpirun`



When you wrap an MPI, make sure that you are wrapping the first instance of the name found in the user's search path. This is the one returned by the 'which' command on Linux.

For example, on our example system `my_mpi` is installed as follows:

```
rw-rw-rw- 1 root system 31 Apr 18 19:21 /usr/bin/my_mpi -> /usr/my_mpi_dir/bin/my_mpi
```

And 'which' returns the following:

```
bash-2.05b# which my_mpi
/usr/bin/my_mpi
```

Here, you must wrap the link, not the binary.



## 13.11 Wrapping an MPI Using the `pbsrun_wrap` Script

The master script is the `pbsrun_wrap` command, which takes two arguments: the `mpirun` to be wrapped, and a PBS-supplied wrapper. The `pbsrun_wrap` command neatly wraps the original `mpirun` so that everything is transparent for the job submitter. See [“pbsrun\\_wrap” on page 52 of the PBS Professional Reference Guide](#), and [“pbsrun” on page 41 of the PBS Professional Reference Guide](#).

The `pbsrun_wrap` command does the following:

- Renames the original, named `mpirun.<flavor>`, to `mpirun.<flavor>.actual`
- Instantiates the wrapper as `pbsrun.<flavor>`
- Creates a link named `mpirun.<flavor>` that calls `pbsrun.<flavor>`
- Creates a link so that `pbsrun.<flavor>` calls `mpirun.<flavor>.actual`

### 13.11.1 Passing Arguments

Any `mpirun` version/flavor that can be wrapped has an initialization script ending in `.init`, found in `$PBS_EXEC/lib/MPI`:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init
```

When executed inside a PBS job, the `pbsrun.<flavor>` script calls a version-specific initialization script which sets variables to control how the `pbsrun.<flavor>` script uses options passed to it. For example, `pbsrun.<flavor>` calls `$PBS_EXEC/lib/MPI/pbsrun.<flavor>.init` to manage the arguments passed to it. You can modify the `.init` scripts to specify which arguments should be retained, ignored, or transformed.

When the `mpirun` wrapper script is run inside a PBS job, then it translates any `mpirun` call of the form:

```
mpirun [options] <executable> [args]
```

into

```
mpirun [options] pbs_attach [special_option_to_pbs_attach] <executable> [args]
```

where *[special options]* refers to any option needed by `pbs_attach` to do its job (e.g. `-j $PBS_JOBID`).

See [“Options” on page 42 of the PBS Professional Reference Guide](#) for a description of how to customize the initialization scripts.

### 13.11.2 Restricting MPI Use to PBS Jobs

You can specify that a wrapped MPI can be used only inside of PBS, by using the `-s` option to the `pbsrun_wrap` command. This sets the `strict_pbs` option in the initialization script (e.g. `pbsrun.ch_gm.init`, etc...) to `1` from the default of `0`. This means that the `mpirun` being wrapped by `pbsrun` will be executed only when it is called inside a PBS environment. Otherwise, the user gets the following error:

```
Not running under PBS
exiting since strict_pbs is enabled; execute only in PBS
```

By default, when the wrapper script is executed outside of PBS, a warning is issued about "not running under PBS", but it proceeds as if the actual program had been called in standalone fashion.

### 13.11.3 Format of pbsrun\_wrap Command

The `pbsrun_wrap` command has this format:

```
pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<keyword>
```

Make sure that you wrap the correct instance of the `mpirun`. If a user's job script would call a link, wrap the link. See [section 13.10.1, “Wrap the Correct Instance”, on page 564](#).

### 13.11.4 Actions During Wrapping

The `pbsrun_wrap` script instantiates the `pbsrun` wrapper script as `pbsrun.<mpirun version/flavor>` in the same directory where `pbsrun` is located, and sets up the link to the actual `mpirun` call via the symbolic link:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```

For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

- Save original `mpirun.ch_gm` script:  
`mv /opt/mpich-gm/bin/mpirun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm.actual`
- Instantiate `pbsrun` wrapper script as `pbsrun.ch_gm`:  
`cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm`
- Link `mpirun.ch_gm` to actually call `pbsrun.ch_gm`:  
`ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm`
- Create a link so that `pbsrun.ch_gm` calls `mpirun.ch_gm.actual`:  
`ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link`

### 13.11.5 Requirements

The `mpirun` being wrapped must be installed and working on all the vnodes in the PBS cluster.

### 13.11.6 Caveats and Restrictions

- For MPIs that are wrapped using `pbsrun_wrap`, the maximum number of ranks that can be launched in a job is the number of entries in the `$PBS_NODEFILE`.
- MVAPICH2 must use the "mpd" process manager if it is to be integrated with PBS. During the configuration step when you build MVAPICH2, set the "process manager" setting to `mpd`, as follows:  
`--with-pm=mpd`  
Other process managers such as "hydra" and "gforker" may not work correctly with PBS.
- If you wrap a version of Intel MPI `mpirun` less than 4.0.3, Hydra is not supported.
- Wrapping Intel MPI is **deprecated**.

## 13.11.7 Links to Wrapper Script Information

The following table lists the links to the description of each wrapper script used by `pbsrun_wrap`:

**Table 13-2: Links to Wrapper Descriptions**

| MPI Wrapper                                                         | Link to Description                                                                                                             |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| MPICH2                                                              | See <a href="#">“MPICH2 mpirun: pbsrun.mpich2”</a> on page 47 of the PBS Professional Reference Guide.                          |
| MPICH-GM with MPD<br><b>Wrapper is deprecated.</b>                  | <a href="#">“MPICH-GM mpirun (mpirun.mpd) with MPD: pbsrun.gm_mpd”</a> on page 45 of the PBS Professional Reference Guide.      |
| MPICH-GM with <code>rsh/ssh</code><br><b>Wrapper is deprecated.</b> | <a href="#">“MPICH-GM mpirun (mpirun.ch_gm) with rsh/ssh: pbsrun.ch_gm”</a> on page 44 of the PBS Professional Reference Guide. |
| MPICH-MX with MPD<br><b>Wrapper is deprecated.</b>                  | <a href="#">“MPICH-MX mpirun (mpirun.mpd) with MPD: pbsrun.mx_mpd”</a> on page 46 of the PBS Professional Reference Guide.      |
| MPICH-MX with <code>rsh/ssh</code><br><b>Wrapper is deprecated.</b> | <a href="#">“MPICH-MX mpirun (mpirun.ch_mx) with rsh/ssh: pbsrun.ch_mx”</a> on page 45 of the PBS Professional Reference Guide. |
| MVAPICH<br><b>Wrapper is deprecated.</b>                            | <a href="#">“MVAPICH1 mpirun: pbsrun.mvapich1”</a> on page 49 of the PBS Professional Reference Guide.                          |
| MVAPICH2                                                            | <a href="#">“MVAPICH2 mpiexec: pbsrun.mvapich2”</a> on page 50 of the PBS Professional Reference Guide.                         |
| Intel MPI                                                           | <a href="#">“Intel MPI mpirun: pbsrun.intelmpi”</a> on page 48 of the PBS Professional Reference Guide.                         |

## 13.11.8 Wrapping Multiple MPIs with the Same Name

You may want more than one MPI environment with the same name, for example a 32-bit and a 64-bit version of MPICH2.

1. Create two new MPICH2 initialization scripts by copying that for MPICH2:

```
cd $PBS_EXEC/lib/MPI
cp pbsrun.mpich2.init.in pbsrun.mpich2_32.init.in
cp pbsrun.mpich2.init.in pbsrun.mpich2_64.init.in
```

2. Then wrap them:

```
pbsrun_wrap <path to 32-bit MPICH2>/bin/mpirun pbsrun.mpich2_32
pbsrun_wrap <path to 64-bit MPICH2>/bin/mpirun pbsrun.mpich2_64
```

Calls to `<path to 32-bit MPICH2>/bin/mpirun` will invoke `/usr/pbs/bin/pbsrun.mpich2_32`. The 64-bit version is invoked with calls to `<path to 64-bit MPICH2>/bin/mpirun`.

## 13.11.9 See Also

See [“pbsrun”](#) on page 41 of the PBS Professional Reference Guide for a description of the `pbsrun` script. See [“pbsrun\\_wrap”](#) on page 52 of the PBS Professional Reference Guide for a description of the master wrapping script.

## 13.12 Unwrapping MPIs Using the `pbsrun_unwrap` Script

You can also use the matching `pbsrun_unwrap` command to unwrap the MPIs you wrapped using `pbsrun_wrap`.

For example, you can unwrap the two MPICH2 MPIs from [13.11.8](#) above:

```
pbsrun_unwrap pbsrun.mpich2_32
pbsrun_unwrap pbsrun.mpich2_64
```

See “`pbsrun_unwrap`” on page 51 of the [PBS Professional Reference Guide](#).

## 13.13 Integration By Hand

For MPIs that must be wrapped by hand, PBS supplies wrapper scripts which call the original and use `pbs_attach` to give MoM control of jobs.

Wrapping an MPI by hand yields the same result as wrapping using `pbsrun_wrap`, but you must perform the steps by hand.

Wrapping by hand involves the following steps (which are the same steps taken by `pbsrun_wrap`):

- You rename the original MPI command
- You create a link whose name is the same as the original MPI command; this link calls the wrapper script
- You edit the wrapper script to call the original MPI command
- You make sure that the link to the wrapper script(s) is available to each user's `PATH`.

The following table lists MPIs, their wrapper scripts, and a link to instructions:

**Table 13-3: Scripts for Wrapping MPIs by Hand**

| MPI Name                        | Script Name             | Link to Instructions                                                                |
|---------------------------------|-------------------------|-------------------------------------------------------------------------------------|
| HP MPI                          | <code>pbs_mpihp</code>  | <a href="#">section 13.13.1, “Integrating HP MPI and Platform MPI”, on page 568</a> |
| MPICH<br>Wrapper is deprecated. | <code>pbs_mpirun</code> | <a href="#">section 13.13.4, “Integrating MPICH-P4”, on page 570</a>                |
| Platform MPI                    | <code>pbs_mpihp</code>  | <a href="#">section 13.13.1, “Integrating HP MPI and Platform MPI”, on page 568</a> |
| HPE MPI                         | <code>mpiexec</code>    | <a href="#">section 13.13.5, “Integrating HPE MPI”, on page 571</a>                 |

### 13.13.1 Integrating HP MPI and Platform MPI

PBS supplies a wrapper script for HP MPI and Platform MPI called `pbs_mpihp`. The `pbs_mpihp` script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.

You can make `pbs_mpihp` transparent to users; see the instructions that follow.

### 13.13.2 Steps to Integrate HP MPI or Platform MPI

Make sure that you wrap the correct instance of the MPI. If a user's job script would call a link, wrap the link. See [section 13.10.1, “Wrap the Correct Instance”, on page 564](#).

The `pbs_mpirun` command looks for a link with the name `PBS_EXEC/etc/pbs_mpihp` that points to the HP `mpirun`. The `pbs_mpihp` command follows this link to HP's `mpirun`. Therefore, the wrapping instructions are different from the usual. See [“pbs\\_mpihp” on page 76 of the PBS Professional Reference Guide](#) for more information on `pbs_mpihp`.

1. Rename HP's `mpirun`:  

```
cd <MPI installation location>/bin
mv mpirun mpirun.hp
```
2. Link the user-callable `mpirun` to `pbs_mpihp`:  

```
cd <MPI installation location>/bin
ln -s $PBS_EXEC/bin/pbs_mpihp mpirun
```
3. Create a link to `mpirun.hp` from `PBS_EXEC/etc/pbs_mpihp`. `pbs_mpihp` will call the real HP `mpirun`:  

```
cd $PBS_EXEC/etc
ln -s <MPI installation location>/bin/mpirun.hp pbs_mpihp
```

### 13.13.2.1 Setting Up `rsh` and `ssh` Commands

When wrapping HP MPI with `pbs_mpihp`, note that `rsh` is the default used to start the `mpids`. If you wish to use `ssh` or something else, be sure to set the following or its equivalent in `$PBS_HOME/pbs_environment`:

```
PBS_RSHCOMMAND=ssh
```

### 13.13.2.2 Restrictions and Caveats for HP MPI and Platform MPI

- The `pbs_mpihp` script can be used only on HP-UX and Linux.
- The HP `mpirun` or `mpiexec` must be in the job submitter's `PATH`.
- The version of the HP `mpirun` or `mpiexec` must be HPMPI or Platform.
- Under the wrapped HP MPI, the job's working directory is changed to the user's home directory.

## 13.13.3 Integrating Open MPI

All Open MPI versions allow you to compile the MPI with the PBS TM interface. We recommend compiling all Open MPI versions with the TM module.

All versions of Open MPI can be transparent to the job submitter.

### 13.13.3.1 Compiling Open MPI with the TM Module

If the TM interface library is in the standard location, `PBS_EXEC/lib/`, Open MPI will find it and use it. You need to explicitly configure with TM only if it's in a non-standard location.

To integrate Open MPI with PBS, configure Open MPI with the `--with-tm` command-line option to the `configure` script. For example:

```
./configure --prefix=/opt/openmpi/1.4.4 --with-tm=${PBS_EXEC}
make
make install
```

After you compile Open MPI on one host, make it available on every execution host that will use it, by means of shared file systems or local copies.

For the Open MPI website information on compiling with the TM option, see:

---

<http://www.open-mpi.org/faq/?category=building#build-rte-tm>

### 13.13.3.2 Verifying Use of TM Interface

To see whether your Open MPI installation has been configured to use the TM interface:

```
% mpi_info | grep tm
MCA ras: tm (MCA v2.0, API v2.0, Component v1.3)
MCA plm: tm (MCA v2.0, API v2.0, Component v1.3)
```

### 13.13.3.3 See Also

See <http://www.open-mpi.org/faq/?category=building#build-rte-tm> for information about building Open MPI with the TM option.

## 13.13.4 Integrating MPICH-P4

**Wrapper is deprecated.** PBS supplies a wrapper script called `pbs_mpirun` for integrating MPICH-P4 with PBS by hand. The `pbs_mpirun` script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.

You can make `pbs_mpirun` transparent to job submitters. See the following steps.

### 13.13.4.1 Restrictions

- The `pbs_mpirun` command can be used only with MPICH using P4 on Linux.
- Usernames must be identical across hosts.

### 13.13.4.2 Options for `pbs_mpirun`

The usage for `pbs_mpirun` is the same as `mpirun` except for the listed options. All other options are passed directly to `mpirun`:

#### `-machinefile`

The value for this option is generated by `pbs_mpirun`. The value used for the `-machinefile` option is a temporary file created from the `PBS_NODEFILE` in the format expected by `mpirun`.

If the `-machinefile` option is specified on the command line, a warning is output saying "Warning, `-machinefile` value replaced by PBS".

#### `-np`

The default value for the `-np` option is the number of entries in `PBS_NODEFILE`.

### 13.13.4.3 Steps to Integrate MPICH-P4

To make `pbs_mpirun` transparent to the user, replace standard `mpirun` with `pbs_mpirun`. Make sure that you wrap the correct instance of the MPI. If a user's job script would call a link, wrap the link. See [section 13.10.1, “Wrap the Correct Instance”](#), on page 564.

- Install MPICH-P4 into `<path to mpirun>`
- Rename `mpirun` to `mpirun.std`:  
`mv <path to mpirun>/mpirun <path to mpirun>/mpirun.std`
- Create link called `mpirun` in `<path to mpirun>` that points to `pbs_mpirun`  
`ln -s <path to pbs_mpirun>/pbs_mpirun mpirun`
- Edit `pbs_mpirun` to change the call to `mpirun` so that it calls `mpirun.std`

At this point, using `mpirun` actually invokes `pbs_mpirun`.

### 13.13.4.4 Setting Up Environment Variables and Paths

- For `pbs_mpirun` to function correctly for users who require the use of `ssh` instead of `rsh`, you can do one of the following:
  - Set `PBS_RSHCOMMAND` in the login environment
  - Set `P4_RSHCOMMAND` externally to the login environment, then have job submitters pass the value to PBS via `qsub ( 1 )'s -v or -V` arguments:  
`qsub -vP4_RSHCOMMAND=ssh ...`  
 or  
`qsub -V ...`
  - Set `P4_RSHCOMMAND` in the `pbs_environment` file in `PBS_HOME` and then advise users to not set `P4_RSHCOMMAND` in the login environment
- Make sure that `PATH` on remote machines contains `PBS_EXEC/bin`. Remote machines must all have `pbs_attach` in the `PATH`.
- The `PBS_RSHCOMMAND` environment variable should not be set by the user.
- When using SuSE Linux, use `"ssh -n"` in place of `"ssh"`.

## 13.13.5 Integrating HPE MPI

PBS supplies its own `mpiexec` on machines running supported versions of HPE MPI, in order to provide a standard interface for use by job submitters. This `mpiexec` calls the standard HPE `mpirun`. If users call this `mpiexec`, PBS will manage, track, and cleanly terminate multi-host MPI jobs.

If job submitters call HPE MPI directly, they must use `pbs_attach` in their job scripts in order to give PBS the same control over jobs; see the HPE documentation.

MPI jobs can be launched across multiple machines. PBS users can run an MPI job within a specific partition.

When job submitters use `mpiexec` in their job scripts, HPE MPI is transparent. Jobs run normally whether the PBS-supplied `mpiexec` is called inside or outside of PBS.

### 13.13.5.1 Supported Platforms

The PBS-supplied `mpiexec` runs on machines running supported versions of HPE MPI.



### 13.13.5.2 Steps to Integrate HPE MPI

Make sure that the PBS-supplied `mpiexec` is in each user's `PATH`.

### 13.13.5.3 Invoking HPE MPI

PBS uses the MPI-2 industry standard `mpiexec` interface to launch MPI jobs within PBS. If executed on a non-HPE system, PBS's `mpiexec` will assume it was invoked by mistake. In this case it will use the value of `PATH` (outside of PBS) or `PBS_O_PATH` (inside PBS) to search for the correct `mpiexec` and if one is found, exec it.

### 13.13.5.4 Using HPE MPI Over InfiniBand

To use InfiniBand, set the `MPI_USE_IB` environment variable to 1.

### 13.13.5.5 Using CSA with HPE MPI

PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

### 13.13.5.6 Prerequisites

- In order to run single-host or multi-host jobs, the HPE Array Services must be correctly configured. An Array Services daemon (`arrayd`) must run on each host that will run MPI processes. For a single-host environment, `arrayd` only needs to be installed and activated. However, for a multi-host environment where applications will run across hosts, the hosts must be properly configured to be an array.
- HPE systems communicating via HPE's Array Services must all use the same version of the `sgi-mpt` and `sgi-arraysvcs` packages. HPE systems communicating via HPE's Array Services must have been configured to interoperate with each other using the default array. See HPE's `array_services(5)` man page.
- "`rpm -qi sgi-arraysvcs`" should report the same value for *Version* on all systems.
- "`rpm -qi sgi-mpt`" should report the same value for *Version* on all systems.
- "`chkconfig array`" must return "on" for all systems
- `/usr/lib/array/arrayd.conf` must contain an array definition that includes all systems.
- `/usr/lib/array/arrayd.auth` must be configured to allow remote access:  
The "AUTHENTICATION NOREMOTE" directive must be commented out or removed  
Either "AUTHENTICATION NONE" should be enabled or keys should be added to enable the SIMPLE authentication method.
- If any changes have been made to the `arrayd` configuration files (`arrayd.auth` or `arrayd.conf`), the array service must be restarted.
- `rsh(1)` must work between the systems.
- PBS uses HPE's `mpirun(1)` command to launch MPI jobs. HPE's `mpirun` must be in the standard location.
- The location of `pbs_attach(8B)` on each vnode of a multi-vnode MPI job must be the same as it is on the primary execution host vnode.

### 13.13.5.7 Environment Variables

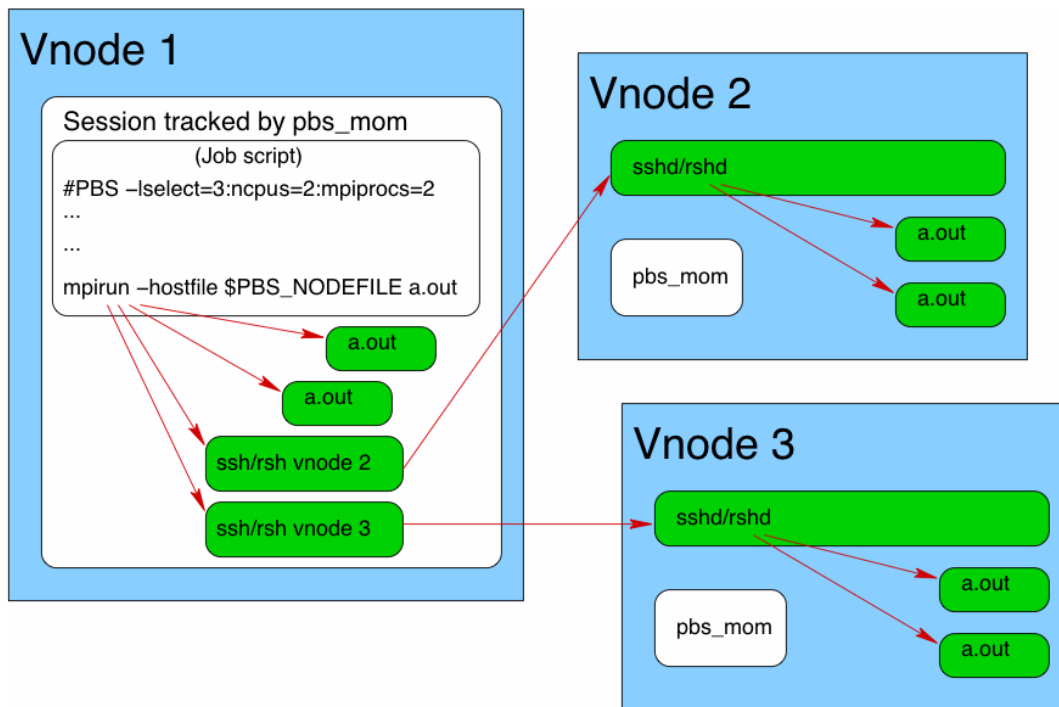
- If the `PBS_MPI_DEBUG` environment variable's value has a nonzero length, PBS will write debugging information to standard output.
- The `PBS_ENVIRONMENT` environment variable is used to determine whether `mpiexec` is being called from within a PBS job.
- If it was invoked by mistake, the PBS `mpiexec` uses the value of `PBS_O_PATH` to search for the correct `mpiexec`.
- To use InfiniBand, set the `MPI_USE_IB` environment variable to 1.

## 13.14 How Processes are Started Using MPI and PBS

### 13.14.1 Starting Processes under Non-integrated MPIs

The following figure illustrates how processes are started on sister vnodes when using a non-integrated MPI:

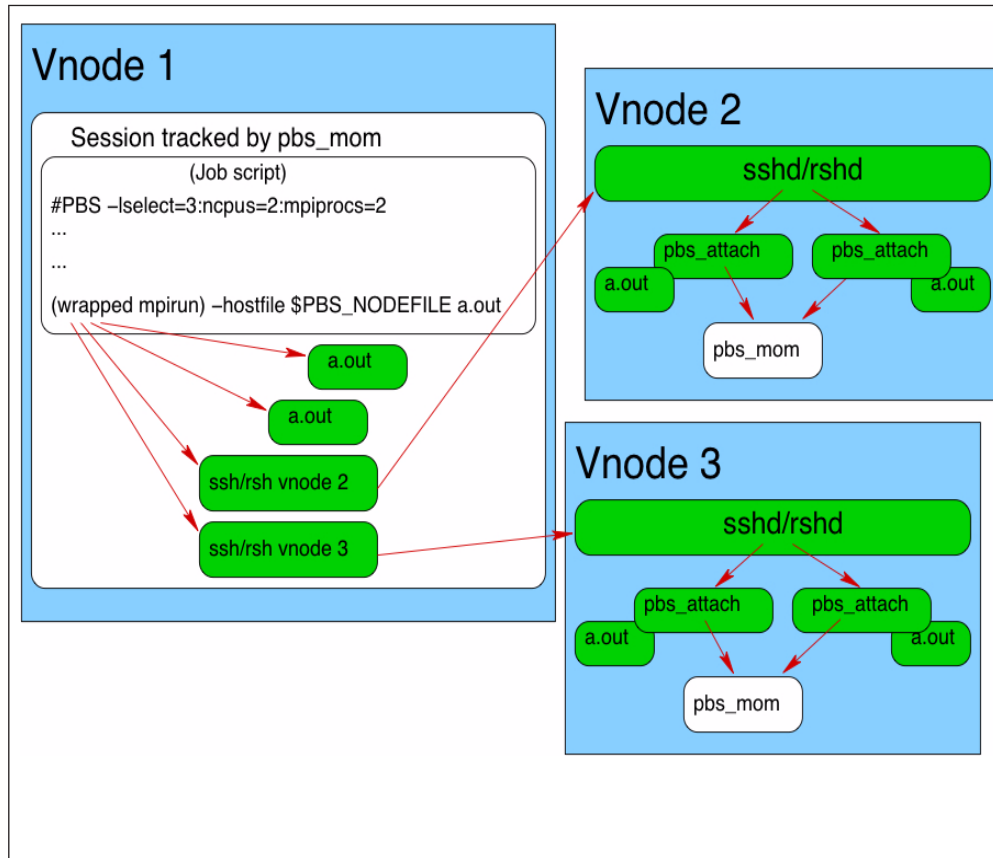
Figure 13-4: PBS does not know about the processes on vnodes 2 and 3, because those processes were generated outside of the scope of PBS.



### 13.14.2 Starting Processes under Wrapped MPIs

The following figure illustrates how processes are started on sister vnodes when using a wrapped MPI:

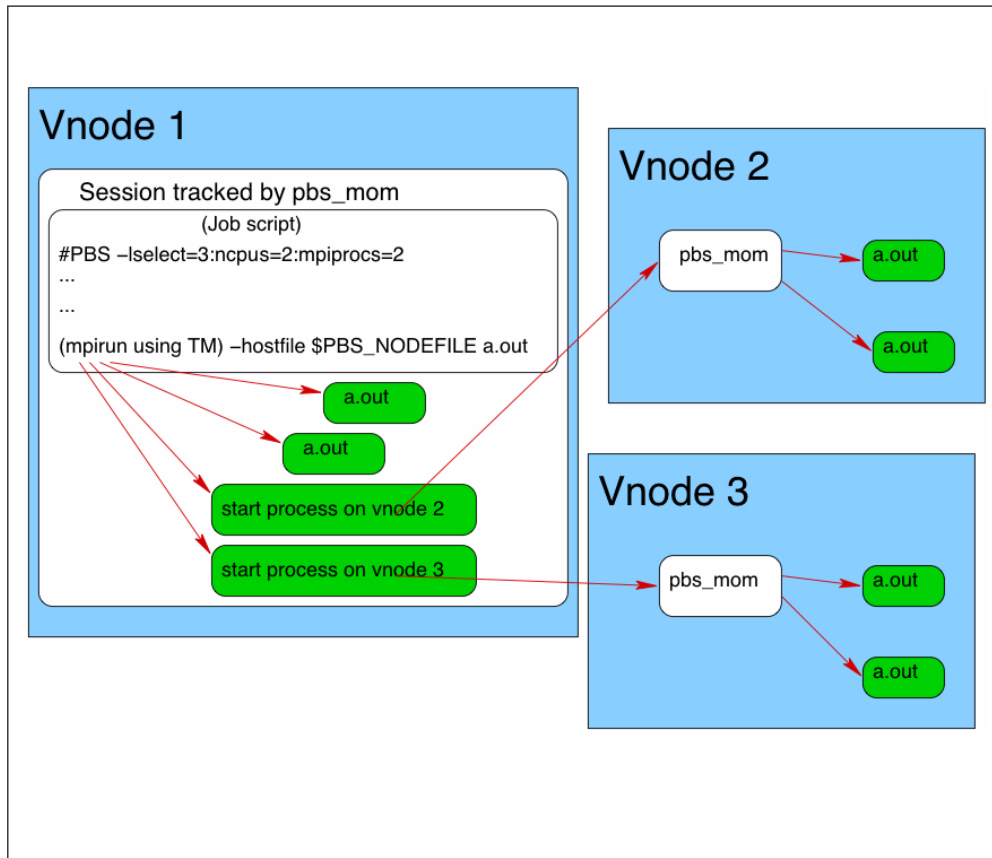
Figure 13-5: PBS knows about processes on vnodes 2 and 3, because `pbs_attach` tells those MoMs which processes belong to which jobs



### 13.14.3 Starting Processes Under MPIs Employing the TM Interface

The following figure illustrates how processes are started on sister vnodes when using an MPI that employs the TM interface:

Figure 13-6: PBS knows about processes on vnodes 2 and 3, because the TM interface talks directly to pbs\_mom, and pbs\_mom starts the processes on vnodes 2 and 3



## 13.15 Limit Enforcement with MPI

PBS can enforce the following for a job using MPI:

- Per-process limits via `setrlimit(2)` on sister vnodes
  - The `setrlimit` process limit can be enforced only when using an MPI that employs the TM interface directly, which is Open MPI only
- Limits set via MoM parameters, e.g. `cpuburst` and `cpuaverage`, on sister vnodes
  - PBS can enforce these limits using any integrated MPI
- Job-wide limits such as `cput`, `mem`
  - PBS can enforce job-wide limits using any integrated MPI

Once a process is started, process limits cannot be changed.

---

## 13.16 Restrictions and Caveats for MPI Integration

- Be sure to wrap the correct instance of the MPI. See [section 13.10.1, “Wrap the Correct Instance”, on page 564](#)
- Some applications write scratch files to a temporary location in `tmpdir`. The location of `tmpdir` is host-dependent. If you are using an MPI that is not integrated with the PBS TM interface, and your application needs scratch space, the location of `tmpdir` for the job should be consistent across execution hosts. You can specify the root for `tmpdir` in the MoM's `$tmpdir` configuration parameter. PBS sets the job's `TMPDIR` environment variable to the temporary directory it creates for the job.

# Configuring PBS for SELinux

## 14.1 Overview of PBS Support for MLS-compliant SELinux

With this version of PBS Professional, we offer a separate package named `PBSPro_2021.1.3-RHEL7_x86_64_selinux.tar.gz` that supports SELinux enforcement mode used with one or more MLS policies on RHEL 7. In this chapter, we describe how to use SELinux PBS only.

## 14.2 Terminology

In this section, we use the following terms:

**DCID**

Director of Central Intelligence

**MLS**

Multi-level Security

**PL4**

Protection Level 4

## 14.3 How Support for SELinux Works

### 14.3.1 Security Context

SELinux PBS collects the security context at the time a request is made, records it in the server, and associates it with a user's job when the job is passed to the MoM. MoMs then use the information to create user processes with the correct context.

### 14.3.2 Authorization

Job submitters can alter, delete, and hold only those jobs with a security context which matches that of the requester. PBS managers and operators are expected to have sufficient SELinux privilege to operate on all users' jobs.

### 14.3.3 Authentication

When authenticating a user, this version of PBS captures the user's SELinux security context, collecting the SELinux user identity, role, type, and MLS levels.

The only supported authentication method when using PBS with SELinux is the default of *resvport*.

### 14.3.4 Instantiation

When a MoM creates a user process, she creates it with the security context of the user who submitted the job. For each job, PBS records the security context of the job submitter in the `security_context` job attribute.

## 14.4 Enforcement of Permissions

PBS does not make the permission decisions required by MLS or DCID PL4. These decisions are made by the SELinux mechanisms delivered with the OS.

### 14.4.1 Policy Files

Altair supplies policy files describing the permissions for PBS. These files are in SELinux format.

The SELinux policy framework includes the permissions PBS needs to impersonate users. The policy framework is encapsulated in a set of policy files for this product.

SELinux policy files for PBS are located in `$PBS_EXEC/selinux/`. These policy files are the following:

`pbs.fc`

`pbs.if`

`pbs.te`

`pbs_dontaudit.te`

PBS policy files must be inspected and vetted by site security personnel before installation. See [section 14.8.3, “Installation Steps”, on page 580](#).

### 14.4.2 Location of `pbs_mom.pamd` File

The `pbs_mom.pamd` file is shipped in `$PBS_EXEC/selinux`.

For polyinstantiation to work properly, copy `$PBS_EXEC/selinux/pbs_mom.pamd` to the `/etc/pam.d/` directory.

## 14.5 Special Attributes and Directories

`$PBS_EXEC/selinux`

Directory that contains SELinux policy files and miscellaneous data.

`security_context`

Job attribute. Contains security context of job submitter. Set by PBS to the security context of the job submitter at the time of job submission. Visible to all. If not present when a request is submitted, an error occurs, a server message is logged, and the request is rejected.

Format: String in SELinux format

Default: Unset



## 14.6 Prerequisites

Because PBS managers and operators may need access to information requiring elevated privilege such as job names or security contexts, PBS requires that PBS managers and operators have sufficient SELinux privilege to operate on all users' jobs.

You can enable SELinux only on RHEL 7 hosts.

## 14.7 Caveats and Restrictions

- You cannot upgrade from a non-SELinux compliant PBS version to this version; you must do a fresh install.
- This version of PBS cannot interoperate with non-SELinux compliant clients (`qsub`, `qstat`, etc.).
- This version of PBS cannot interoperate with non-SELinux compliant complexes; you cannot do peer scheduling, route jobs, etc.
- You cannot mix compliant and non-compliant daemons
- In order to use SELinux with containers, you will probably need to adjust your site policy.
- The X forwarding that is enabled by PBS does not work with this version of PBS. You may be able to set up X forwarding via normal methods, without the help of PBS. Omit the "-X" `qsub` argument, and forward the `DISPLAY` information into the job itself.
- If a user runs `qstat` on a non-server host, all jobs are reported regardless of the value of `query_other_jobs`. To protect against this, set the server's `acl_hosts` attribute to exclude all non-server hosts:

```
set server acl_hosts="-<non-server host>,-<non-server host>"
```

For example:

```
set server acl_hosts="-submithost1,-submithost2,-submithost3"
```

Set the server's `acl_host_enable` attribute to *True* to enable restricting host access:

```
set server acl_host_enable=true
```

- An SELinux-enabled PBS complex cannot interoperate with non-SELinux-enabled PBS complexes.

## 14.8 Installing PBS For Use With SELinux

### 14.8.1 Pre-installation Requirements

- Make sure that the GraphQL interface is disabled: set the `webapi_enable` server attribute to *False*.
- An SELinux run-time environment must be present.
- The SELinux run-time environment must be able to compile and install new policies.
- The site must configure polyinstantiation as needed.
- You must install, start, and stop PBS using an account that is not subject to polyinstantiation.
- If PBS will run on more than one host, you must ensure that the MLS label is passed through and trusted among the PBS peer services. Add the following line to each PBS node's `/etc/netlabels.rules`:  

```
map add default address:<site-specific address/mask> protocol:cipsov4,32
```
- Make sure that the data service management account is configured; we recommend that it is called *pbsdata*. See [“Create PBS Data Service Management Account” on page 24 in the PBS Professional Installation & Upgrade Guide](#).

## 14.8.2 Installing in Non-default Location

The policy files, for example `pbs.fc`, contain full paths that assume default locations for `PBS_HOME`, `PBS_EXEC`, and `PBS_CONF_FILE`. If you will install PBS in a non-default location, make sure that you set these correctly. We recommend that you review these anyway.

## 14.8.3 Installation Steps

1. Make sure you are using an account that is not subject to polyinstantiation.
2. Make sure you have sufficient privilege to install a policy.
3. Extract the policy files:

```
rpm2cpio pbs-<version>.el7.x86_64.rpm | cpio -id './opt/pbs/selinux/pbs*.[fit] [cfe]'
```

4. Site security personnel inspect the policy files and PBS package.
5. Install the PBS policy module. We provide the following instruction for convenience:

```
cd opt/pbs/selinux
make -f /usr/share/selinux/devel/Makefile && /usr/sbin/semodule -i pbs.pp pbs_dontaudit.pp
```

6. Install the PBS package, but do not start PBS:

```
rpm -i pbspro-server-<version>.el7.x86_64.rpm
```

7. Put the PBS PAM module in place. (Polyinstantiation is controlled by the PAM session module. You need a PBS PAM module in order to allow `pbs_mom` to polyinstantiate user jobs.) Copy `pbs_mom.pamd` from `$PBS_EXEC/selinux/` on the PBS server installation host to `/etc/pam.d/pbs_mom` on all execution hosts.
8. Make sure that `/etc/pam.d/pbs_mom` is readable by all.

## 14.8.4 Starting SELinux PBS

Use the following instructions to start PBS. Do **not** use the standard instructions for starting PBS. Use the exact instructions below:

```
systemctl start pbs.service
```

## 14.8.5 Post-installation Steps

### 14.8.5.1 Configure Job Privacy

Once PBS has been started, set the `query_other_jobs` server attribute to *False*.

### 14.8.5.2 Set Privacy for PBS Logs

Protect the PBS service logs from being read or written by anyone except root. The logs are found here:

```
$PBS_HOME/mom_logs
```

```
$PBS_HOME/sched_logs
```

```
$PBS_HOME/server_logs
```

```
$PBS_HOME/comm_logs
```

Change permissions on the logs:

```
chmod 700 $PBS_HOME/mom_logs $PBS_HOME/sched_logs $PBS_HOME/server_logs $PBS_HOME/comm_logs
```

## 14.8.6 Configuring Ports Used by PBS

If client commands such as `qstat`, `qsub`, etc., frequently fail with error code 15007, the likely problem is that the port in the 512-1023 range that PBS is trying to use to communicate has other SELinux policy rules set up which prohibit use by PBS.

For example, in this illustration PBS tried to use port 548, which is listed like this in `semanage`:

```
dhcpd_port_t tcp 547, 548, 647, 847, 7911
dhcpd_port_t udp 67, 547, 548, 647, 847
```

In our illustration, the audit log shows something like this:

```
type=AVC msg=audit(1437660325.774:5672072): avc: denied { name_bind } for pid=40996
comm="pbs_iff" src=548 scontext=staff_u:sysadm_r:sysadm_t:s1
tcontext=system_u:object_r:dhcpd_port_t:s0 tclass=tcp_socket
```

When this happens, you need to let PBS know which ports in the 512 to 1023 range may be used by PBS. Specify at least 128 ports for use by PBS. Use the `semanage` command to appropriately label the ports that are usable by PBS:

```
semanage port -a -t pbsd_iff_port_t -p tcp <port range>
```

See `semanage-port(8)`.

## 14.9 Configuring PBS for SELinux

### 14.9.1 Configure File Staging Utilities

PBS needs to use file staging utilities that can preserve security labels when staging files in and out. Files to be used for a job must have the user's context. Copying files without preserving the context results in a file that cannot be used by the job. When copying a local file, MoM automatically preserves the context. You must configure MoM to preserve context when copying remote files. You can use the following option for authenticated file transfer between Linux systems. The `-xattrs` option preserves context:

```
rsync -e ssh -xattrs
```

The following is another option for preserving context:

```
cp --preserve=context
```

MoM performs remote staging using the file transfer methods that you specify, or the default if none is specified. You specify the method you want by putting the path in the `PBS_SCP` and `PBS_RCP` variables in `pbs.conf`. First MoM tries the path in `PBS_SCP`, then the path in `PBS_RCP`, so put the same path in both. The following is a summary of how to tell MoM to use `rsync`:

- Write a script that passes the desired options and arguments to `rsync`
- Edit `pbs.conf`, and specify the path to the script in `PBS_RCP` and `PBS_SCP`
- If the MoM is running, HUP the MoM

### 14.9.1.1 Steps to Configure Utility

1. Write a wrapper script named `rsync_pbs` that passes all arguments except for the first (`-Brvp` or `-rp`) to `rsync`. MoM uses the `-Brvp` flags when calling `PBS_SCP`, and the `-rp` flags when calling `PBS_RCP`. The arguments that were being passed, and that you can borrow, are the following:

\$1 `-Brvp` or `-rp`

\$2 path to source

\$3 path to destination

For example, our script passes all but the first argument to `rsync` as `$*`. We get rid of the first argument using the `shift` command.

In `pbs.conf`:

```
PBS_SCP=/usr/bin/rsync_pbs
```

In `/usr/bin/rsync_pbs`:

```
#!/bin/sh
```

```
shift
```

```
/usr/bin/rsync -e ssh -xattrs $*
```

2. Configure both `PBS_RCP` and `PBS_SCP` with the path to `rsync_pbs`.
3. If the MoM is already running, HUP the MoM.

## 14.10 Managing an SELinux System

### 14.10.1 Checking Security Context

PBS writes the security context for a job in its `security_context` attribute. For example, if you need to compare the security context of jobs and users:

Find job security context:

```
bash-4.2$ qstat -f | grep security
security_context = user_u:user_r:user_t:s3:c1,c2
```

Find user security context:

```
bash-4.2$ id -Z
user_u:user_r:user_t:s3:c1,c2
```

# Managing Power Usage

## 15.1 Monitoring and Controlling Job Power Usage

### 15.1.1 Power Provisioning: Monitoring and Controlling Job Power Usage

PBS Professional can control and monitor job power usage. PBS can assign a power profile for each job at submission time to control the job's power draw, and jobs can request power profiles.

PBS collects energy consumption information and records it in each job's `resources_used.energy` value. PBS provides information about job energy usage in the output of the `qstat` command, and records energy usage in the accounting logs.

For each job or power profile change, PBS provisions each vnode with the required power profile. The `ee` resource represents one or more power profiles on each vnode. Each vnode's `current_ee` attribute shows its current power profile.

Jobs can request a power profile, by requesting a value for the `ee` resource, or PBS can set each job's `ee` request. When a job requests a power profile, it is sent to vnodes that have this profile available, and when the job runs, the vnodes where the job runs are set to the power profile requested by the job.

The default power setting for a node is no power capping. If a job runs on a node, the node uses the requested power profile, but when the job finishes, the node goes back to the default setting.

#### 15.1.1.1 Monitoring Power Use by Jobs

To see the power used by a job, you can use `qstat` to examine the job's `resources_used.energy`.

### 15.1.2 Platforms Supporting Power Provisioning

Power provisioning is supported on HPE 8600 machines with HPE Performance Cluster Manager (HPCM).

### 15.1.3 Power Provisioning on HPE

#### 15.1.3.1 Overview of Power Provisioning on HPE

On HPE, PBS uses the power API for HPE Performance Cluster Manager (HPCM). PBS handles querying for available power profiles and setting the `ee` resource to the available power profiles. You enable power provisioning at the server and vnodes.

#### 15.1.3.2 Setting Power Profiles on HPE

On each vnode, PBS queries the HPE Performance Cluster Manager (HPCM) for available power profiles, and sets the vnode's `resources_available.ee` to the power profiles available on that vnode.

### 15.1.3.3 Enabling Power Provisioning on HPE

To enable power provisioning:

1. On each vnode where you want power provisioning enabled, set the `power_provisioning` vnode attribute to *True*:  
`Qmgr: set node <node name> power_provisioning=true`
2. Enable the `PBS_power` hook:  
`Qmgr: set pbshook PBS_power enabled=true`
3. HUP each MoM. If the vnode does not report `resources_available.eoe`, HUP the MoM again.

### 15.1.3.4 Setting Job Power Resource Requests

Each job can request or be assigned one of the power profiles you have defined.

You write a `queuejob` hook that sets each job's power attributes to the correct values for the requested profile. The hook maps each profile to a value for each of the power attributes. For example, if a compute node supports the following settings:

```
pstate range = 100-300
pgov range = 50-150
pcap_node range = 250-500
pcap_accelerator range = 500-1000
```

and you want to have profiles named "low", "med", and "high", the following table shows sample power profile settings:

**Table 15-1: Sample Power Profile Settings**

| Power Profile | pstate | pgov | pcap_node | pcap_accelerator |
|---------------|--------|------|-----------|------------------|
| low           | 100    | 50   | 250       | 500              |
| med           | 200    | 100  | 350       | 700              |
| high          | 300    | 150  | 500       | 1000             |

### 15.1.3.4.i Writing Power Profile Hook for Cray

Create a `queuejob` hook that sets each job's attributes to reflect its requested profile. Set each job's desired power profile by setting any of the following job attributes in the hook:

- Set the `pcap_node` job attribute to the value corresponding to the Cray `capmc set_power_cap --node` setting
- Set the `pstate` job attribute to the corresponding Cray ALPS p-state setting. Note that `pcap_node` takes precedence, and some settings for `pcap_node` can result in `pstate` being ignored.
- Set the `pcap_accelerator` job attribute to the value corresponding to the Cray `capmc set_power_cap --accel` setting
- Set the `pgov` job attribute for CPU throttling to the corresponding setting for p-governor

Example 15-1: We will use three profiles called "low", "med", and "high", and we will set `pcap_node` and `pstate` for each job requesting a profile:

```
import pbs

e = pbs.event()
j = e.job

profile = j.Resource_List['eoe']
if profile is None:
 res = j.Resource_List['select']
 if res is not None:
 for s in str(res).split('+')[0].split(':'):
 if s[:4] == 'eoe=':
 profile = s.partition('=')[2]
 break

pbs.logmsg(pbs.LOG_DEBUG, "got profile '%s'" % str(profile))
if profile == "low":
 j.Resource_List["pstate"] = "1900000"
 j.Resource_List["pcap_node"] = 100
 pbs.logmsg(pbs.LOG_DEBUG, "set low")
elif profile == "med":
 j.Resource_List["pstate"] = "220000"
 j.Resource_List["pcap_node"] = 200
 pbs.logmsg(pbs.LOG_DEBUG, "set med")
elif profile == "high":
 j.Resource_List["pstate"] = "240000"
 pbs.logmsg(pbs.LOG_DEBUG, "set high")
else:
 pbs.logmsg(pbs.LOG_DEBUG, "unhandled profile '%s'" % str(profile))

e.accept()
```



### 15.1.3.5 Enabling Power Provisioning on Cray

Enable power provisioning:

1. On each vnode where you want power provisioning enabled, set the `power_provisioning` vnode attribute to *True*:  
`Qmgr: set node <node name> power_provisioning=true`
2. Enable the `PBS_power` hook:  
`Qmgr: set pbshook PBS_power enabled=true`

### 15.1.3.6 Caveats for Power Provisioning on Cray

Note that `pcap_node` takes precedence, and some settings for `pcap_node` can result in `pstate` being ignored.

## 15.1.4 Terminology for Power Provisioning

### Activate a power profile

To set a power profile on a node, for example, to set a node's power profile to match the specifications for "low".

### Deactivate a power profile

To reset the power profile of the node to its default setting, which is no power capping.

## 15.1.5 Caveats and Restrictions for Using Power Profiles

- Make sure that your power provisioning `queuejob` hook takes into account your desired order of precedence for an explicit request for `pcap_node` and/or `pstate` versus a request for a profile. You may want users to be able to override power profile settings for `pcap_node` and/or `pstate`, or you may want profiles to override explicit requests.
- You cannot use power profiles on any hosts where the PBS server and/or scheduler are running.
- You cannot suspend jobs on vnodes that are using power profiles, meaning that you cannot use preemption via suspension on vnodes that are using power profiles.
- If you disable the `PBS_power` hook while a job is running, the vnodes where the job runs do not have their profile deactivated when the job finishes, and the job's `resources_used.energy` value is not set at the end of the job.
- A prologue script will not run when the `PBS_power` hook is enabled. Any prologue script must be converted to an `execjob_prologue` hook.
- If a job does not request a value for `aoe`, there is no activation of a power profile on a node, but the job's `resources_used.energy` is still calculated.
- If a job requests values for both `aoe` and `aoe`, PBS addresses the `aoe` request first.
- There is no PBS interface to `RUR` that can be used by administrators or job submitters.
- If `pbs.conf` is not in `/etc` on a host, add `PBS_CONF_FILE` to the `PBS_HOME/pbs_environment` file for that host, and set it to the path to `pbs.conf` on the host. For example, if `/var/pbs.conf` is the location of the `pbs.conf` file, add the following line to `PBS_HOME/pbs_environment`:  
`PBS_CONF_FILE=/var/pbs.conf`
- PBS does not automatically set `resources_available.aoe` on machines that host the PBS server/scheduler.
- PBS can provide precise power consumption accounting only where jobs are allocated exclusively. Vnodes must have the `sharing` attribute set so that jobs get exclusive use of the vnode, or jobs must request exclusive use of vnodes.

---

## 15.2 Power Management Attributes, Resources, Etc.

### ***energy***

Resource. Consumable. PBS records the job's energy usage in the job's `resources_used.energy`.

Format: *float*

Units: *kWh*

### ***eeo***

Resource. Stands for "Energy Operational Environment". Non-consumable. When set on a vnode in `resources_available.eeo`, contains the list of available power profiles. When set for a job in `Resource_List.eeo`, can contain at most one power profile. (A job can request only one power profile.) Automatically added to `resources:` line in `sched_config`.

Default value for `resources_available.eeo`: *unset*

Format: *string\_array*

### ***current\_eeo***

Vnode attribute. Shows the current value of `eeo` on the vnode.

Visible to all. Settable by manager. We do not recommend setting this attribute manually.

Format: *string*

Default: *unset*

### ***last\_state\_change\_time***

Vnode attribute. Records the most recent time that this node changed state.

Set by PBS. Readable by Manager and Operator.

Format: integer seconds since epoch

Default: no default

### ***last\_used\_time***

Vnode attribute. Records the most recent time that this node finished being used for a job or reservation.

Set at creation or reboot time. Updated when node is released early from a running job. Reset when node is ramped up.

Set by PBS. Readable by Manager and Operator.

Format: integer seconds since epoch

Default: no default

### ***max\_concurrent\_nodes***

Hook configuration parameter. Specifies the maximum number of nodes that can be powered up or down at one time. Enabled when the `PBS_power` hook is enabled. Used by the `PBS_power` hook.

set by Manager and Operator. Readable by all.

Format: positive integer

Default: 5

***min\_node\_down\_delay***

Hook configuration parameter. Specifies the minimum time a node is powered down before it can be powered back up. Enabled when the the **PBS\_power** hook is enabled.

set by Manager and Operator. Readable by all.

Format: integer seconds

Default: *1800 seconds*

***node\_idle\_limit***

Hook configuration parameter. Specifies the minimum idle time for a node to be considered for powering down. Enabled when the the **PBS\_power** hook is enabled.

set by Manager and Operator. Readable by all.

Format: integer seconds

Default: *1800 seconds*

***pstate***

Job attribute. Cray ALPS reservation setting for CPU frequency corresponding to p-state. See BASIL 1.4 documentation.

Settable by and visible to all PBS users.

Units: *hertz*

Format: *string*

Default: *unset*

Example: *pstate = 2200000*

***pgov***

Job attribute. Cray ALPS reservation setting for CPU throttling corresponding to p-governor. See BASIL 1.4 documentation. We do not recommend using this attribute.

Visible to all. Settable by all.

Format: *string*

Default: *unset*

***pcap\_node***

Job attribute. Power cap for a node. Corresponds to Cray `capmc set_power_cap --node` setting. See `capmc` documentation.

Visible to and settable by all.

Units: *watts*

Format: *int*

Default: *unset*

***pcap\_accelerator***

Job attribute. Power cap for an accelerator. Corresponds to Cray `capmc set_power_cap --accel` setting. See `capmc` documentation.

Visible to and settable by all.

Units: *watts*

Format: *int*

Default: *unset*

---

***poweroff\_eligible***

Vnode attribute. Specifies whether this node is eligible to have its power managed by PBS.

set by Manager. Readable by all.

Format: Boolean

Default: False (not eligible)

***power\_provisioning***

Server attribute. Reflects use of power profiles and managing node power via PBS. Set by PBS to *True* when the `PBS_power` hook is enabled.

Set by PBS. Read-only.

Format: Boolean

Default: *unset*, which behaves like *False* (not enabled)

***power\_provisioning***

Vnode attribute. Specifies whether this node is eligible to have its power managed by PBS, including whether it can use power profiles.

set by Manager. Readable by all.

Format: Boolean

Default: *False* (not eligible)

***sleep***

Vnode state. Indicates that this vnode was ramped down or powered off via PBS power management. This tells the scheduler that it can schedule jobs on this vnode; in that case PBS powers the vnode back up.

## 15.3 Caveats and Restrictions for Power Management

- If a reservation is created with a start time coming up soon, where the reservation requires nodes that are currently powered off, the reservation may start in degraded mode until all of the nodes can be powered up.
- Do not set `resources_available.eoe` on vnodes. This is handled by PBS.



# 16

## Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

### 16.1 Introduction

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

### 16.2 Definitions

#### **AOE**

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

#### **Master Provisioning Script, Master Script**

The script that makes up the provisioning hook

#### **Provision**

To install an OS or application, or to run a script which performs installation and/or setup

#### **Provisioning Hook**

The hook which performs the provisioning, either by calling other scripts or running commands

#### **Provisioning Tool**

A tool that performs the actual provisioning, e.g. HPE Performance Cluster Manager (HPCM)

#### **Provisioned Vnode**

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

### 16.3 How Provisioning Can Be Used

- Each application requires specific version of OS

The site runs multiple applications, and each application may be certified to run on a specific OS. In this situation, a job that will run an application requiring a specific OS requests the OS, and PBS provisions the required OS.

- The site needs differently configured images of the same OS to be loaded at different times

The site has multiple projects, and each project requires the OS to be configured in a different way on a group of hosts. In this situation, PBS provisions groups of hosts with the correct OS image, for the time period needed by each project. The OS image is configured and supplied by the site administrator.

- The entire site needs different OSes at different times of day

The entire site runs one OS during certain hours, and a different OS at other times.

- A user reserves multiple vnodes running the same version of an OS

A user may need a specific version of an OS for a period of time. For example, a user needs 5 nodes running a specific version of RHEL from 5pm Friday until 5am Monday.

- The administrator wants to limit the number of hosts that are being provisioned at any one time, for any of the following reasons:
  - The network can become overwhelmed transferring OS images to execution nodes
  - The hosts can draw excessive power if many are powering up at the same time
  - Some sites notify the administrator whenever an execution node goes down, and when several vnodes are provisioned, the administrator is paged repeatedly

## 16.4 How Provisioning Works

### 16.4.1 Overview of Provisioning

PBS allows you to create a provisioning hook, which is a hook that is triggered by a provisioning event. When this hook is triggered, it manages the required provisioning on the vnodes to be provisioned. The hook calls a provisioning mechanism such as HPE Performance Cluster Manager to accomplish the provisioning. The provisioning hook runs at the PBS server.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

PBS allows you to configure each vnode with a list of available AOE's. This list is specified in the vnode's `resources_available.aoe` resource. Each vnode's `current_aoe` attribute shows that vnode's current AOE. The scheduler queries each vnode's `aoe` resource and `current_aoe` attribute in order to determine which vnodes to provision for each job.

When users submit jobs, they can request a specific AOE for each job. When the scheduler runs each job, it either finds the vnodes that satisfy the job's requirements, or provisions the required vnodes.

Users can create reservations that request AOE's. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE.

#### 16.4.1.1 Rebooting When Provisioning

When provisioning a vnode with some AOE's, the vnode must be rebooted as part of the provisioning process. Some OS installations, for example, require rebooting. In this case, the provisioning script must cause the vnode to be rebooted.

When the installation does not require a reboot, the provisioning script does not need to cause the vnode to be rebooted. For example, provisioning with some applications does not require a reboot.

## 16.4.2 How Vnodes Are Selected for Provisioning

Each job can request at most one AOE. When scheduling the job, PBS looks for vnodes with the requested AOE, as with any other resource. If there are not enough vnodes with the requested AOE, PBS tries to provision vnodes in order to satisfy the job's requirements.

### 16.4.2.1 Provisioning Policy

PBS allows a choice of provisioning policies. You set the scheduler's `provision_policy` configuration parameter to be either `"avoid_provision"` or `"aggressive_provision"`. The default provisioning policy is `"aggressive_provision"`.

#### `avoid_provision`

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses `node_sort_key` to sort these vnodes.

If it cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, it does the following:

- PBS uses `node_sort_key` to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.
- Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

#### `aggressive_provision`

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses `node_sort_key` to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

### 16.4.2.2 Examples of Vnode Selection

The following examples show how provisioning policy can affect which vnodes are selected for provisioning.

Example 16-1: 3 vnodes

In `sched_config`:

```
node_sort_key: "ncpus HIGH"
```

We have 3 nodes as described in the following table:

**Table 16-1: Example Configuration**

| Vnode/Host | Number of CPUs | Current AOE | State       |
|------------|----------------|-------------|-------------|
| host1      | 1              | aoe1        | <i>free</i> |
| host2      | 2              | unset       | <i>free</i> |
| host3      | 3              | aoe2        | <i>free</i> |

No jobs are running on any of the vnodes.

Case 1: aggressive provisioning

```
provision_policy: "aggressive_provision"
```

```
Job submitted with -lselect=ncpus=1:aoe=aoe1
```

In this case, host3 is used to run the job and host3 is provisioned.



Case 2: avoiding provisioning

`provision_policy: "avoid_provision"`

Job submitted with `-lselect=ncpus=1:aoe=aoe1`

In this case, host1 is used to run the job and host1 is not provisioned.

Example 16-2: 5 vnodes

The following table shows the example configuration:

**Table 16-2: Example Configuration**

| Vnode/Host | AOE Available | Current AOE | State       |
|------------|---------------|-------------|-------------|
| N1         | aoe1, aoe2    | aoe1        | <i>busy</i> |
| N2         | aoe1, aoe2    | aoe2        | <i>free</i> |
| N3         | aoe1, aoe2    | NULL        | <i>free</i> |
| N4         | aoe1, aoe2    | aoe1        | <i>free</i> |
| N5         | aoe1, aoe2    | aoe1        | <i>free</i> |

The vnodes are sorted in the order N1, N2, N3, N4, N5.

A job is submitted with:

`qsub -lselect=3:ncpus=1:aoe=aoe1 -lplace=scatter`

The job needs three vnodes with aoe1. Assume that all other requests except that for the AOE can be satisfied by any vnode.

Case 1: aggressive provisioning

The scheduler selects N2, N3 and N4. It has not considered the AOE instantiated on these vnodes. It then provisions N2 and N3 since N2 has a different AOE instantiated on it and N3 is not provisioned yet. N4 is not provisioned, because it has the requested AOE already instantiated.

Case 2: avoiding provisioning

First, the scheduler selects N4 and N5. It does not choose N2 since it has a different AOE instantiated, and it does not choose N3 since it does not have any AOE instantiated. But N4 and N5 together do not satisfy the job's requirement of 3 vnodes.

Second, the scheduler seeks vnodes that if provisioned can satisfy the job's request. N2 and N3 can each satisfy the job's request, so it chooses N2, because it comes first in sorted order.

The job runs on N4, N5 and N2. N2 is provisioned.

### 16.4.2.3 Rules for Vnode Selection for Provisioning

A vnode is not selected for provisioning for the following reasons:

- It does not have the requested AOE available in its list
- It does not have provisioning enabled on it
- It has other running or suspended jobs
- It already has the requested AOE

---

### 16.4.2.4 Triggering Provisioning

When a job requires a vnode, and the vnode's `current_aoe` attribute is unset, or is set to a different AOE from the one requested, the vnode is provisioned.

## 16.4.3 Provisioning And Reservations

### 16.4.3.1 Creating Reservations that Request AOE

A reservation can request at most one AOE.

When a user creates a reservation that requests an AOE, the scheduler searches for vnodes that can satisfy the reservation. When searching, the scheduler follows the rule specified in the `provision_policy` scheduling parameter in `<sched_priv_directory>/sched_config`. See the `pbs_sched(8B)` manual page.

The vnodes allocated to a reservation that requests an AOE are put in the `resv-exclusive` state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

### 16.4.3.2 Submitting Jobs to a Reservation

If a job that requests an AOE is submitted to a reservation, the reservation must request the same AOE.

### 16.4.3.3 Running a Job in a Reservation Having a Requested AOE

A job can run in a reservation that has requested an AOE, as long as the job fits the following criteria:

- It requests the same AOE as the reservation

If the job has requested no AOE, or an AOE different from that of the reservation, the job is rejected.

## 16.4.4 How Provisioning Affects Jobs

### 16.4.4.1 Preemption and Provisioning

A job that has requested an AOE will not preempt another job, regardless of whether the job's requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOE.

### 16.4.4.2 Backfilling and Provisioning

If the job being backfilled around or the job doing the backfilling share a vnode, a job that has requested an AOE will not play any part in backfilling:

- It will not be backfilled around by smaller jobs.
- It will not be used as the job that backfills around another job.

### 16.4.4.3 Walltime and Provisioning

A job's walltime clock is started after provisioning is over.

### 16.4.4.4 Using qrun

When a job requesting an AOE is run via `qrun -H`, the following happens:

- If the requested AOE is available on the specified vnodes, those vnodes are provisioned with the requested AOE
- If the requested AOE is not available on the specified vnodes, the job is held

## 16.4.5 Vnode States and Provisioning

### 16.4.5.1 States Associated With Provisioning

The following vnode states are associated with provisioning:

#### provisioning

A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.

#### wait-provision

There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the server's `max_concurrent_provision` attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the *wait-provisioning* state. No jobs are run on vnodes in the *wait-provisioning* state.

#### resv-exclusive

The vnodes allocated to a reservation that requests an AOE are put in the *resv-exclusive* state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

### 16.4.5.2 Provisioning Process

The following table describes how provisioning and vnode state transitions interact:

**Table 16-3: Vnode State Transitions and Provisioning**

| Event                                                                                                                                                                                                 | Starting Vnode State     | Ending Vnode State       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|--------------------------|
| Vnode is selected for provisioning                                                                                                                                                                    | <i>free</i>              | <i>provisioning</i>      |
| Provisioning on vnode finishes                                                                                                                                                                        | <i>provisioning</i>      | <i>free</i>              |
| 1. Job running on this vnode leaving some resources available<br>2. No job running on this vnode                                                                                                      | <i>free</i>              | <i>free</i>              |
| Job running on this vnode, using all resources                                                                                                                                                        | <i>free</i>              | <i>job-busy</i>          |
| Vnode is selected for provisioning, but other vnodes being provisioned have already reached maximum allowed number of concurrently provisioning vnodes                                                | <i>free</i>              | <i>wait-provisioning</i> |
| This vnode is waiting to be provisioned for a multi-vnode job, and provisioning fails for another of the job's vnodes                                                                                 | <i>wait-provisioning</i> | <i>free</i>              |
| Provisioning fails for this vnode                                                                                                                                                                     | <i>provisioning</i>      | <i>offline</i>           |
| This vnode is waiting to be provisioned, and another vnode finishes provisioning, bringing the total number of provisioning vnodes below the limit specified in <code>max_concurrent_provision</code> | <i>wait-provisioning</i> | <i>provisioning</i>      |

### 16.4.5.3 Vnode State When Provisioning Fails

If provisioning fails on a vnode, that vnode is put into the *offline* state.

If provisioning for a multi-vnode job fails on one vnode, any vnodes in the *wait-provisioning* state are put into the *free* state.

### 16.4.5.4 Using the `qmgr` Command on Vnodes In Process of Provisioning

The following changes cannot be made to a provisioning vnode (a vnode in the *provisioning* state):

- Changing value of `current_aoe` vnode attribute
- Modifying resource `resources_available.aoe`
- Changing the state of the vnode. The `qmgr` command returns an error if this is attempted.
- Deleting the vnode from the server. The `qmgr` command returns an error if this is attempted.

The following can be modified while a vnode is provisioning:

- The server's `max_concurrent_provision` attribute
- A provisioning vnode's `provision_enable` attribute

The following cannot be set on the server host:

- `current_aoe` vnode attribute
- `provision_enable` vnode attribute
- The `resources_available.aoe` resource

## 16.4.6 Attributes, Resources, and Parameters Affecting Provisioning

### 16.4.6.1 Host-level Resources

#### `aoe`

The built-in `aoe` resource is a list of AOE's available on a vnode. Case-sensitive. You specify the list of AOE's that can be requested on a vnode by setting the value of `resources_available.aoe` to that list. Each job can request at most one AOE.

Automatically added to the "resources" line in `<sched_priv directory>/sched_config`.

Cannot be modified while a vnode is provisioning.

Non-consumable. Cannot be set on the server host. Can be set only by a Manager.

Format: `string_array`.

Default: unset.

Python attribute value type: `str`

---

### 16.4.6.2 Vnode Attributes

#### current\_aoe

The `current_aoe` vnode attribute shows which AOE is currently instantiated on a vnode. Case-sensitive.

At startup, each vnode's `current_aoe` attribute is unset. You must set the attribute to the currently instantiated AOE if you want the scheduler to be able to choose vnodes efficiently.

The value of this attribute is set automatically after a vnode is provisioned.

This attribute cannot be modified while a vnode is provisioning.

Cannot be set on the server host. Settable by Manager only; visible to all.

Format: String.

Default: Unset.

#### provision\_enable

This attribute controls whether the vnode can be provisioned. If set to *True*, the vnode can be provisioned.

Cannot be set on the server host.

Settable by Manager only; visible to all.

Format: Boolean

Default: Unset

### 16.4.6.3 Server Attributes

#### max\_concurrent\_provision

The maximum number of vnodes allowed to be in the process of being provisioned. Settable by Manager only; readable by all. When unset, default value is used. Cannot be set to zero; previous value is retained.

Format: Integer

Default: 5

Python attribute value type: *int*

### 16.4.6.4 Hook Attributes

All attributes of the provisioning hook affect provisioning. See [“Hook Attributes” on page 350 of the PBS Professional Reference Guide](#).

### 16.4.6.5 Scheduler Configuration Parameters

#### provision\_policy

Specifies the provisioning policy to be used. Valid values: *avoid\_provision*, *aggressive\_provision*.

See [section 16.4.2.1, “Provisioning Policy”, on page 593](#).

#### avoid\_provision

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses `node_sort_key` to sort these vnodes.

If it cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, it does the following:

PBS uses `node_sort_key` to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

**aggressive\_provision**

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses `node_sort_key` to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Default: "*aggressive\_provision*".

## 16.5 Configuring Provisioning

### 16.5.1 Overview of Configuring Provisioning

The administrator configures provisioning attributes, provides a provisioning tool, and writes a provisioning hook. The administrator configures each vnode to be provisioned with a list of AOE resources, where each resource is an AOE that is available to be run on that vnode. These resources are tags that tell the scheduler what can be run on that vnode. The administrator should also inform the scheduler about the current environment on each vnode, by setting the vnode's `current_aoe` attribute. It is also necessary to enable provisioning on each vnode to be provisioned and to set provisioning policy at the server and scheduler.

#### 16.5.1.1 Steps in Configuring Provisioning

These are the steps that the administrator must take:

1. Provide a provisioning tool such as HPE Performance Cluster Manager (HPCM). See [section 16.5.2, “Provide a Provisioning Tool”, on page 599](#).
2. Prepare each OS, application, or script that is to be used in provisioning. See [section 16.5.3, “Prepare Images”, on page 600](#).
3. Configure each vnode to be provisioned with the appropriate resources. See [section 16.5.4, “Define aoe Resources”, on page 600](#).
4. Optional: publish each vnode's current AOE. See [section 16.5.5, “Inform Scheduler of Current AOE”, on page 600](#).
5. Write the provisioning hook's script. See [section 16.5.6, “Write the Provisioning Script”, on page 601](#).
6. Create the empty provisioning hook, import the script, and configure the hook. See [section 16.5.7, “Create and Configure the Provisioning Hook”, on page 602](#).
7. Configure provisioning policy. See [section 16.5.8, “Configure Provisioning Policy”, on page 603](#).
8. Enable provisioning on vnodes. See [section 16.5.9, “Enable Provisioning on Vnodes”, on page 604](#).
9. Enable the provisioning hook. See [section 16.5.10, “Enable Provisioning Hook”, on page 604](#).

### 16.5.2 Provide a Provisioning Tool

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. This provisioning tool can either be written into the provisioning hook script, or be a separate script that is called by the provisioning hook script. You can write the provisioning tool yourself, or you can use something like the HPE Performance Cluster Manager (HPCM) cluster management tool. Your provisioning tool may be able to employ network-accessible power control units.

### 16.5.3 Prepare Images

You must prepare each image, application, or script you will use. Make sure that each is available to the target vnode. For example, if you use a diskless node server, put your images on the diskless node server.

The values for the `ncpus` and `mem` resources must be the same for all OS images that may be instantiated on a given vnode.

### 16.5.4 Define aoe Resources

The `aoe` resource is of type `string_array`, and is used to hold the names of the AOE's available at each vnode. This resource is not consumable. This resource is unset by default, and by default is added to the `resources` line in `<sched_priv directory>/sched_config`. See [“Resources Built Into PBS” on page 265 of the PBS Professional Reference Guide](#). The `aoe` resource is visible to all, but settable by the PBS Manager and Operator only.

The scheduler must be able to find out which AOE's can be run on which vnodes. To tag each vnode with the AOE's that can run on it, set that vnode's `resources_available.aoe` attribute to the list of available AOE's. For example, if vnode V1 is to run RHEL and SLES, and the hook script will recognize *rhel* and *sles*, set the vnode's `resources_available.aoe` attribute to show this:

```
Qmgr: set node V1 resources_available.aoe = "rhel, sles"
```

It is recommended that you make a list of all of the AOE's that may be used in provisioning in your PBS complex. The list is to facilitate script writing and resource configuration. Each entry in this list should contain at least the following information:

- Full description of the AOE
- Resource name of the AOE
- Vnodes that are to run the AOE
- Location where script should look for the AOE

For example, the list might look like the following table:

**Table 16-4: Example AOE List**

| Description                       | Resource Name   | Vnodes                        | Location                                     |
|-----------------------------------|-----------------|-------------------------------|----------------------------------------------|
| SuSE SLES 12 64-bit               | <i>sles12</i>   | mars, jupiter, neptune, pluto | imageserver.example.com:/images/sles12-image |
| SuSE SLES 15 64-bit               | <i>sles15</i>   | mars, jupiter, pluto          | imageserver.example.com:/images/sles15-image |
| Red Hat Enterprise Linux 8 64-bit | <i>rhel7</i>    | luna, aitne, io               | imageserver.example.com:/images/rhel8-image  |
| Windows Server 2016 64-bit        | <i>winsrv16</i> | luna, aitne, io               | \\WinServer\ C:\images\winsrv16              |

### 16.5.5 Inform Scheduler of Current AOE

Each vnode has an attribute called `current_aoe` which is used to tell the scheduler what the vnode's current AOE is. This attribute is unset by default. The attribute is of type `string`. It is visible to all, but settable by the PBS Manager only.

You can set this attribute on each vnode that will be used in provisioning. Set it to the value of the AOE that is currently instantiated on the vnode. So for example, using the table in [section 16.5.4, “Define aoe Resources”, on page 600](#), if vnode `pluto` is running 64-bit SuSE SLES 15, set `current_aoe` to `sles15`:

```
Qmgr: set node pluto current_aoe = sles15
```

When PBS provisions a vnode with a new AOE, the PBS server sets the value of `current_aoe` to the new AOE.

If PBS cannot provision a vnode with the desired AOE, it marks the vnode *offline* and unsets the value of `current_aoe`.

## 16.5.6 Write the Provisioning Script

You create the provisioning hook using a provisioning script which must manage all provisioning, either directly, or indirectly by calling other scripts. The script in the hook is the master provisioning script.

The script that does the provisioning must have the logic needed to provision the specified vnode with the specified AOE.

There are two types of provisioning. One is when the vnode is rebooted after installing/uninstalling the OS/application or running the script. The other is when the vnode is not rebooted after installing/uninstalling the OS/application or running the script.

The master provisioning script must meet the following requirements:

- Written in Python
- Arguments to the script are the vnode name and the AOE name
- If the vnode must be rebooted for provisioning, the provisioning script must cause the target vnode to be rebooted
- Must indicate success using the correct return value:
  - Return `pbs.event.accept(0)` if provisioning is successful and the vnode is rebooted
  - Return `pbs.event.accept(1)` if provisioning is successful and the vnode is not rebooted
- Must indicate failure to PBS by using `pbs.event.reject()`
- If the master provisioning script calls other scripts, it must wait for them to finish before returning success or failure to PBS

### 16.5.6.1 Arguments to Master Script

The arguments to the master script are the following:

- Name of vnode to be provisioned  
Supplied to the hook via the PBS provision event object, as `pbs.event.vnode`
- Name of AOE to be instantiated on the target vnode  
Supplied to the hook via the PBS provision event object, as `pbs.event.aoe`

These values can be passed to scripts that are called by the master script.

### 16.5.6.2 Return Values

The master script must indicate to PBS whether it succeeded or failed in a way that PBS can understand.



### 16.5.6.2.i Success

By default, `pbs.event.accept()` returns zero. The script must return different values for successful provisioning, depending on whether the vnode is rebooted:

- If provisioning is successful and the vnode is rebooted, the script must return *0* (zero) to PBS via `pbs.event.accept(0)`.
- If provisioning is successful and the vnode is not rebooted, the script must return *1* (one) to PBS via `pbs.event.accept(1)`.

### 16.5.6.2.ii Failure

If provisioning fails, the script must use `pbs.event.reject()` to indicate failure. By default, `pbs.event.reject()` returns 255. To return another failure code, use the following:

```
pbs.event.reject(error message, error code)
```

where error code is any number between 2 and 255. Returning an error code in `pbs.event.reject()` is optional.

## 16.5.6.3 Master Script Calls Subscript

Often, the master script (the hook script) calls another script, depending on the provisioning required. The subscript does the actual provisioning of the target vnode with the requested AOE. In this case, the master script must wait for the subscript to return and indicate success or failure. The master script then propagates the result to PBS.

Example of a fragment of a master script calling a subscript:

```
return_value = os.system("/var/vendor/vendor_prov.sh " <arguments to vendor_prov.sh>)
```

## 16.5.7 Create and Configure the Provisioning Hook

The provisioning hook causes any provisioning to happen. The provisioning hook is a Python script which either does the provisioning directly or calls other scripts or tools. Typically the provisioning hook calls other scripts, which do the actual work of provisioning. For complete information on writing hooks, see the PBS Professional Hooks Guide.

You can have at most one provisioning hook. Do not attempt to create more than one provisioning hook.

In the steps that follow, we use as examples a provisioning hook named "Provision\_Hook", and an ASCII script named "master\_provision.py".

### 16.5.7.1 Create the Hook

To create the provisioning hook:

```
Qmgr: create hook <hook name>
```

For example, to create a provisioning hook called Provision\_Hook:

```
Qmgr: create hook Provision_Hook
```

### 16.5.7.2 Import the Hook Script

If the hook script is called "master\_provision.py", and it is ASCII, and it is located in `/root/data/`, importing the hook script looks like this:

```
Qmgr: import hook Provision_Hook application/x-python default
 /root/data/master_provision.py
```

See ["Importing Hooks" on page 35 in the PBS Professional Hooks Guide](#) for more about importing hooks.

### 16.5.7.3 Configure the Hook Script

#### 16.5.7.3.i Set Event Type

The event type for the provisioning hook is called "*provision*". To set the event type:

```
Qmgr: set hook Provision_Hook event = provision
```

Do not try to assign more than one event type to the provisioning hook.

#### 16.5.7.3.ii Set Alarm Time

The default alarm time for hooks is *30 seconds*. This may be too short for a provisioning hook. You should set the alarm time to a value that is slightly more than the longest time required for provisioning. Test provisioning each AOE, and find the longest time required, then add a small amount of extra time. To set the alarm time:

```
Qmgr: set hook Provision_Hook alarm = <number of seconds required>
```

## 16.5.8 Configure Provisioning Policy

### 16.5.8.1 Set Maximum Number of Concurrently Provisioning Vnodes

The value of the server's `max_concurrent_provision` attribute specifies the largest number of vnodes that can be in the process of provisioning at any time. The default value of this attribute is *5*. Set the value of this attribute to the largest number of vnodes you wish to have concurrently provisioning. See [section 16.4.6.3, "Server Attributes", on page 598](#) for more information on the attribute.

#### 16.5.8.1.i Considerations

You may wish to limit the number of hosts that can be in the process of provisioning at the same time:

- So that the network isn't overwhelmed transferring OS images to execution nodes
- So the hosts won't draw excessive power when powering up at the same time

Many sites have tools that notify them when an execution node goes down. You may want to avoid being paged every time an execution node is provisioned with a new AOE.

### 16.5.8.2 Set Scheduling Policy

When a job is scheduled to be run, and the job requests an AOE, PBS can either try to fit the job on vnodes that already have that AOE instantiated, or it can choose the vnodes regardless of AOE. Choosing regardless of AOE is the default behavior; the assumption is that the chances of finding free vnodes that match all the requirements including that of the requested AOE are not very high.

Provisioning policy is controlled by the `provision_policy` scheduling parameter in `<sched_priv directory>/sched_config`. This parameter is a string which can take one of two values: *avoid\_provision* or *aggressive\_provision*. If you want PBS to try first to use vnodes whose AOE's already match the requested AOE, set `provision_policy` to *avoid\_provision*. If you want PBS to choose vnodes regardless of instantiated AOE, set it to *aggressive\_provision*.

For details about the `provision_policy` parameter, see [section 16.4.2.1, "Provisioning Policy", on page 593](#).

For jobs that do not request an AOE, `node_sort_key` is used to choose vnodes.

---

## 16.5.9 Enable Provisioning on Vnodes

PBS will provision only those vnodes that have provisioning enabled. Provisioning on each vnode is controlled by its `provision_enable` attribute. This attribute is Boolean, with a default value of *False*. You enable provisioning on a vnode by setting its `provision_enable` attribute to *True*.

This attribute cannot be set to *True* on the server host.

See [section 16.4.6.2, “Vnode Attributes”, on page 598](#) for details about the `provision_enable` vnode attribute.

## 16.5.10 Enable Provisioning Hook

The last step in configuring provisioning is enabling the provisioning hook. The provisioning hook is enabled when its enabled attribute is set to *True*. To set the enabled attribute to *True* for the provisioning hook named `Provision_Hook`:

```
Qmgr: set hook Provision_Hook enabled = True
```

## 16.6 Viewing Provisioning Information

### 16.6.1 Viewing Provisioning Hook Contents

To see the contents of the provisioning hook, export them:

```
qmgr -c "export hook <hook name> application/x-python default" > <output-path>/<output-filename>
```

For example, if the provisioning hook is named `Provision_Hook`, and you wish to export the contents to `/usr/user1/hook_contents`:

```
qmgr -c "export hook Provision_Hook application/x-python default" > /usr/user1/hook_contents
```

### 16.6.2 Viewing Provisioning Hook Attributes

To view the provisioning hook's attributes, use the `list hook` option to the `qmgr` command:

```
qmgr -c "list hook <hook name>"
```

---

## 16.6.3 Printing Provisioning Hook Creation Commands

To print the provisioning hook's creation commands, use the `print hook` option to the `qmgr` command:

```
qmgr -c "p hook"
#
Create hooks and set their properties.
#
#
Create and define hook my_prov_hook
#
create hook my_prov_hook
set hook my_prov_hook type = site
set hook my_prov_hook enabled = True
set hook my_prov_hook event = provision
set hook my_prov_hook user = pbsadmin
set hook my_prov_hook alarm = 30
set hook my_prov_hook order = 1
import hook my_prov_hook application/x-python base64 -
c2xzbgwK
```

## 16.6.4 Viewing Attributes and Resources Affecting Provisioning

### 16.6.4.1 Server Attributes

To see the server attributes affecting provisioning, print the server's information using the `qmgr` command:

```
qmgr -c "print server"
```

You will see output similar to the following:

```
qmgr
Max open servers: 49
Qmgr: p s
#
Create queues and set their attributes.
#
#
Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
Set server attributes.
#
set server scheduling = True
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server resv_enable = True
set server node_fail_requeue = 310
set server pbs_license_min = 0
set server pbs_license_max = 2147483647
set server pbs_license_linger_time = 31536000
set server license_count = "Avail_Global:0 Avail_Local:256 Used:0 High_Use:0"
set server max_concurrent_provision = 5
```

---

### 16.6.4.2 Viewing Vnode Attributes and Resources

To see vnode attributes and resources affecting provisioning, use the `-a` option to the `pbsnodes` command:

```
pbsnodes -a
host1
 Mom = host1
 ntype = PBS
 state = free
 pcpus = 2
 resources_available.aoe = osimage1, osimage2
 resources_available.arch = linux
 resources_available.host = host1
 resources_available.mem = 2056160kb
 resources_available.ncpus = 2
 resources_available.vnode = host1
 resources_assigned.mem = 0kb
 resources_assigned.ncpus = 0
 resources_assigned.vmem = 0kb
 resv_enable = True
 sharing = default_shared
 provision_enable = True
 current_aoe = osimage2
```

## 16.7 Requirements and Restrictions

### 16.7.1 Site Requirements

#### 16.7.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

#### 16.7.1.2 Provisioning Tool Required

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. Examples of provisioning tools are the following:

- The HPE Performance Cluster Manager (HPCM) cluster management tool
- Dual boot system
- Network-accessible power control units

#### 16.7.1.3 Single Provisioning Hook Allowed

The PBS server allows only one provisioning hook. If you have an existing provisioning hook and you import a provisioning script, that script will become the contents of the hook, whether or not the hook already has a script. The new script will overwrite the existing provisioning hook script.

---

### 16.7.1.4 Provisioning Hook Cannot Have Multiple Event Types

The provisioning hook cannot have more than one event type.

### 16.7.1.5 AOE Names Consistent Across Complex

Make AOE names consistent across the complex. The same AOE should have the same name everywhere.

## 16.7.2 Usage Requirements

### 16.7.2.1 Restriction on Concurrent AOE's on Vnode

Only one AOE can be instantiated at a time on a vnode.

Only one kind of `aoe` resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

### 16.7.2.2 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

### 16.7.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOE's are requested by the two occurrences.

### 16.7.2.4 Hook Script and AOE Must Be Compatible

The requested AOE must be available to the vnode to be provisioned. The following must be *True*:

- The AOE must be in the list of available AOE's for the vnode
- Each AOE listed on a vnode must be recognized by the provisioning hook script.
- The vnode must have provisioning enabled

### 16.7.2.5 Provisioning Hook Must Be Ready

- The provisioning hook must obey the following rules:
  - It must exist
  - It must have a Python script imported
  - It must be enabled
  - It must be designed to invoke an external script or command for AOE's that are to be used

---

### 16.7.2.6 Server Host Cannot Be Provisioned

The server host cannot be provisioned: a MoM can run on the server host, but that MoM's vnode cannot be provisioned. The `provision_enable` vnode attribute, `resources_available.aoe`, and `current_aoe` cannot be set on the server host.

### 16.7.2.7 PBS Attributes Not Available to Provisioning Hook

The provisioning hook cannot operate on PBS attributes except for the following:

- The name of the vnode to be provisioned: `pbs.event.vnode`
- The AOE to be instantiated: `pbs.event.aoe`

### 16.7.2.8 `avoid_provision` Incompatible with `smp_cluster_dist`

The `avoid_provision` provisioning policy is incompatible with the `smp_cluster_dist` scheduling scheduler configuration parameter. If a job requests an AOE, the `avoid_provision` policy overrides the behavior of `smp_cluster_dist`.

## 16.8 Defaults and Backward Compatibility

By default, PBS does not provide provisioning. You must configure PBS to provide provisioning.

## 16.9 Example Scripts

### 16.9.1 Sample Master Provisioning Hook Script With Explanation

We show a sample provisioning hook script, and an explanation of what the script does. For readability, the sample script is a master script calling two subscripts.

This provisioning hook allows two kinds of provisioning request:

- For the application AOE named "App1", via the script `app_prov.sh`  
The `app_prov.sh` script does not reboot the vnode
- For other provisioning, via the vendor-provided provisioning shell script `vendorprov.sh`  
The `vendorprov.sh` script reboots the vnode



### 16.9.1.1 Sample Master Provisioning Hook Script

```

import pbs (1)
import os (2)

e = pbs.event() (3)
vnode = e.vnode (4)
aoe = e.aoe (5)

if (aoe == "App1"): (6)
 appret = os.system("/var/user/app_prov.sh
 " + vnode + " " + aoe) (7)
 if appret != 1: (8)
 e.reject("Provisioning without reboot
 failed", 210) (9)
 else:
 e.accept(1) (10)

ret = os.system("/var/vendor/vendorprov.sh
 " + vnode + " " + aoe) (11)

if ret != 0: (12)
 e.reject("Provisioning with reboot
 failed", 211) (13)
else:
 e.accept(0) (14)

```

### 16.9.1.2 Explanation of Sample Provisioning Hook Script

- Lines 1 and 2 import the `pbs` and `os` modules.
- Line 3 puts the PBS provisioning event into the local variable named "e".
- Lines 4 and 5 store the target vnode name and the name of the AOE to be instantiated on the target vnode in local variables.
- Line 6 checks whether provisioning of the application AOE named "App1" is requested.
- Line 7 is where the actual code to do non-rebooting provisioning could go. In this example, we call a subscript, passing the name of the target vnode and the requested AOE, and storing the return value in "appret".  
The non-rebooting provisioning subscript should return 1 on success.
- Line 8 checks whether non-rebooting provisioning via `app_prov.sh` succeeded.
- Line 9 returns the error code 210 and an error message to PBS if `app_prov.sh` failed.
- Line 10 returns 1 via `pbs.event.accept(1)` if non-rebooting provisioning succeeded.
- Line 11 calls the vendor-supplied script that is responsible for doing rebooting provisioning whenever "App1" is not the AOE.

The name of the target vnode and the requested AOE are passed to this script.

The vendor-supplied script should expect these two arguments. The return value from this script is stored in the variable named "ret".

- Line 12 checks whether rebooting provisioning via the vendor-supplied script `vendorprov.sh` was successful.
- Line 13: If the return value is anything but `zero` (success), the provisioning hook script passes the error code `211` back to PBS, along with an error message.
- Line 14 returns success to PBS via `pbs.event.accept(0)` and the master script exits.

## 16.9.2 Sample Master Provisioning Hook Script Calling Performance Cluster Manager

The following is a master provisioning hook script that calls HPE Performance Cluster Manager (HPCM):

```
-*- coding: utf-8 -*-
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe

if (aoe=="App1"):
 ret = os.system("/root/osprov/application.sh " + vnode + " " + aoe)
 if ret != 0:
 e.reject("Non-reboot provisioning failed",ret)
 else:
 e.accept(1)

ret = os.system("/root/osprov/sgi_provision.sh " + vnode + " " + aoe)
if ret != 0:
 e.reject("Reboot provisioning failed",ret)
else:
 e.accept(0)
```

## 16.9.3 Sample Script Set

This is a set of example Linux scripts designed to work together. They are the following:

`provision_hook.py`

This is the script for the provisioning hook. It calls the master provisioning script.

`provision_master.py:`

This is the master provisioning script. It is responsible for rebooting the machine being provisioned. It calls `update_grub.sh` to update the current AOE.

`update_grub.sh`

This shell script updates the linux `grub.conf` file and sets the value for `current_aoe` after the reboot.

The `update_grub.sh` script must be modified according to the grub configuration of the system in question before being run.

---

### 16.9.3.1 Provisioning Hook Script

provision\_hook.py:

```
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe
#print "vnode:" + vnode
#print "AOE:" + aoe

if (aoe=="App1"):
 print "Provisioning an application"
 e.accept(1)

ret = os.system("python /root/provision_master.py " + vnode + " " + aoe + " " + "lin")
#print "Python top level script returned " + str(ret)
if ret != 0:
 e.reject("Provisioning failed",ret)
else:
 e.accept(0)
```

---

### 16.9.3.2 Master Provisioning Script

provision\_master.py:

```
#!/usr/bin/python

#-----
success : 0
failure : 1
#-----
win_or_lin == 1 : windows
win_or_lin == 0 : linux
#-----
1 is TRUE
0 is FALSE
#-----

import sys
import os

vnode = sys.argv[1]
aoe = sys.argv[2]
win_or_lin = sys.argv[3]

print vnode, aoe

if not aoe.find('win'):
 print "aoe is win"
 isvnodewin = 1
else:
 print "aoe is *nix"
 isvnodewin = 0

print "win_or_lin = [", win_or_lin, "]"

if (win_or_lin == "win"):
 print "entering window server"
 if isvnodewin:
#----- WINDOWS -> WINDOWS
 ret = os.system("pbs-sleep 05")
#----- WINDOWS -> WINDOWS

 else:
#----- WINDOWS -> LINUX
 ret = os.system("pbs-sleep 05")
#----- WINDOWS -> LINUX
```

---

```

ret = os.system("pbs-sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
 ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
 if not ret:
 print "that machine is now up"
 exit(0)

 ticks = ticks + 1
 print "ticks = ", ticks
 if ticks > timeout:
 print "exit ticks = ", ticks
 print "that machine didn't come up after 2 mins, FAIL"
 exit(1)

else:
 print "entering linux server"
 if isvnodewin:
#----- LINUX -> WINDOWS
 ret = os.system("sleep 05")
#----- LINUX -> WINDOWS

 else:
#----- LINUX -> LINUX

 ret = os.system("scp -o StrictHostKeyChecking=no /root/update_grub.sh " + vnode + " :/root
> /dev/null 2>&1")
 if ret != 0:
 print "scp failed to copy"
 exit(1)

 ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + "
\"/root/update_grub.sh " + vnode + " " + aoe + " 1 " + " \" > /dev/null 2>&1")
 if ret != 0:
 print "failed to run script"
 exit(1)

 ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " \"reboot\" " + " >
/dev/null 2>&1")
 if ret != 0:
 print "failed to reboot that machine"
 exit(1)

#----- LINUX -> LINUX

```

```
ret = os.system("sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
 ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
 if not ret:
 print "that machine is now up"
 exit(0)

 print "ticks = ", ticks
 ticks = ticks + 1
 if ticks > timeout:
 print "That machine didn't come up after 2 mins. FAIL"
 exit(1)
```

---

### 16.9.3.3 Grub Update Shell Script

update\_grub.sh:

```
#!/bin/sh

if [$# -lt 2]; then
 echo "syntax: $0 <machine ip> <aoe name>"
 exit 1
fi

machine=$1
aoe_name=$2

menufile="/boot/grub/grub.conf"
if [! -f "$menufile"]; then
 echo "grub.conf file not found. $machine using grub bootloader?"
 exit 1
fi

link=`ls -l $menufile | cut -c1`
if ["$link" = "l"]; then
 menufile=`ls -l $menufile | awk -F"-"> '{print $2}'`
 echo "Found link file, original file is $menufile"
fi

titles=`cat $menufile | grep title | awk -F"title" '{print $2}' | sed 's/^[ \t]//g`
lines=`echo -e "$titles" | wc -l`

found_aoe_index=-1
count=0
while [$count -lt $lines]
do
 lineno=`expr $count + 1`
 title=`echo -e "$titles" | head -n $lineno | tail -n 1`
 if ["$aoe_name" = "$title"]; then
 found_aoe_index=$count
 fi
 count=`expr $count + 1`
done

if [$found_aoe_index = -1]; then
 echo "Requested AOE $aoe_name is not found on machine $machine"
 exit 2
fi

new_def_line="default=$found_aoe_index"
```

---

```
def_line=`cat $menufile | grep "^default="`

echo "new_def_line=$new_def_line"
echo "def_line=$def_line"
echo "menufile=$menufile"

cp $menufile /boot/grub/grub.conf.backup
cat $menufile | sed "s/^$def_line/$new_def_line/g" > grub.out
if [-s grub.out]; then
 mv grub.out $menufile
else
 exit 1
fi

service pbs stop

exit 0
```

## 16.10 Advice and Caveats

### 16.10.1 Using Provisioning Wisely

It is recommended that when using provisioning, you set PBS up so as to prevent things such as the following:

- User jobs not running because vnodes used in a reservation have been provisioned, and provisioning for the reservation job will take too long
- Excessive amounts of time being taken up by provisioning from one AOE to another and back again

In order to avoid problems like the above, you can do the following to keep specific AOE requests together:

- For each AOE, associate a set of vnodes with a queue. Use a hook to move jobs into the right queues.
- Create a reservation requesting each AOE, then use a hook to move jobs requesting AOE's into the correct reservation.



### 16.10.1.1 Preventing Provisioning

You may need to prevent specific users or groups from using provisioning. You can use a job submission, job modification, or reservation creation hook to prevent provisioning. For more about hooks, see the PBS Professional Hooks Guide. The following is an example of a hook script to prevent USER1 from provisioning:

```
import pbs
import re

#--- deny user access to provisioning

e = pbs.event()
j = e.job #--- Use e.resv to restrict provisioning in reservation
who = e.requestor

unallow_u list = ["USER1"]

if who not in unallow_u list
 e.accept(0)

#User request AOE in select?
if j.Resource_List["select"] != None:
 s = repr(j.Resource_List["select"])
 if re.search("aoe=", s) != None:
 pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to
 provision" % (who))
 e.reject("User not allowed to provision")

#User request AOE?
if j.Resource_List["aoe"] != None:
 pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to
 provision" % (who))
 e.reject("User not allowed to provision")

e.accept(0)
```

### 16.10.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

## 16.11 Errors and Logging

### 16.11.1 Errors

#### 16.11.1.1 Errors Resulting in Marking Vnodes Offline

A vnode is marked *offline* if:

- Provisioning fails for the vnode
- The AOE reported by the vnode does not match the requested AOE after the provisioning script finishes

A vnode is not marked *offline* if provisioning fails to start due to internal errors in the script.

#### 16.11.1.2 Errors Resulting in Requeueing Job

Before provisioning a vnode with a requested OS, the server checks to see whether MoM's hook files are synced. If not, the server creates a timed task to check again. If the server fails to create the timed task, it requeues the job, and logs following error messages at log level 0x0001:

```
"Resource temporarily unavailable (11) in prov_startjob, Unable to set task for prov_startjob;
 requeueing the job"
```

```
"Cannot allocate memory (12) in prov_startjob, Unable to set task for prov_startjob; requeueing the
 job"
```

```
"Cannot allocate memory (12) in check_and_run_jobs, Unable to set task for prov_startjob;
 requeueing the job"
```

### 16.11.2 Logging

#### 16.11.2.1 Accounting Logs

For each job and reservation, an accounting log entry is made whenever provisioning starts and provisioning ends. Each such log entry contains a list of the vnodes that were provisioned, the AOE that was provisioned on these vnodes, and the start and end time of provisioning.

The accounting log entry for the start of provisioning is identified by the header "*P*", and the entry for the end of provisioning is identified by the header "*p*".

Example:

Printed when job starts provisioning:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
 prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=START start_time=1231928746"
```

Printed when job stops provisioning:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
 prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=SUCCESS
 end_time=1231928812"
```

Printed when provisioning for job failed:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
 prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=FAILURE
 end_time=1231928812"
```

## 16.11.2.2 Server Logs

### 16.11.2.2.i Messages Printed at Log Level 0x0080

"vnode <vnode name>: Vnode offlined since it failed provisioning"  
 "vnode <vnode name>: Vnode offlined since server went down during provisioning"  
 "Provisioning for Job <job id> succeeded, running job"  
 "Job failed to start provisioning"  
 "Provisioning for Job <job id> failed, job held"  
 "Provisioning for Job <job id> failed, job queued"

### 16.11.2.2.ii Messages Printed at Log Level 0x0100

"Provisioning of Vnode <vnode name> successful"  
 "Provisioning of <vnode name> with <AOE name> for <job ID> failed, provisioning exit status=<number>"  
 "Provisioning of <vnode name> with <aoe name> for <job id> timed out"  
 "Provisioning vnode <vnode> with AOE <AOE> started successfully"  
 "provisioning error: AOE mis-match"  
 "provisioning error: vnode offline"

### 16.11.2.2.iii Messages Printed at Log Level 0x0002

"Provisioning hook not found"

### 16.11.2.2.iv Messages Printed at Log Level 0x0001

"Provisioning script recompilation failed"  
 "Resource temporarily unavailable (11) in prov\_startjob, Unable to set task for prov\_startjob; requeueing the job"  
 "Cannot allocate memory (12) in prov\_startjob, Unable to set task for prov\_startjob; requeueing the job"  
 "Cannot allocate memory (12) in check\_and\_run\_jobs, Unable to set task for prov\_startjob; requeueing the job"

## 16.11.2.3 Scheduler Logs

### 16.11.2.3.i Messages Printed at Log Level 0x0400

Printed when vnode cannot be selected for provisioning because requested AOE is not available on vnode:

"Cannot provision, requested AOE <aoe-name> not available on vnode"

Printed when vnode cannot be selected for provisioning because vnode has running or suspended jobs, or the reservation or job would conflict with an existing reservation:

"Provision conflict with existing job/reservation"

Printed when vnode cannot be selected for provisioning because `provision_enable` is unset or set *False* on vnode:

"Cannot provision, provisioning disabled on vnode"

Printed when job cannot run because server is not configured for provisioning:

"Cannot provision, provisioning disabled on server"

Printed when multiple vnodes are running on the host:

"Cannot provision, host has multiple vnodes"

Printed when vnodes are sorted according to `avoid_provision` policy:

"Re-sorted the nodes on aoe <aoe name>, since aoe was requested"

### 16.11.2.3.ii Messages Printed at Log Level 0x0100

Printed when a vnode is selected for provisioning by a job:

"Vnode <vnode name> selected for provisioning with <AOE name>"

## 16.11.3 Error Messages

Printed when vnode is provisioning and `current_aoe` is set or unset or `resources_available.aoe` is modified via `qmgr`:

"Cannot modify attribute while vnode is provisioning"

Printed when `qmgr` is used to change state of vnode which is currently provisioning:

"Cannot change state of provisioning vnode"

Printed when vnode is deleted via '`qmgr > delete node <name>`' while it is currently provisioning:

"Cannot delete vnode if vnode is provisioning"

Printed when `provision_enable`, `current_aoe` or `resources_available.aoe` are set on host running PBS server, scheduler, and communication daemons:

"Cannot set provisioning attribute on host running PBS server and scheduler"

Printed when `current_aoe` is set to an AOE name that is not listed in `resources_available.aoe` of the vnode:

"Current AOE does not match with `resources_available.aoe`"

Printed when an event of a hook is set to '*provision*' and there exists another hook that has event '*provision*':

"Another hook already has event '*provision*', only one '*provision*' hook allowed"

Printed when `qsub` has `-laoe` and `-lselect=aoe`:

"-lresource= cannot be used with "select" or "place", resource is: aoe"

Job comment printed when job fails to start provisioning:

"job held, provisioning failed to start"

Printed when job is submitted or altered so that it does not meet the requirements that all chunks must request same AOE, and this AOE must match that of any reservation to which the job is submitted:

"Invalid provisioning request in chunk(s)"



# 17

## Support for HPE

### 17.1 Support for HPE with Cpusets

#### 17.1.1 Briefly, How PBS Manages Cpusets

As of version 2020.1, PBS uses the standard MoM on HPE machines, and uses the cgroups hook to manage cpusets on HPE machines. See [Chapter 6, "Configuring and Using PBS with Cgroups", on page 311](#).

PBS automatically examines the topology of the machine, and creates child vnodes to represent subsets of the machine. PBS also organizes the machine's vnodes into placement sets. When PBS runs a job on an HPE execution host, the cgroups hook creates the cpuset in which the job runs, and destroys the cpuset after the job is finished.

#### 17.1.2 Cpusets and Vnodes

The PBS MoM represents a machine as a set of vnodes. Each vnode is visible via commands such as `pbsnodes`. Each vnode must have its own logical memory pool, so you get one vnode per logical memory pool. All of the vnodes on one multi-vnode host are managed by one instance of `pbs_mom`.

A cpuset is a group of CPUs and memory nodes around which an inescapable wall has been placed. The OS manages a cpuset so that processes executing within the cpuset are typically confined to use only the resources defined by the cpuset.

#### 17.1.3 Requirements for Managing Cpusets

If you want PBS to manage the cpusets on a machine:

- Use the cgroups hook
- You must use a supported version of HPE MPI
- You use the PBS start/stop script to start MoM instead of `pbs_mom`

#### 17.1.4 Where to Use Cpusets

Use PBS to manage your cpusets wherever you want jobs to be fenced into their own CPUs and memory. This can also be useful on other machines, such as the HPE 8600, depending on the individual machine.

#### 17.1.5 Settings for sharing Attribute

The cgroups hook sets the sharing attribute for each vnode as follows:

- On MC990X and Superdome Flex, the hook sets the sharing attribute for the parent vnode to *default\_shared*
- On MC990X and Superdome Flex, the hook sets the sharing attribute for all other vnodes to *default\_shared*
- On 8600, the hook sets the sharing attribute for each vnode to *default\_shared*

### 17.1.5.1 Creating Vnodes

We recommend using the cgroups hook to manage the machine and create any child vnodes. If you use the cgroups hook, do not create vnodes via a Version 2 configuration file. However, if you are not using the cgroups hook, you can create your vnode definitions by hand. You can have MoM create any child vnodes via a Version 2 configuration file. See [section 3.3, “Creating Vnodes”, on page 42](#).

#### 17.1.5.1.i Caveats for Creating Vnodes

Do not attempt to create more than one vnode per logical memory pool. Your jobs will not run correctly.

### 17.1.5.2 Configuring Vnodes

If necessary, you can modify child vnodes created by the hook, by using an `exechost_startup` hook or via a Version 2 configuration file. See [section 3.4, “Configuring Vnodes”, on page 45](#).

## 17.1.6 Comprehensive System Accounting

PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

## 17.2 Support for HPE Cray Shasta

PBS runs on HPE Cray Shasta exactly as it does on standard Linux machines. The only information in this chapter that applies to HPE Cray Shasta is contained within this section ([Section 17.2, “Support for HPE Cray Shasta”](#)). Each compute node behaves like a standard Linux machine, and runs one MoM. By default, each compute node is represented by one vnode. When you create vnodes on HPE Cray Shasta, use the host shortname as the vnode name.

### 17.2.1 HPE Cray Shasta Is Different from XC

If you are used to PBS on Cray XC machines, working with HPE Cray Shasta is different. Configuring PBS on HPE Cray Shasta is the same as on a standard Linux machine. You can ignore all of the Cray XC instructions in the rest of this chapter. For example:

- Batch mode and state do not apply
- Special Cray built-in resources do not apply to HPE Cray Management System
- You don't need to set `vntype`
- You don't need `node_fail_requeue` to be zero
- The `PBS_alps_inventory_check` hook is not used for HPE Cray Shasta
- You don't need `vnode_pool`
- Hyperthreads are the same as on a standard Linux machine

#### 17.2.1.1 Not Supported on HPE Cray Shasta

- Suspend/resume is not supported on HPE Cray Shasta
- Power awareness is not supported on HPE Cray Shasta

## 17.2.2 Hook for PBS on HPE Cray Shasta

On HPE Cray Shasta, PBS uses a built-in hook called *PBS\_cray\_atom*, which runs for *execjob\_begin* and *execjob\_end* events. The hook notifies the Cray when each job starts, and when each job should be deleted. This hook should be enabled by default, but we recommend making sure that it is.

If the hook alarms while running for the *execjob\_begin* event (POST and DELETE), the vnode(s) where the hook was running are marked offline.

If the hook alarms while running for the *execjob\_end* event (DELETE), the hook rejects the action. The default timeout for this hook is 300 seconds.

### 17.2.2.1 HPE Cray Shasta Hook Configuration File

The configuration file for the *PBS\_cray\_atom* hook is formatted as a JSON object. Here is the default configuration file:

```
{
 "post_timeout": 30,
 "delete_timeout": 30,
 "unix_socket_file": "/var/run/jacsd/jacsd.sock"
}
```

#### 17.2.2.1.i Configuration File Parameters

##### "post\_timeout"

Time limit for *POST* requests.

Units: *seconds*

Format: *float*

Default: *30 seconds*

##### "delete\_timeout"

Time limit for *DELETE* requests.

Units: *seconds*

Format: *float*

Default: *30 seconds*

##### "unix\_socket\_file"

Path to the UNIX socket file to be used for authentication.

Format: *string*

Default: *"/var/run/jacsd/jacsd.sock"*

## 17.2.3 Responding to Node Health

On HPE Cray Shasta, Cray tasks take care of marking nodes unavailable or available. If Cray tasks decide that node health is not acceptable, Cray tasks will bring down the PBS MoM on that node. After you restore the node to usability, you must restart the MoM. If there are running jobs, use the *pbs\_mom -p* option in order to preserve and track running jobs. See [“Impact of Stop-Restart on Running Linux Jobs” on page 154 in the PBS Professional Installation & Upgrade Guide](#).





# Support for NEC SX-Aurora TSUBASA

## 18.1 Vnodes for NEC SX-Aurora TSUBASA

The basic hardware unit for NEC SX-Aurora TSUBASA is a *vector host* (a standard x86 server) connected to a set of accelerators called *vector engines* (VEs) via optional PCIe. The unit can consist of one or more NUMA nodes. Each unit uses one or more host channel adapters to communicate with other units and with the rest of the world.

The increasing order of communication overhead is first within a vector engine, then between vector engines via a shared PCIe, then between vector engines via PCIs on a common vector host, and finally between vector engines on separate vector hosts.

PBS creates topology-aware vnodes by grouping each PCIe with its associated vector host and vector engines together into one vnode. If there is no PCIe, PBS groups the vector host and its VEs into a vnode. A NUMA node without its own PCIe is in its own vnode.

PBS automatically creates vnodes to represent the host topology when you start PBS on execution hosts after the built-in `PBS_sx_aurora` hook is enabled.

PBS tries to do topology-aware scheduling by grouping job processes on vector engines in a way that produces the lowest communication overhead. When a job requests vector engines, PBS tries to assign vector engines from a single vnode to minimize communication overhead between vector engines.

For how to request resources on NEC SX-Aurora TSUBASA, see ["Submitting Jobs to NEC SX-Aurora TSUBASA", on page 205 of the PBS Professional User's Guide](#).

## 18.2 Terminology

### HCA

Host channel adapter. Network interconnect used by vnode. Each vector host can have one or more HCAs.

### Vector engine, VE

Accelerator associated with vector host. Executes parallel and/or vectorized numeric operations.

### Vector host, VH

Standard x86 server. Performs tasks such as I/O.

### VE offloading

Main operations that take place on the vector host offload parallel and/or vectorized numeric operations to vector engines. In offloading, NEC MPI launches processes on the VH, and those processes then launch other job processes on VEs assigned to the job. See ["Using VE Offloading", on page 210 of the PBS Professional User's Guide](#).

---

## 18.3 Resources for SX-Aurora TSUBASA

### nves

Host-level consumable integer. Allows you to specify the number of vector engines per chunk. PBS sets the available VEs on a vnode in `resources_available.nves`. The default for `resources_available.nves` is number of VEs attached to the PCIe. The out-of-the-box default value for a job request is zero; PBS assigns a value of zero unless the administrator has set the value otherwise.

### nhcas

Chunk-level non-consumable integer. When requested in a job chunk, PBS sets `_NEC_HCA_LIST_IO` and `_NEC_HCA_LIST_MPI` environment variables accordingly for that chunk. When not requested for a chunk, PBS sets `_NEC_HCA_LIST_IO` and `_NEC_HCA_LIST_MPI` to include all HCAs on a host.

### ve\_mem

Job-wide string. Used for reporting the maximum memory on vector engines used by job.

### ve\_cput

Job-wide string. Used for reporting the total CPU time, in seconds, on vector engines used by job.

### ncpus

PBS sets the value of `resources_available.ncpus` on each vnode to  $(\text{\#CPUs on whole host} / \text{\#vnodes on host}) - \text{\#VEs on vnode}$ . One CPU per VE is reserved for the VEOS daemon.

### mem

PBS sets the value of `resources_available.mem` on each vnode by dividing the memory of the whole host equally among the vnodes on the host.

## 18.4 Configuring PBS for NEC SX-Aurora TSUBASA

1. Make sure that the PBS server and MoM daemons are installed and started. See [“Installing via RPM on Linux Systems” on page 24 in the PBS Professional Installation & Upgrade Guide](#).

2. Enable the PBS\_sx\_aurora hook:

```
sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora enabled=True"
```

3. Optional: the default frequency for the hook is 20 seconds. You can set the frequency for the PBS\_sx\_aurora hook in seconds:

```
sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora freq=<frequency>"
```

4. Create the custom resource nves for requesting and managing VEs:

```
sudo /opt/pbs/bin/qmgr -c "c r nves type= long,flag=nhm"
```

5. Add the custom nves resource to the {PBS\_HOME}/sched\_priv/sched\_config resources: line:

```
resources: "ncpus, mem, arch, host, vnode, aoe, eoe, nves"
```

6. HUP the scheduler(s):

```
kill -HUP <scheduler PID>
```

7. Create the custom resource nhcas for managing HCAs:

```
sudo /opt/pbs/bin/qmgr -c "c r nhcas type= long,flag=hm"
```

8. Create the custom resources ve\_cput and ve\_mem for accounting:

```
sudo /opt/pbs/bin/qmgr -c "c r ve_cput type=long, flag=h"
```

```
sudo /opt/pbs/bin/qmgr -c "c r ve_mem type=size, flag=h"
```

9. Recommended: sort your vnodes first by nves, then by ncpus. Replace the default node\_sort\_key in {PBS\_HOME}/sched\_priv/sched\_config with the following:

```
node_sort_key: "nves HIGH unused"
```

```
node_sort_key: "ncpus HIGH unused"
```

10. Set NEC-specific environment variables for all hosts. Set the following in each host's {PBS\_HOME}/pbs\_environment file:

```
NMPI_LAUNCHER_EXEC={PBS_EXEC}/bin/pbs_tmrsh
```

```
NMPI_TTY_COMPAT=ON
```

11. For NEC MPI version 2.21.0 or later, install the execjob\_epilogue hook named "run\_epilogue\_necmpi" to clean up the memory on VE nodes that were used by a job, after the job finishes. Otherwise the memory could, very rarely, remain unreleased:

```
qmgr -c "create hook run_epilogue_necmpi event=execjob_epilogue"
```

```
qmgr -c "set hook run_epilogue_necmpi enabled=True"
```

```
qmgr -c "set hook run_epilogue_necmpi user=pbsuser"
```

```
qmgr -c "set hook run_epilogue_necmpi order=999"
```

```
qmgr -c "import hook run_epilogue_necmpi application/x-python default
/opt/nec/ve/mpi/libexec/memrelease.py"
```

12. Restart PBS so that the PBS\_sx\_aurora hook can create vnodes on all hosts in the cluster:

```
sudo systemctl restart pbs
```

---

## 18.5 Debugging on NEC SX-Aurora TSUBASA

- To see all the DEBUG and INFO messages logged by the hook, increase the log level by setting the `$logevent` parameter to `4095` (0xffffffff) in the MoM's configuration file, `PBS_HOME/mom_priv/config`.
- If you need to detect failed devices faster, you can change the hook's frequency. The frequency is specified in seconds. The default hook frequency is 20 seconds. You can set the frequency for the `PBS_sx_aurora` hook:  

```
sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora freq=<frequency>"
```

## 18.6 Suspending and Resuming Jobs

On the SX-Aurora TSUBASA, partial process swapping (PPS) means copying part of the memory on VEs being used by VE processes onto memory on the VH (the PPS buffer), then later copying the PPS buffer on the VH back to memory on the VEs.

PPS allows high-priority jobs to run before current lower-priority VE processes finish, by suspending and swapping out those lower-priority processes, then resuming the swapped-out VE processes after the high-priority jobs finish.

Swapping out a VE process means copying part of the memory area being used by the VE process onto the PPS buffer, and freeing the memory area so that other, higher-priority, VE processes can use it.

Swapping in a VE process means copying the memory area in the PPS buffer used for the swapped-out VE process back to memory on the VE.

When PBS runs a higher-priority job by swapping out a lower-priority job, it does the following:

1. Suspend the execution of VE processes and VH processes belonging to the lower priority job
2. Swap out the VE processes of the lower-priority job, using SX-Aurora TSUBASA interfaces
3. Run the higher-priority job
4. After completion of the higher-priority job, swap the VE processes of the lower-priority job back
5. Resume the execution of the lower-priority job

## 18.7 Job Accounting on NEC SX-Aurora TSUBASA

When PBS writes accounting records, PBS records `nves` in both `Resource_List.nves` and `resources_assigned.nves`. PBS also writes `ve_mem` and `ve_cput` as part of the value of the job's `resources_used` attribute.

# Mixed Linux-Windows Operation

## 19.1 Introduction to Mixed Linux-Windows Operation

You can add Windows execution and client hosts to a Linux PBS complex, creating a *mixed-mode complex*. These Windows hosts must be in an Active Directory domain. Linux systems must use MUNGE rather than reserved-port authentication, and Windows users must be active directory users. Communication should be encrypted using TLS for improved security. The server needs to authenticate both Linux and Windows users. We describe how to set up a mixed-mode complex in this section.

On Windows, MoM automatically sets `resources_available.arch` to "windows" for the local vnode. Users submitting Windows jobs must request Windows hosts by specifying "windows" for the `arch` resource. For example:

```
qsub -lselect=1:arch=windows ...
```

Users submitting Windows jobs must cache their passwords at each execution and client host before submitting jobs, and each time their password changes. Job submitters use the [pbs\\_login](#) command to cache their passwords.

### 19.1.1 Caveats for Mixed Linux-Windows Operation

- You cannot submit a Linux job from a Windows client
- Group limits are not enforced for Windows jobs; for example, "set queue max\_queued\_res.ncpus = [g:<group name> = <limit>]" has no effect

## 19.2 Configuration

1. Start with a normal working Linux PBS complex. See the *PBS Professional Installation & Upgrade Guide*.

### 19.2.1 Configure Authentication

1. Configure MUNGE authentication for Linux clients and pwd for Windows clients.

The default reserved-port (resvport) method is not secure for mixed-mode operation, because Windows does not have a concept of reserved ports. Follow the instructions in [section 11.4, "Authentication for Daemons & Users", on page 508](#). After you have integrated MUNGE, put this in the server's `/etc/pbs.conf` file:

```
PBS_SUPPORTED_AUTH_METHODS=munge, pwd
```

2. Restart the PBS daemons. On each Linux host:

```
systemctl restart pbs
```

or

```
<path to start/stop script> pbs restart
```

3. Make sure that you can submit jobs and that hooks work.

## 19.2.2 Windows Hosts and Users in Active Directory Domain

1. Make sure the new Windows execution and client hosts are part of the same Windows Active Directory domain.
2. Make sure that Active Directory Authentication works: verify that the users added to the AD domain can log in to all the Windows hosts.

## 19.2.3 Allow Linux Authentication of Windows Active Domain Users

You can use various methods to allow Linux hosts to authenticate Windows Active Domain users. We show an example using SSSD here.

1. On the Linux host running the server, and any hosts running extra comms, configure `sssd` so that the users of the Windows domain can log in to the Linux host on which `pbs_server` and `sssd` run. For an example, see [section 11.4.5, “Configuring SSSD”, on page 510](#). For information on configuring `sssd`, see [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html-single/windows\\_integration\\_guide/index#sssd-ad-proc](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/windows_integration_guide/index#sssd-ad-proc) and <https://access.redhat.com/articles/3023951>.

If you want the Linux host to automatically create a home directory for an Active Directory user if that home directory does not exist at login, you may have to set SELinux to permissive mode. This is optional.

2. Verify that `sssd` is correctly configured.
  - a. Run the following commands:

```
id <username>
su - <username>
<password>
```
  - b. As a Windows domain user, `ssh` to the Linux host running `sssd`

## 19.2.4 Configure User Authorization

We recommend setting `flatuid` to *False* for the PBS complex, so that users need a `.rhosts` file to enable authorization. For example, to configure the `.rhosts` file so that user `User1` can submit jobs from submission host `Winclient1`, make sure that there is a file named `.rhosts` in `User1`'s home directory on the server host, and that this file contains the following entry:

```
Winclient1 User1
```

## 19.2.5 Install PBS on Windows Hosts

1. Install MoMs on your Windows execution hosts, and install the PBS client commands on your Windows client hosts. See [section 3.7, “Installing PBS on Windows Hosts”, on page 38](#).
2. Configure the remote file copy mechanism to be used by Windows execution hosts:
  - If you will use `scp` for your remote file copy mechanism, configure passwordless `ssh`; see [section 9.7.10.1, “Enabling Passwordless Authentication”, on page 448](#).
  - If you will use the `$usecp` MoM parameter to specify your remote file copy mechanism, you do not need to configure passwordless `ssh`, unless it is required by the MPI implementation you are using.
3. Create the parent vnode for each Windows host; see [section 3.3.3, “Creating the Parent Vnode”, on page 44](#):

```
qmgr -c "create node <name of parent vnode>"
```

## 19.2.6 Set Up TLS Encryption

Configure TLS encryption for daemon-daemon communication. For Windows authentication to work securely, we strongly recommend using TLS encryption in the complex. To set up TLS encryption, we need a CA certificate and TLS certificate key pair generated from any system with `openssl` set up. We will use the same key pair on all the server and comm hosts. For an example of how to configure PBS for TLS encryption, see [section 11.5.2.2, “Example of Configuring PBS for TLS Encryption”, on page 518](#).

## 19.3 Troubleshooting Mixed Linux-Windows Complex

- Job comment contains "failed to Impersonate Logged On User on <hostname>;job has bad password"  
The user might not be allowed to log on locally according to the local or global policy settings; MoM is unable to impersonate the job submitter in order to run the job. Check the Windows policy settings; make sure the user is allowed to log in to the execution host.
- Failed to send auth request  
auth: error returned: 15029  
auth: Failed to send auth request  
No support for requested service.  
cannot connect to server pbsserver3 (errno=15029)  
Check whether the server's `PBS_SUPPORTED_AUTH_METHOD` parameter includes `pwd`; add it if not. Restart the server and try again.
- User not known to the underlying authentication module  
auth: error returned: 15019  
auth: PAM authentication failed for testuser2 with error: User not known to the underlying authentication module



When this error occurs despite testuser2 existing and giving the correct password, it may be due to SSSD settings. One reason could be that 'use\_fully\_qualified\_domain' is *True* in the SSSD settings. Change that to *False* and verify that the user can log in from the server using the following commands, in our example:

```
id username:
[pbsadmin@pbsserver3 ~]$ id testuser4
uid=775213102(testuser4) gid=775200513(domain users) groups=775200513(domain users)
su - username:
[pbsadmin@pbsserver3 ~]$ su - testuser2
Password:
Last login: Mon Apr 20 13:40:32 UTC 2022 on pts/1
[testuser2@pbsserver3 ~]$
```

- Windows MoM fails to register

After installing pbs\_mom on Windows and executing win\_postinstall.py, even though the postinstall script completes successfully, the server still shows state of the new Windows vnode as "state-unknown, down". In the comm logs there are repeated messages of authentication failure for the service account.

Check for and delete a stale password file named ".pbs\_cred.CR" from the home directory of the PBS service account used to run the Windows MoM.

- Vnode does not go to *free* state, and the following error message appears in the server logs

```
init_pam;libpam.so not found
validate_auth_data;Failed to initialize the library
tcp_pre_process;Failed to initialize the library
wait_request;process socket failed
```

Check whether the pam library is installed. If it is installed, make sure that the PATH variable points to the location where the library is installed. The library name might be libpam.so.\*. In this case, create a soft link to the library as shown in the example below:

```
ln -s /usr/lib64/libpam.so.0.83.1 /usr/lib64/libpam.so
```

- Files fail to stage out using scp.exe from C:\Windows\System32\OpenSSH

With OpenSSH (scp.exe) installed in the C:\Windows\System32 folder, stage out failures are observed as shown:

```
12/27/2022 00:25:58;0100;PBS_stage_file;Job;11.mixlin;User pbsuser passworded
12/27/2022 00:25:58;0080;PBS_stage_file;Job;sys_copy;CreateProcessAsUser(928,
C:/Windows/System32/OpenSSH-Win64/scp.exe -Brv \PROGRA~2\PBS\home\spool\11.mixlin.OU
"pbsuser@mixlin:/home/pbsuser/STDIN.o11") under acct pbsuser
wdir=C:\Users\pbsuser\Documents\PBS Pro}}
12/27/2022 00:25:58;0080;PBS_stage_file;Fil;sys_copy;command:
C:/Windows/System32/OpenSSH-Win64/scp.exe -Brv C:/PROGRA~2/PBS/home/spool/11.mixlin.OU
pbsuser@mixlin:/home/pbsuser/STDIN.o11 status=10002, try=1
```

This is caused by Windows WOW64 filesystem redirection, since PBS is a 32-bit application. See Microsoft article <https://docs.microsoft.com/en-us/windows/win32/winprog64/file-system-redirector>.

Replace "Sysetm32" with "Sysnative" in the PBS\_SCP parameter in pbs.conf and restart the PBS\_MOM service. You do not need to move or reinstall OpenSSH from C:\Windows\System32 path.

# 20

# Problem Solving

Additional information is always available online at the PBS website, [www.pbsworks.com](http://www.pbsworks.com). The last section in this chapter tells you how to get additional assistance from the PBS Support staff.

## 20.1 Debugging Tools

### 20.1.1 Debugging Commands

The following commands will provide helpful debugging information: `qstat`, `tracejob`, `qmgr`, and `pbsnodes`.

### 20.1.2 Setting Corefile Size

To set the size of the core file for a PBS daemon, you can set `PBS_CORE_LIMIT` in `pbs.conf`. Set this on the machine where the daemon runs. This can be set to an integer number of bytes or to the string "unlimited". If this is unset, the limit is inherited from the shell environment, which you can check via `uname -c`.

### 20.1.3 Using the debuginfo RPM Package

PBS is shipped with debuginfo package(s). When you unzip the PBS product download package the debuginfo package(s) can be found alongside the other PBS packages containing the server etc.

Normally, you do not need to install any debuginfo package(s). You only need to install the debuginfo package(s) when the support team recommends doing so to aid in diagnosing a problem. The contents of the debuginfo package files are automatically installed in the default location for your Linux distribution, typically under `/usr/lib/debug` and `/usr/src/debug`.

The debuginfo package(s) names are platform-dependent. When you install a debuginfo package, nothing additional is installed in `PBS_HOME` or `PBS_EXEC`.

### 20.1.4 Finding PBS Version Information

Use the `qstat` command to find out what version of PBS Professional you have.

```
qstat -fB
```

In addition, each PBS command will print its version information if given the `--version` option. This option cannot be used with other options.

### 20.1.5 Troubleshooting and Hooks

PBS is shipped with tools for debugging hooks. See ["Debugging Hooks", on page 185 of the PBS Professional Hooks Guide](#).

You may wish to disable hook execution in order to debug PBS issues. To verify whether hooks are part of the problem, disable each hook by setting its `enabled` attribute to *False*.

---

## 20.2 Security and Permissions Problems

### 20.2.1 Directory Permission Problems

If for some reason the access permissions on the PBS file tree are changed from their default settings, a component of the PBS system may detect this as a security violation, and refuse to execute. If this is the case, an error message to this effect will be written to the corresponding log file. You can run the `pbs_probe` command to check (and optionally correct) any directory permission (or ownership) problems. See [“pbs\\_probe” on page 80 of the PBS Professional Reference Guide](#) for details on usage of the `pbs_probe` command.

#### 20.2.1.1 Correcting Permissions Problems on Linux

You can use the `pbs_probe` command to detect and repair file and directory permissions problems. You can run `pbs_probe` in report mode or fix mode; in report mode, it reports the errors found; in fix mode, it attempts to fix detected problems, and reports any problems it could not fix.

To fix permissions errors, log into the host you wish to check, and run the following command:

```
pbs_probe -f
```

See the `pbs_probe(8B)` manual page.

#### 20.2.1.2 Correcting Permissions Problems on Windows

You can use the `pbs_mkdirs` command to correct file and directory permissions problems on Windows. The command checks and if necessary repairs the permissions of configuration files such as `pbs_environment` and `mom_priv/config`. You should run the `pbs_mkdirs` command only while the PBS MoMs are stopped.

To repair permissions on an execution host, log into the host and run the following commands:

```
net stop pbs_mom
pbs_mkdirs mom
net start pbs_mom
```

## 20.3 Troubleshooting Jobs

### 20.3.1 Job Held Due to Invalid Password

If a job fails to run due to an invalid password, then the job is held with hold type `p` (bad password), its comment field updated with why it failed, and an email is sent to the owner for remedy action. Root or administrator can release the hold via [qrls](#). See [“qhold” on page 150 of the PBS Professional Reference Guide](#) and [“qrls” on page 183 of the PBS Professional Reference Guide](#).

## 20.3.2 Requeueing a Job "Stuck" on a Down Vnode

PBS Professional will detect if a vnode fails when a job is running on it, and will automatically requeue and schedule the job to run elsewhere. If the user marked the job as "not rerunnable" (i.e. via the `qsub -r n` option), then the job will be deleted rather than requeued. If the affected vnode is on the primary execution host, the requeue will occur quickly. If it is another vnode in the set assigned to the job, it could take a few minutes before PBS takes action to requeue or delete the job. However, if the auto-requeue feature is not enabled, or if you wish to act immediately, you can manually force the requeueing and/or rerunning of the job. See [section 8.6.2, "Node Fail Requeue: Jobs on Failed Vnodes", on page 412](#).

If you wish to have PBS simply remove the job from the system, use the `-Wforce` option to `qdel` :

```
qdel -Wforce <job ID>
```

If instead you want PBS to requeue the job, and have it immediately eligible to run again, use the `-Wforce` option to `qrerun`

```
qrerun -Wforce <job ID>
```

See ["Job Input & Output Files", on page 33 of the PBS Professional User's Guide](#).

## 20.3.3 Job Cannot be Executed

If a user receives a mail message containing a job ID and the line "Job cannot be executed", the job was aborted by MoM when she tried to place it into execution. The complete reason can be found in one of two places, MoM's log file or the standard error file of the user's job. If the second line of the message is "See Administrator for help", then MoM aborted the job before the job's files were set up. The reason will be noted in MoM's log. Typical reasons are a bad user/group account, checkpoint/restart file, or a system error. If the second line of the message is "See job standard error file", then MoM had created the job's file and additional messages were written to standard error. This is typically the result of a bad resource request.

## 20.3.4 Running Jobs with No Active Processes

On very rare occasions, PBS may be in a situation where a job is in the *Running* state but has no active processes. This should never happen as the death of the job's shell should trigger MoM to notify the server that the job exited and end-of-job processing should begin. If this situation is noted, PBS offers a way out. Use the `qsig` command to send `SIGNULL`, signal 0, to the job. If MoM finds there are no processes then she will force the job into the exiting state. See ["qsig" on page 195 of the PBS Professional Reference Guide](#).

## 20.3.5 Jobs that Can Never Run

If backfilling is being used, the scheduler looks at the job being backfilled around and determines whether that job can never run.

If backfilling is being used, the scheduler determines whether that job can or cannot run now, and if it can't run now, whether it can ever run. If the job can never run, the scheduler logs a message saying so.

The scheduler only considers the job being backfilled around. That is the only job for which it will log a message saying the job can never run.

This means that a job that can never run will sit in the queue until it becomes the most deserving job. Whenever this job is considered for having small jobs backfilled around it, the error message "resource request is impossible to solve: job will never run" is printed in the scheduler's log file. If backfilling is not being used, this message will not appear.

If backfilling is not being used, the scheduler determines only whether that job can or cannot run now. The scheduler won't determine if a job will ever run or not.

## 20.3.6 Job Comments for Problem Jobs

PBS can detect when a job cannot run with the current unused resources and when a job will never be able to run with all of the configured resources. PBS can set the job's comment attribute to reflect why the job is not running.

If the job's comment starts with "Can never run", the job will never be able to run with the resources that are currently configured. This can happen when:

- A job requests more of a consumable resource than is available on the entire complex
- A job requests a non-consumable resource that is not available on the complex

For example, if there are 128 total CPUs in the complex, and the job requests 256 CPUs, the job's comment will start with this message.

If the job's comment starts with "Not running", the job cannot run with the resources that are currently available. For example, if a job requests 8 CPUs and the complex has 16 CPUs but 12 are in use, the job's comment will start with this message.

You may see the following comments. R is for "Requested", A is for "Available", and T is for "Total":

```
"Not enough free nodes available"
"Not enough total nodes available"
"Job will never run with the resources currently configured in the complex"
"Insufficient amount of server resource <resource name> (R | A | T | <requested value>
 !=<available values for requested resource>)"
"Insufficient amount of queue resource <resource name> (R | A | T | <requested value> !=<available
 values for requested resource>)"
"Error in calculation of start time of top job"
"Can't find start time estimate"
```

The "Can Never Run" prefix may be seen with the following messages:

```
"Insufficient amount of resource <resource name> (R | A | T | <requested value> !=<available
 values for requested resource>)"
"Insufficient amount of Server resource <resource name> (R | A | T | <requested value>
 !=<available values for requested resource>)"
"Insufficient amount of Queue resource <resource name> (R | A | T | <requested value> !=<available
 values for requested resource>)"
"Not enough total nodes available"
"can't fit in the largest placement set, and can't span psets"
```

## 20.3.7 Bad UID for Job Execution

For a job to be accepted by the PBS server, the user at the submitting host must pass an `ruserok()` test.

From the `RCMD(3)` man page:

The `iruserok()` and `ruserok()` functions take a remote host's IP address or name, respectively, two usernames and a flag indicating whether the local user's name is that of the superuser. Then, if the user is NOT the superuser, it checks the `/etc/hosts.equiv` file. If that lookup is not done, or is unsuccessful, the `.rhosts` in the local user's home directory is checked to see if the request for service is allowed.

If this file does not exist, is not a regular file, is owned by anyone other than the user or the superuser, or is writable by anyone other than the owner, the check automatically fails. Zero is returned if the machine name is listed in the `hosts.equiv` file, or the host and remote username are found in the `.rhosts` file; otherwise `iruserok()` and `ruserok()` return -1. If the local domain (as obtained from `gethostname(2)`) is the same as the remote domain, only the machine name need be specified.

If the server attribute `flatuid` is set to true, this test is skipped and the job is accepted based on the submitting users name alone (with obvious security implications).

You can run the following command:

```
Qmgr: set server flatuid=true
```

Flatuid or not, to run as a user other than the job owner (the submitter) you must have authorization to do so. Otherwise, any user could run a job as any other user. You authorize for userA to run a job as userB the same way you authorize userA@host1 to run a job as userA on host2 when flatuid is Not SET, i.e. see `.ruserok()` and `.rhosts`.

Here is a test program to see if `ruserok` passes for a given user and host. There are two use cases:

- User submitting job from remote host to server getting unexpected "Bad UID" message. That is, user doesn't have access when he thinks he should.
- User(s) can delete, etc other user(s) jobs. That is, one user is able to act as what he thinks is a different user, server sees them as being equivalent.

Build this with "cc ruserok.c -o ruserok"

Usage (run on the PBS server system):

```
ruserok remote_host remote_user1 local_user2
```

where:

*remote\_host*: the host from which the job is being submitted, or where the PBS client command is issued

*remote\_user1*: the username of the user submitting the job, or issuing the client command

*local\_user2*: the username of the user *remote\_user1* is trying to submit the job as, or owner of the job that *remote\_user1* is trying to act on with the client command

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
 int rc;
 char hn[257];
 if (argc != 4)
 { fprintf(stderr, "Usage: %s remote_host remote_user1 local_user2\n", argv[0]); return 1;
 }
 if (gethostname(hn, 256) < 0)
 { perror("unable to get hostname"); return 2; }
 hn[256] = '\0';
 printf("on local host %s, from remote host %s\n", hn, argv[1]);
 rc = ruserok(argv[1], 0, argv[2], argv[3]);
 if (rc == 0)
 printf("remote user %s is allowed access as local user %s\n", argv[2], argv[3]);
 else
 printf("remote user %s is denied access as local user %s\n", argv[2], argv[3]);
 return 0;
}
```

---

## 20.3.8 Windows: Bad UID for Job Execution

If, when attempting to submit a job to a remote server, `qsub` reports:

```
BAD uid for job execution
```

Then you need to add an entry in the remote system's `.rhosts` or `hosts.equiv` pointing to your Windows machine. Be sure to put in all hostnames that resolve to your machine. See [section 2.3.6, “User Authorization Under Windows”, on page 15](#).

If remote account maps to an Administrator-type account, then you need to set up a `.rhosts` entry, and the remote server must carry the account on its `acl_roots` list.

## 20.3.9 New Jobs Not Running

If PBS loses contact with the Altair License Server, any jobs currently running will not be interrupted or killed. The PBS server will continually attempt to reconnect to the license server, and re-license the assigned vnodes once the contact to the license server is restored.

No new jobs will run if PBS server loses contact with the ALM license server.

## 20.3.10 Job Stuck in Exiting State

A job can be stuck in the Exiting state if the user submits the job from a directory where the user does not have write access. You can forcefully delete the job:

### 20.3.10.1 `qdel -Wforce <job ID>`

## 20.4 Troubleshooting Daemons

### 20.4.1 Server Host Bogs Down After Startup

If the server host becomes unresponsive a short time after startup, the server may be trying to contact the wrong license server.

#### 20.4.1.1 Symptoms

15 seconds to one or two minutes after you start the PBS server, the system becomes unresponsive.

#### 20.4.1.2 Problem

The problem may be caused by the `pbs_license_info` server attribute pointing to an old FLEX license server. This attribute should point to the new ALM license server. See the *PBS Works Licensing Guide*.

#### 20.4.1.3 Treatment

On some Linux systems, the effects of memory starvation on subsequent responsiveness may be long-lasting. Therefore, instead of merely killing and restarting the PBS server, we recommend rebooting the machine.



Take the following steps:

1. Reboot the machine into single-user mode.
2. Determine the correct value for `pbs_license_info` and set the `PBS_LICENSE_INFO` entry in `pbs.conf` to this value.
3. Reboot, or change `runlevel` to multi-user.
4. Using `qmgr`, set the `pbs_license_info` server attribute to the correct value:  

```
qmgr -c "set server pbs_license_info = <port>@<license server hostname>"
qmgr -c "set server scheduling= true"
```
5. Stop the PBS server process.
6. Continue normally.

## 20.4.2 Server Does Not Start

The server may not start due to problems with the data service. For more on the PBS data service, see [“pbs\\_datservice” on page 61 of the PBS Professional Reference Guide](#).

## 20.4.3 Primary Server Periodically Restarting

If the primary server keeps restarting, an unknown secondary server may be contacting it. This can happen when `PBS_PRIMARY` and `PBS_SECONDARY` are missing from `pbs.conf`, but a secondary server has been started.

## 20.4.4 PBS Data Service Does Not Start

- You may need to create the data service management account. This must be creating before installing PBS. See [“Create PBS Data Service Management Account” on page 24 in the PBS Professional Installation & Upgrade Guide](#).
- If you see an error message saying "PBS data service is running on another host - cannot start", there may be a problem with the lock file in `$PBS_HOME/dataservice/pbs_dblock`:
  - Problem during failover between two hosts, where the primary host still has a lock on the file
  - Ungraceful shutdown, where the primary host has an incorrectly, still-locked, lock file; look at the primary server host.
  - File system issues that interfere with the locking, unlocking, and/or access, of the lock file.

## 20.4.5 Server Dies Inexplicably

Check the data service. When the data service dies, the server automatically goes down too.

## 20.4.6 Data Service Running When PBS Server is Down

You can use the `pbs_datservice` command to stop the data service. See [“pbs\\_datservice” on page 61 of the PBS Professional Reference Guide](#).



---

## 20.4.7 Scheduler Cannot Reliably Contact Server

If you see a series of 15031 errors, this can happen when PBS\_PRIMARY and PBS\_SECONDARY are missing from pbs.conf, but a secondary server has been started.

## 20.4.8 PBS Daemon Will Not Start

If the PBS server, MoM, or scheduler fails to start up, it may be refusing to start because it has detected permissions problems in its directories or on one or more of its configuration files, such as pbs\_environment or mom\_priv/config.

## 20.4.9 Troubleshooting Windows Daemon Problems

### 20.4.9.1 Windows: MoMs Do Not Start

- In the case where the PBS daemons, the Active Directory database, and the domain controller are all on the same host, some PBS MoMs may not start up immediately. If the Active Directory services are not running when the PBS MoMs are started, the MoMs won't be able to talk to the domain controller. This can prevent the PBS MoMs from starting. As a workaround, wait until the host is completely up, then retry starting the failing MoM.

Example:

```
net start pbs_mom
```

- In a domained environment, if the PBS service account is a member of any group besides "Domain Users", the install program will fail to add the PBS service account to the local Administrators group on the install host. Make sure that the PBS service account is a member of only one group, "Domain Users" in a domained environment.
- If the MoM fails to start up because of permission problems on some of its configuration files like pbs\_environment or mom\_priv/config, then correct the permission by running:  

```
pbs_mkdirs mom
```

## 20.5 Troubleshooting Vnodes

### 20.5.1 Vnodes Down

The PBS server determines the state of hosts (*up* or *down*), by communicating with MoM on the host. The state of vnodes may be listed by two commands: qmgr and pbsnodes

```
Qmgr: list node @active
```

```
pbsnodes -a
```

```
Node jupiter state = state-unknown, down
```

A vnode in PBS may be marked "*down*" in one of two substates. For example, the state above of vnode "jupiter" shows that the server has not had contact with MoM since the server came up. Check to see if a MoM is running on the vnode. If there is a MoM and if the MoM was just started, the server may have attempted to poll her before she was up. The server should see her during the next polling cycle in 10 minutes. If the vnode is still marked "state-unknown, down" after 10+ minutes, either the vnode name specified in the server's node file does not map to the real network hostname or there is a network problem between the server host and the vnode.

If the vnode is listed as:

```
pbsnodes -a
```

```
Node jupiter state = down
```

then the server has been able to ping MoM on the vnode in the past, but she has not responded recently. The server will send a "ping" PBS message to every free vnode each ping cycle, 10 minutes. If a vnode does not acknowledge the ping before the next cycle, the server will mark the vnode down.

## 20.5.2 Bad Vnode on Startup

If, when the server starts up, one or more vnodes cannot be resolved, the server marks the bad vnode(s) in state *"state-unknown, down"*.

## 20.5.3 Oversubscribed Resources

Vnode resources such as ncpus may be oversubscribed when using the aggressive\_provision provisioning policy. In this case, switch to the avoid\_provision policy:

1. Back up your scheduler configuration by copying sched\_config to sched\_config.backup
2. Edit sched\_config, and change the provision\_policy parameter from  
provision\_policy: "aggressive\_provision"  
to  
provision\_policy: "avoid\_provision"
3. Restart the scheduler:  
**kill -HUP <scheduler PID>**

See [section 16.4.2.1, "Provisioning Policy", on page 593](#).

# 20.6 Troubleshooting Client Commands

## 20.6.1 Windows: Client Commands Slow

PBS caches the IP address of the local host, and uses this to communicate between the daemons. If the cached IP address is invalidated, PBS can become slow. In both scenarios, jobs must be killed and restarted.

### 20.6.1.1 Scenario: Wireless Router, DHCP Enabled

The system is connected to a wireless router that has DHCP enabled. DHCP returned a new IP address for the server short name, but DNS is resolving the server full name to a different IP address.

The IP address and server full name have become invalid due to the new DHCP address. PBS has cached the IP address of the server full name.

Therefore, the PBS server times out when trying to connect to the scheduler and local MoM using the previously cached IP address. This makes PBS slow.

Symptom:

1. PBS is slow.
  - a. Server logs show "Could not contact scheduler".
  - b. `pbsnodes -a` shows that the local node is down.
2. First IP addresses returned below don't match:

```
cmd.admin> pbs_hostn -v <server_short_name>
cmd.admin> pbs_hostn -v <server_full_name>
```

Workaround: cache the correct new IP address of the local server host.

1. Add the address returned by `pbs_hostn -v <server_short_name>` (normally the DHCP address) to `%WINDIR%\system32\drivers\etc\hosts` file as follows:  
`<DHCP address> <server_full_name> <server_short_name>`
2. Restart the MoM:
 

```
cmd.admin> net stop pbs_mom
cmd.admin> net start pbs_mom
```

## 20.6.2 Windows: qstat Errors

If the `qstat` command produces an error such as:

```
illegally formed job identifier.
```

This means that the DNS lookup is not working properly, or reverse lookup is failing. Use the following command to verify DNS reverse lookup is working

```
pbs_hostn -v hostname
```

If however, `qstat` reports "No Permission", then check `pbs.conf`, and look for the entry "PBS\_EXEC". `qstat` (in fact all the PBS commands) will execute the command "PBS\_EXEC/sbin/pbs\_iff" to do its authentication. Ensure that the path specified in `pbs.conf` is correct.

## 20.6.3 Clients Unable to Contact Server

If a client command (such as `qstat` or `qmgr`) is unable to connect to a server there are several possibilities to check. If the error return is 15034, "No server to connect to", check (1) that there is indeed a server running and (2) that the default server information is set correctly. The client commands will attempt to connect to the server specified on the command line if given, or if not given, the server specified by `SERVER_NAME` in `pbs.conf`.

If the error return is 15007, "No permission", check for (2) as above. Also check that the executable `pbs_iff` is located in the search path for the client and that it is setuid root. Additionally, try running `pbs_iff` by typing:

```
pbs_iff -t server_host 15001
```

Where `server_host` is the name of the host on which the server is running and 15001 is the port to which the server is listening (if started with a different port number, use that number instead of 15001). Check for an error message and/or a non-zero exit status. If `pbs_iff` exits with a non-zero status, either the server is not running or was installed with a different encryption system than was `pbs_iff`.

## 20.7 Troubleshooting PBS Licenses

### 20.7.1 Wrong License Server: Out of Memory

If you run out of memory shortly after startup, the server may be looking for the wrong license server. See [section 20.4.1, “Server Host Bogs Down After Startup”, on page 640](#).

### 20.7.2 Unable to Connect to License Server

If PBS cannot contact the license server, the server will log a message:

```
"Unable to connect to license server at pbs_license_info=..."
```

If the license server location is incorrectly initialized (e.g. if the host name or port number is incorrect), PBS may not be able to pinpoint the misconfiguration as the cause of the failure to reach a license server.

If PBS cannot detect a license server host and port when it starts up, the server logs an error message:

```
"Did not find a license server host and port (pbs_license_info=<X>). No external license server
will be contacted"
```

### 20.7.3 Insufficient Minimum Licenses

If the PBS server cannot get the number of licenses specified in `pbs_license_min` from the license server, the server will log a message:

```
"checked-out only <X> CPU licenses instead of pbs_license_min=<Y> from license server at host <H>,
port <P>. Will try to get more later."
```

### 20.7.4 Wrong Type of License

If the PBS server encounters a proprietary license key that is of the wrong type, the server will log the following message:

```
"license key #1 is invalid: invalid type or version".
```

## 20.8 Crash Recovery

PBS daemons could terminate unexpectedly either because the host machine stops running or because the daemon itself stops running. The daemon may be killed by mistake, or may (rarely) crash. The server may terminate if the filesystem runs out of space.

### 20.8.1 Recovery When Host Machine Stops

If the host machine stops running, no special steps are required, since PBS will be started when the machine starts.

#### 20.8.1.1 Execution Host Stops

If the host machine is an execution host, any jobs that were running on that host were terminated when the machine stopped, and when MoM is restarted, she will report to the server that those jobs are dead, and begin normal activity. The server will automatically restart any jobs that can be restarted.

Shutting down one host of a multi-host job will cause that job to be killed. The job will have to be rerun; restarting the MoM on the stopped host with the -p option will not help the job. See [“pbs\\_mom” on page 71 of the PBS Professional Reference Guide](#).

### 20.8.1.2 Server/scheduler/communication Host Stops

If the host machine is the server/scheduler/communication host, no data is lost and no jobs are lost, because the server writes everything to disk. The server is restarted automatically upon machine startup.

The scheduler is started automatically upon machine startup. The scheduler starts fresh each cycle, so it does not lose data.

## 20.8.2 Recovery When Daemon Stops

For more detailed information on starting and stopping PBS, see [“Starting & Stopping PBS on Linux” on page 143 in the PBS Professional Installation & Upgrade Guide](#).

## 20.9 Other Troubleshooting

### 20.9.1 Problem With Dynamic Resource

If you need to debug a dynamic resource being supplied by an external script, it may help to follow these steps:

1. Set the scheduler's `log_events` parameter to 4095 (everything is logged)  
`qmgr -c "set sched <scheduler name> log_events = 4095"`
2. Send a SIGHUP to the scheduler (`pbs_sched`)
3. The scheduler log will contain the value the scheduler reads from the external script

### 20.9.2 Cannot Create Formula or Hook

You must run `qmgr` at the server host when operating on the server's `job_sort_formula` attribute or on hooks. For example, attempting to create the formula at another host will result in the following error:

```
qmgr obj= svr=default: Unauthorized Request job_sort_formula
```

### 20.9.3 Windows: PBS Cannot Locate Configuration File

If PBS is installed on a hard drive other than `C:`, it may not be able to locate the `pbs.conf` global configuration file. If this is the case, PBS will report the following message:

```
E:\Program Files\PBS\exec\bin>qstat -
pbsconf error: pbs conf variables not found:
PBS_HOME PBS_EXEC
No such file or directory
qstat: cannot connect to server UNKNOWN (errno=0)
```

To correct this problem, set `PBS_CONF_FILE` to point `pbs.conf` to the right path. Normally, during PBS Windows installation, this would be set in `system autoexec.bat` which will be read after the Windows system has been restarted. Thus, after PBS Windows installation completes, be sure to reboot the Windows system in order for this variable to be read correctly.

---

## 20.9.4 Filesystem Runs Out of Space

If your filesystem has run out of space, the server may experience errors or may crash. If the server is still running, you need only to free up enough space. If the server has crashed, you must restart it. See [“Server: Starting, Stopping, Restarting” on page 147 in the PBS Professional Installation & Upgrade Guide](#).

## 20.9.5 Unrecognized Timezone Variable

Problem: you see this message:

```
pbs_rsub: Bad time specification(s)
```

Reason: The time zone is not specified correctly in `PBS_TZID`. On later Linux updates, the system's zoneinfo files may have some countries represented under different names from those in previous releases. For example, *Asia/Calcutta* has been replaced by *Asia/Kolkata*.

In order to create reservations, the PBS server must recognize the `PBS_TZID` environment variable at the submission host. The appropriate zone location for the submission host can be obtained from the machine on which the PBS Professional server is installed.

- On Linux platforms, either use the `tzselect` command, if it is available, or look in the underlying operating system's `zone.tab` timezone location file, which may be found under `/usr/share/zoneinfo/zone.tab`. While the PBS server is running and can contact the execution machine, use the Linux `tzselect` utility to determine the value for `PBS_TZID`.
- On all other platforms, look in the list of libical supported zoneinfo locations available under `$PBS_EXEC/lib/ical/zoneinfo/zones.tab`.

## 20.10 Getting Help

If the material in the PBS manuals is unable to help you solve a particular problem, you may need to contact the PBS Support Team for assistance. The PBS Professional support team can be reached directly via email and phone; contact information is on the inside front cover of each manual.



# Index

\$logevent MoM parameter [AG-431](#)  
\$restrict\_user [AG-521](#)  
\$restrict\_user\_exceptions [AG-521](#)  
\$restrict\_user\_maxsysid [AG-521](#)  
.rhosts [AG-507](#)

## A

### access

by group [AG-492](#)  
by user [AG-492](#)  
    effect of flatuid [AG-506](#)  
control lists [AG-492](#)  
from host [AG-492](#)  
to a queue [AG-492](#)  
to a reservation [AG-492](#)  
to server [AG-492](#)

### accounting

account [AG-534](#), [AG-538](#), [AG-539](#)  
alt\_id [AG-534](#), [AG-539](#)  
authorized\_hosts [AG-532](#)  
authorized\_users [AG-532](#)  
ctime [AG-532](#), [AG-533](#), [AG-534](#), [AG-536](#), [AG-539](#),  
    [AG-541](#), [AG-542](#)  
duration [AG-532](#)  
end [AG-532](#), [AG-534](#), [AG-540](#)  
etime [AG-533](#), [AG-534](#), [AG-536](#), [AG-539](#), [AG-540](#),  
    [AG-541](#), [AG-542](#)  
exec\_host [AG-535](#)  
exec\_vnode [AG-535](#)  
Exit\_status [AG-535](#), [AG-540](#)  
group [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#), [AG-540](#),  
    [AG-541](#), [AG-543](#)  
jobname [AG-533](#), [AG-535](#), [AG-539](#), [AG-540](#),  
    [AG-541](#), [AG-543](#)  
jobobit [AG-535](#), [AG-540](#)  
name [AG-532](#)  
owner [AG-532](#)  
qtime [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#), [AG-540](#),  
    [AG-541](#), [AG-543](#)  
queue [AG-532](#), [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#),  
    [AG-540](#), [AG-541](#), [AG-543](#)  
Resource\_List [AG-532](#), [AG-533](#), [AG-535](#), [AG-536](#),  
    [AG-539](#), [AG-540](#), [AG-541](#), [AG-543](#)  
session [AG-533](#), [AG-535](#), [AG-536](#), [AG-540](#),  
    [AG-542](#), [AG-543](#)  
start [AG-532](#), [AG-533](#), [AG-535](#), [AG-536](#), [AG-540](#),

[AG-542](#), [AG-543](#)  
    user [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#), [AG-540](#),  
        [AG-542](#), [AG-543](#)  
accounting\_id [AG-534](#), [AG-538](#)  
acl\_group\_enable  
    queue attribute [AG-500](#)  
acl\_groups  
    queue attribute [AG-500](#)  
acl\_host\_enable  
    queue attribute [AG-500](#)  
    server attribute [AG-500](#)  
acl\_hosts  
    queue attribute [AG-500](#)  
    server attribute [AG-500](#)  
acl\_roots [AG-524](#)  
acl\_user\_enable  
    queue attribute [AG-500](#)  
    server attribute [AG-500](#)  
acl\_users  
    queue attribute [AG-500](#)  
    server attribute [AG-500](#)  
ACLs [AG-492](#)  
    default behavior [AG-493](#)  
    format [AG-493](#)  
    group [AG-494](#)  
    host [AG-494](#)  
    matching entry [AG-495](#)  
    modifying behavior [AG-493](#)  
    overrides [AG-506](#)  
    removing entity [AG-497](#)  
    rules for creating [AG-497](#)  
    user [AG-494](#)  
    who can create [AG-498](#)  
activate a power profile [AG-586](#)  
advance reservation [AG-197](#), [AG-532](#)  
aggressive\_provision [AG-593](#)  
AOE [AG-591](#)  
aoe resource  
    defining [AG-600](#)  
application license  
    floating [AG-272](#)  
        definition [AG-229](#)  
    floating externally-managed [AG-272](#)



## Index

---

- application licenses [AG-270](#)
  - floating license PBS-managed [AG-273](#)
  - license units and features [AG-271](#)
  - overview [AG-254](#)
  - per-host node-locked example [AG-275](#)
  - types [AG-270](#)
- ASAP reservation [AG-197](#)
- authentication [AG-577](#)
- authorization [AG-577](#)
- Authorized\_Groups reservation attribute [AG-501](#)
- Authorized\_Hosts reservation attribute [AG-501](#)
- Authorized\_Users reservation attribute [AG-501](#)
- average CPU usage enforcement [AG-303](#)
- average\_cpufactor [AG-303](#)
- average\_percent\_over [AG-303](#)
- average\_trialperiod [AG-303](#)
- avoid\_provision [AG-593](#)

### B

- backfill\_prime [AG-194](#)
- basic fairshare [AG-140](#)
- batch requests [AG-431](#)
- Boolean
  - format [AG-234](#)
- borrowing vnode [AG-228](#), [AG-266](#)
- built-in resource [AG-228](#)

### C

- checkpoint [AG-532](#), [AG-637](#)
  - preemption via [AG-187](#)
- chunk [AG-229](#)
- clienthost [AG-521](#)
- configuration
  - file staging [AG-581](#)
  - rsync [AG-581](#)
  - server [AG-21](#)
- consumable resource [AG-229](#)
- CPU [AG-229](#)
- cpuaverage [AG-303](#)
- cput [AG-143](#)
- creating queues [AG-25](#)
- creation of provisioning hooks [AG-602](#)
- current\_aoe [AG-600](#)
- current\_eoe [AG-587](#)
- custom resource [AG-229](#)

- custom resources
  - application licenses [AG-270](#)
    - floating managed by PBS [AG-273](#)
    - overview [AG-254](#)
    - per-host node-locked [AG-275](#)
    - types [AG-270](#)
  - how to use [AG-252](#)
  - scratch space
    - overview [AG-254](#)
  - static host-level [AG-265](#)
  - static server-level [AG-264](#)
- cycle harvesting
  - ideal\_load [AG-126](#)
  - max\_load [AG-126](#)

### D

- deactivate a power profile [AG-586](#)
- debuginfo [AG-635](#)
- decay [AG-143](#)
- dedicated time [AG-128](#)
- defining aoe resource [AG-600](#)
- defining provisioning policy [AG-603](#)
- defining resources
  - multi-vnode machines [AG-268](#)
- degraded reservation [AG-197](#)
- department [AG-141](#)
- DIS [AG-422](#)
- DNS [AG-644](#)
- dynamic fit [AG-169](#)
- dynamic resource [AG-229](#)

### E

- egroup [AG-141](#)
  - euser [AG-141](#)
- eligible wait time [AG-129](#)
- eligible\_time [AG-129](#), [AG-131](#), [AG-534](#), [AG-539](#)
- energy [AG-587](#)
- enforcement [AG-578](#)
- eoe [AG-583](#), [AG-587](#)
- euser [AG-141](#)
- exec\_host [AG-532](#)
- exiting [AG-129](#)
- express\_queue [AG-184](#)

### F

- fair\_share
  - scheduler parameter [AG-140](#)
- fairshare [AG-139](#), [AG-184](#)
- fairshare entities [AG-141](#)
- fairshare ID [AG-141](#)
- fairshare\_perc [AG-154](#)
- file staging
  - configuration [AG-581](#)

## Index

---

### files

- pbs.conf [AG-644](#)
- policy [AG-578](#)
  - location [AG-578](#)

finished jobs [AG-480](#)

flatuid server attribute [AG-506](#)

### float

- format [AG-234](#)

### floating license

- definition [AG-229](#)
- example [AG-272](#)
- example of externally-managed [AG-272](#)

### format

- Boolean [AG-234](#)
- float [AG-234](#)
- size [AG-235](#)
- string resource value [AG-235](#), [AG-240](#)
- string\_array [AG-235](#), [AG-240](#)

## G

gethostname [AG-521](#)

Globus [AG-21](#)

### group

- access [AG-492](#)
- ACLs [AG-494](#)
- limit [AG-229](#), [AG-285](#)
  - generic [AG-285](#)
  - individual [AG-285](#)

## H

HCA [AG-627](#)

help, getting [AG-647](#)

history jobs [AG-480](#)

### hooks

- creation of provisioning [AG-602](#)
- provisioning [AG-591](#)

### host

- access [AG-492](#)
- ACLs [AG-494](#)

host channel adapter [AG-627](#)

hosts.equiv [AG-507](#)

## I

### ideal\_load

- cycle harvesting [AG-126](#)

indirect resource [AG-229](#), [AG-266](#)

ineligible\_time [AG-129](#)

InfiniBand [AG-573](#)

initial\_time [AG-130](#)

instance [AG-197](#)

instance of a standing reservation [AG-197](#)

instantiation [AG-578](#)

## J

job history [AG-480](#)

- changing settings [AG-482](#)

- configuring [AG-481](#)

- enabling [AG-481](#)

- setting duration [AG-481](#)

job that can never run [AG-637](#)

job-specific ASAP reservation [AG-197](#)

job-specific now reservation [AG-197](#)

job-specific reservation [AG-197](#)

job-specific start reservation [AG-197](#)

## L

last\_state\_change\_time

- vnode attribute [AG-587](#)

last\_used\_time

- vnode attribute [AG-587](#)

### license

- application

  - floating [AG-272](#)

- floating

  - definition [AG-229](#)

limit [AG-230](#), [AG-284](#)

- attributes [AG-290](#)

- cput [AG-301](#)

- file size [AG-301](#)

- generic group limit [AG-229](#), [AG-285](#)

- generic project limit [AG-285](#)

- generic user limit [AG-229](#), [AG-285](#)

- group limit [AG-229](#), [AG-285](#)

- individual group limit [AG-229](#), [AG-285](#)

- individual project limit [AG-285](#)

- individual user limit [AG-230](#), [AG-285](#)

- overall [AG-230](#), [AG-285](#)

- pcput [AG-301](#)

- pmem [AG-301](#)

- project limit [AG-285](#)

- pvmem [AG-301](#)

- user limit [AG-230](#), [AG-285](#)

- walltime [AG-301](#)

### limits

- generic and individual [AG-288](#)

- group [AG-283](#)

- overall limits [AG-288](#)

- project [AG-283](#)

- resource usage [AG-283](#)

- scope [AG-286](#)

- setting limits [AG-292](#)

- user [AG-283](#)

### location

- policy files [AG-578](#)

## Index

log events  
    MoM [AG-431](#)  
    scheduler [AG-431](#)  
    server [AG-431](#)  
log levels [AG-430](#)  
log\_events  
    server attribute [AG-431](#)  
logs  
    permissions [AG-580](#)

## M

mailer [AG-21](#)  
maintenance reservation [AG-197](#)  
Manager  
    privilege [AG-491](#)  
managers server attribute [AG-491](#)  
managing vnode [AG-230](#), [AG-266](#)  
master provisioning script [AG-591](#), [AG-601](#)  
master script [AG-591](#), [AG-601](#)  
matching ACL entry [AG-495](#)  
max\_concurrent\_provision [AG-603](#)  
max\_group\_res [AG-299](#)  
max\_group\_run [AG-299](#)  
max\_group\_run\_soft [AG-299](#)  
max\_load  
    cycle harvesting [AG-126](#)  
max\_queueable [AG-299](#)  
max\_queued [AG-291](#)  
max\_queued\_res [AG-291](#)  
max\_run [AG-290](#)  
max\_run\_res [AG-291](#)  
max\_run\_res\_soft [AG-291](#)  
max\_run\_soft [AG-290](#)  
max\_running [AG-299](#)  
max\_user\_res [AG-299](#)  
max\_user\_res\_soft [AG-299](#)  
max\_user\_run [AG-299](#)  
max\_user\_run\_soft [AG-299](#)  
max\_walltime [AG-216](#)  
memory-only vnode [AG-230](#)  
min\_walltime [AG-216](#)  
MoM  
    log events [AG-431](#)  
MPI\_USE\_IB [AG-573](#)  
mpiexec [AG-571](#)  
multihost placement sets [AG-170](#)  
MUNGE [AG-509](#)

## N

natural vnode [AG-42](#)  
NEC SX-Aurora process swapping [AG-630](#)  
NEC SX-Aurora Tsubasa [AG-627](#)  
nhcas [AG-628](#)

node\_idle\_limit  
    server attribute [AG-588](#)  
non-consumable resource [AG-230](#)  
nonprimetime\_prefix [AG-194](#)  
normal\_jobs [AG-184](#)  
nves [AG-628](#)

## O

Operator  
    privilege [AG-490](#)  
operators server attribute [AG-491](#)  
opt\_backfill\_fuzzy [AG-112](#)  
overall limit [AG-230](#), [AG-285](#)

## P

partial process swapping [AG-630](#)  
password  
    invalid [AG-636](#)  
pbs.conf [AG-581](#), [AG-586](#), [AG-644](#)  
PBS\_AUTH\_METHOD [AG-422](#)  
PBS\_BATCH\_SERVICE\_PORT [AG-422](#)  
PBS\_BATCH\_SERVICE\_PORT\_DIS [AG-422](#)  
PBS\_CLIENT\_CONN\_RETRY\_INTERVAL [AG-379](#),  
    [AG-411](#), [AG-423](#)  
PBS\_CLIENT\_CONN\_RETRY\_TIMEOUT [AG-379](#),  
    [AG-411](#), [AG-423](#)  
PBS\_COMM\_LOG\_EVENTS [AG-423](#)  
PBS\_COMM\_ROUTERS [AG-423](#)  
PBS\_COMM\_THREADS [AG-423](#)  
PBS\_CONF\_SYSLOG [AG-426](#), [AG-435](#)  
PBS\_CONF\_SYSLOGSEVR [AG-426](#), [AG-435](#)  
PBS\_CORE\_LIMIT [AG-423](#)  
PBS\_CP [AG-423](#)  
PBS\_DAEMON\_SERVICE\_USER [AG-423](#)  
PBS\_DATA\_SERVICE\_PORT [AG-423](#)  
PBS\_ENCRYPT\_METHOD [AG-423](#)  
PBS\_ENVIRONMENT [AG-423](#)  
PBS\_EXEC [AG-379](#), [AG-423](#)  
PBS\_EXEC/share [AG-578](#)  
PBS\_HOME [AG-379](#), [AG-423](#)  
pbs\_iff [AG-644](#)  
PBS\_LEAF\_NAME [AG-424](#)  
PBS\_LEAF\_ROUTERS [AG-424](#)  
PBS\_LOCALLOG [AG-424](#), [AG-435](#)  
PBS\_LOG\_HIGHRES\_TIMESTAMP [AG-424](#)  
PBS\_MAIL\_HOST\_NAME [AG-23](#), [AG-424](#)  
PBS\_MANAGER\_SERVICE\_PORT [AG-424](#)  
pbs\_mkdirs [AG-636](#)  
PBS\_MOM\_HOME [AG-379](#), [AG-424](#)  
PBS\_MOM\_NODE\_NAME [AG-424](#)  
PBS\_MOM\_SERVICE\_PORT [AG-424](#)  
PBS\_MPI\_DEBUG [AG-573](#)  
PBS\_OUTPUT\_HOST\_NAME [AG-425](#)

## Index

---

PBS\_PRIMARY [AG-379](#), [AG-425](#)  
pbs\_probe [AG-636](#)  
PBS\_RCP [AG-425](#), [AG-581](#)  
PBS\_REMOTE\_VIEWER [AG-425](#)  
pbs\_rsub [AG-501](#)  
PBS\_SCHED\_THREADS [AG-425](#)  
PBS\_SCP [AG-425](#), [AG-581](#)  
PBS\_SECONDARY [AG-379](#), [AG-425](#)  
PBS\_SERVER [AG-379](#), [AG-426](#)  
PBS\_SERVER\_HOST\_NAME [AG-426](#)  
PBS\_START\_COMM [AG-426](#)  
PBS\_START\_MOM [AG-379](#), [AG-426](#)  
PBS\_START\_SCHED [AG-379](#), [AG-426](#)  
PBS\_START\_SERVER [AG-379](#), [AG-426](#)  
PBS\_SUPPORTED\_AUTH\_METHODS [AG-426](#)  
PBS\_TMPDIR [AG-427](#)  
PBS\_WEBAPI\_PORT [AG-427](#)  
PBS\_WEBAPI\_TLS [AG-427](#)  
pbsfs [AG-144](#)  
pcap\_accelerator [AG-535](#), [AG-539](#), [AG-541](#), [AG-588](#)  
pcap\_node [AG-535](#), [AG-539](#), [AG-541](#), [AG-588](#)  
PCIe [AG-627](#)  
permissions  
    logs [AG-580](#)  
pgov [AG-535](#), [AG-539](#), [AG-541](#), [AG-588](#)  
p-governor [AG-585](#), [AG-588](#)  
placement  
    task [AG-168](#)  
placement pool [AG-169](#)  
placement set [AG-169](#)  
placement sets  
    multihost [AG-170](#)  
policy  
    defining provisioning [AG-603](#)  
    files [AG-578](#)  
    location [AG-578](#)  
power profile  
    activate [AG-586](#)  
    deactivate [AG-586](#)  
power profiles [AG-583](#)  
power\_off\_iteration  
    server attribute [AG-589](#)  
power\_provisioning [AG-587](#)  
    server attribute [AG-587](#), [AG-589](#)  
    vnode attribute [AG-589](#)  
poweroff\_eligible [AG-589](#)  
PPS [AG-630](#)  
preempt\_order [AG-180](#)  
preempt\_prio [AG-181](#)  
preempt\_queue\_prio [AG-181](#)  
preempt\_sort [AG-181](#)  
preemption [AG-180](#)  
preemption via checkpoint [AG-187](#)  
preemptive scheduling [AG-180](#)

preemptive\_sched [AG-180](#)  
primary server [AG-425](#)  
prime\_spill [AG-195](#)  
primetime\_prefix [AG-194](#)  
privilege  
    Manager [AG-491](#)  
    Operator [AG-490](#)  
    user [AG-490](#)  
project [AG-285](#), [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#),  
    [AG-540](#), [AG-541](#), [AG-543](#)  
project limit [AG-285](#)  
    generic [AG-285](#)  
    individual [AG-285](#)  
project limits [AG-283](#)  
prologue [AG-586](#)  
provision\_policy [AG-593](#)  
provisioning  
    creation of hooks [AG-602](#)  
    defining policy [AG-603](#)  
    hooks [AG-591](#)  
    master script [AG-601](#)  
    writing [AG-601](#)  
    overview [AG-592](#)  
    policy [AG-593](#)  
    rebooting [AG-592](#)  
    reservations [AG-595](#)  
    vnode selection [AG-593](#)  
    vnode states [AG-596](#)  
pstate [AG-588](#)

## Q

qdel [AG-637](#)  
qmgr [AG-21](#), [AG-644](#)  
qrerun [AG-637](#)  
qstat [AG-644](#)  
qsub [AG-637](#)  
query\_other\_jobs [AG-580](#)  
queue [AG-33](#)  
    access to a [AG-492](#)  
    ACL [AG-493](#)  
    attribute  
        acl\_group\_enable [AG-500](#)  
        acl\_groups [AG-500](#)  
        acl\_host\_enable [AG-500](#)  
        acl\_hosts [AG-500](#)  
        acl\_user\_enable [AG-500](#)  
        acl\_users [AG-500](#)  
queue\_softlimits [AG-184](#)  
queued jobs [AG-285](#)  
queued\_jobs\_threshold [AG-291](#)  
queued\_jobs\_threshold\_res [AG-291](#)  
queues  
    creating [AG-25](#)

## Index

---

### R

- rcp [AG-425](#)
- rebooting
  - provisioning [AG-592](#)
- reservation [AG-532](#)
  - access to a [AG-492](#)
  - ACL [AG-493](#)
  - advance [AG-197](#)
  - ASAP [AG-197](#)
  - attribute
    - Authorized\_Groups [AG-501](#)
    - Authorized\_Hosts [AG-501](#)
    - Authorized\_Users [AG-501](#)
  - control of creation [AG-493](#)
  - degraded [AG-197](#)
  - instance [AG-197](#)
  - job-specific [AG-197](#)
    - ASAP [AG-197](#)
    - now [AG-197](#)
    - start [AG-197](#)
  - maintenance [AG-197](#)
  - now [AG-197](#)
  - reservation ID [AG-198](#)
  - soonest occurrence [AG-197](#)
  - standing [AG-197](#)
    - instance [AG-197](#)
    - soonest occurrence [AG-197](#)
- reservations [AG-196](#)
  - provisioning [AG-595](#)
- resource [AG-230](#)
  - built-in [AG-228](#)
  - consumable [AG-229](#)
  - custom [AG-229](#)
  - dynamic [AG-229](#)
  - indirect [AG-229](#), [AG-266](#)
  - non-consumable [AG-230](#)
  - shared [AG-230](#), [AG-266](#)
- resource limits [AG-283](#)
- resource usage limits [AG-283](#)
- Resource\_List [AG-532](#), [AG-533](#), [AG-535](#), [AG-536](#), [AG-539](#), [AG-540](#), [AG-541](#), [AG-543](#)
- Resource\_List.eoe [AG-587](#)
- resources
  - unset [AG-160](#)
- resources\_assigned [AG-541](#)
- resources\_available.eoe [AG-587](#)
- resources\_used.energy [AG-586](#), [AG-587](#)
- restrict\_user [AG-521](#)
- restrict\_user\_exceptions [AG-521](#)
- restrict\_user\_maxsysid [AG-522](#)
- resv\_enable [AG-501](#)
- resv\_enable server attribute [AG-493](#)
- roles [AG-489](#)

### RPM

- debuginfo [AG-635](#)
- rsync
  - configuration [AG-581](#)
- run\_count [AG-533](#), [AG-536](#), [AG-543](#)
- run\_time [AG-129](#)

### S

- sched\_preempt\_enforce\_resumption [AG-182](#)
- scheduler
  - log events [AG-431](#)
- scp [AG-425](#)
- scratch space [AG-254](#)
  - dynamic
    - host-level [AG-270](#)
    - server-level [AG-269](#)
  - static
    - host-level [AG-270](#)
    - server-level [AG-270](#)
- script
  - master provisioning [AG-601](#)
  - writing provisioning [AG-601](#)
- secondary server [AG-425](#)
- security\_context job attribute [AG-578](#)
- server
  - access to [AG-492](#)
  - ACL [AG-493](#)
  - attribute
    - acl\_host\_enable [AG-500](#)
    - acl\_hosts [AG-500](#)
    - acl\_user\_enable [AG-500](#)
    - acl\_users [AG-500](#)
    - flatuid [AG-506](#)
    - log\_events [AG-431](#)
    - managers [AG-491](#)
    - operators [AG-491](#)
    - resv\_enable [AG-493](#)
  - log events [AG-431](#)
  - parameters [AG-20](#)
  - primary [AG-425](#)
  - recording configuration [AG-21](#)
  - secondary [AG-425](#)
- server attributes
  - node\_idle\_limit [AG-588](#)
  - power\_off\_iteration [AG-589](#)
  - power\_provisioning [AG-587](#)
- server\_softlimits [AG-184](#)
- set\_power\_cap [AG-588](#)
- setting limits [AG-292](#)
- shared resource [AG-230](#), [AG-266](#)
- shares [AG-140](#)
- size
  - format [AG-235](#)

## Index

---

sleep

    vnode state [AG-589](#)

soonest occurrence [AG-197](#)

sort key [AG-147](#)

sshd [AG-351](#)

standing reservation [AG-197](#)

start reservation [AG-197](#)

states

    vnodes and provisioning [AG-596](#)

static fit [AG-169](#)

strict\_ordering and backfilling [AG-223](#)

string [AG-240](#)

string resource value

    format [AG-235](#), [AG-240](#)

string\_array [AG-240](#)

    format [AG-235](#), [AG-240](#)

support team [AG-647](#)

SX-Aurora [AG-627](#)

syslog [AG-434](#)

### T

task placement [AG-168](#)

TSUBASA [AG-627](#)

type codes [AG-431](#)

### U

unknown node [AG-140](#)

unknown\_shares [AG-140](#)

unset resources [AG-160](#)

usage limits [AG-283](#)

user

    access [AG-492](#)

    ACLs [AG-494](#)

    privilege [AG-490](#)

    roles [AG-489](#)

user limit [AG-230](#), [AG-285](#)

    generic [AG-285](#)

    individual [AG-285](#)

user limits [AG-283](#)

### V

VE [AG-627](#)

VE offloading [AG-627](#)

ve\_cput [AG-628](#), [AG-630](#)

ve\_mem [AG-628](#), [AG-630](#)

vector engine [AG-627](#)

vector host [AG-627](#)

version information [AG-635](#)

VH [AG-627](#)

virtual nodes [AG-41](#)

vnode [AG-41](#)

    borrowing [AG-228](#), [AG-266](#)

    managing [AG-230](#), [AG-266](#)

    memory-only [AG-230](#)

    natural [AG-42](#)

    selection for provisioning [AG-593](#)

    states and provisioning [AG-596](#)

vnode attributes

    last\_state\_change\_time [AG-587](#)

    last\_used\_time [AG-587](#)

### W

Windows

    password [AG-636](#)

writing provisioning script [AG-601](#)

### X

X forwarding [AG-428](#)

xauth [AG-428](#)

## Index

---