

Altair - Breeze User Manual

Version 2025.2.0



Contents

1	Introduction	5
1.1	Tracing and Profiling	5
1.2	Using Breeze Remotely	5
1.3	The Breeze Automation Platform	5
1.4	Summary of the Breeze Workflow	6
1.4.1	Tracing Your Application	6
1.4.2	Analysing a Trace with the Breeze UI	6
1.4.3	Analysing a Trace with Breeze Automation Platform	6
2	Installation	8
2.1	Tar file	8
2.2	Debian package	8
2.3	RPM package	8
2.4	Path setup	8
2.5	Required Items	9
2.6	Licensing	9
2.6.1	Floating Licenses and the Altair License Server	9
2.6.2	Legacy Ellexus Node-locked Licenses	9
2.6.3	Legacy Floating Licenses and the Ellexus License Server	9
3	Tracing an Application at the Command Line	10
3.1	How tracing works	10
3.2	Command Line Options	10
3.3	Grouping I/O operations	11
3.4	Tracing Applications on Remote or Multiple Hosts	12
3.5	Altair Grid Engine version 2022.2 and above	12
3.5.1	Job submission with no shell option	13
3.6	Altair Accelerator & Flowtracer	13
3.6.1	Integration using a Breeze environment configuration	13
3.6.2	Using Bypass Mode to ignore scheduler I/O	13
3.7	Limitations	14
3.8	Tracing Memory-Mapped Files	15
3.9	Removing Confidential Information from Trace Output	15
4	Using the Breeze GUI	17
4.1	Command Line Options	17
4.2	Entering License Location	17
4.3	Importing a Saved Trace	18
4.4	Importing a Saved trace tar file	18
5	Breeze GUI View Details	20
5.1	I/O Summary	20
5.2	Timeline View	21
5.3	Duration View	21
5.4	Profiling Data View	23
5.5	Profile Program	23
5.5.1	Bandwidth	26
5.5.2	Operation Count	26
5.5.3	CPU Load	26
5.5.4	Memory Consumption	27

5.6	Node View	27
5.7	Event View	27
5.8	Program Information View	30
5.9	Search View	30
5.9.1	Using the Search View to find files, libraries and network nodes	30
5.10	Files View	34
5.10.1	Group By	34
5.10.2	Summary Column	35
5.10.3	Filter By Path	35
5.10.4	Location Group	36
5.10.5	Read Group	36
5.10.6	Small Read Group	36
5.10.7	Large Read Group	36
5.10.8	Write Group	36
5.10.9	Small Write Group	37
5.10.10	Large Write Group	37
5.10.11	Seek Group	37
5.10.12	Small Seek Group	37
5.10.13	Other Operation	38
5.11	Program Files View	38
5.12	Network View	38
5.12.1	Location Group	39
5.12.2	Accept, Bind, Connect and Listen Groups	39
5.12.3	Read and Write Groups	39
5.13	Program Network View	39
5.14	Symlink View	40
6	Profiling file I/O	41
6.1	File-system mount points	41
6.2	File-system I/O Counts	41
6.3	File-system I/O Latency	41
6.4	Network I/O Count and Latency	41
6.5	Error Code Counts and Zero-Byte I/O Counts	41
6.6	Memory Consumption	42
6.7	Job CPU Usage	42
6.8	Host CPU Usage	42
6.9	Symlink Chain Counts	42
6.10	File-system Trawls	42
6.11	I/O Sizing	43
7	Guide to MPI Applications	44
7.1	Aggregating I/O data for the whole job	44
7.2	My job ran on 1000 hosts, which trace file should I look at?	44
7.3	Viewing MPI file I/O in Breeze	45
7.4	Tracking MPI comms I/O between ranks in Breeze	45
8	Breeze Automation Platform and Exporting Data	46
8.1	Breeze Automation Platform Licensing	46
8.2	Breeze AP Command Line Options	46
8.2.1	Examples:	50

9 Breeze Healthcheck	51
9.1 Introduction	51
9.2 Installing Breeze Healthcheck	51
9.3 Generating a Healthcheck Report	51
9.4 Creating an Empty Sample Report	51
9.5 Healthcheck Command Line Options	52
10 Container Support	54
10.1 Singularity	54
10.2 Docker	54
11 Troubleshooting	55
11.1 Turning off Breeze tracing for specific binaries	55
11.2 KDE on CentOS 7	55
11.3 X-Forwarding to MacOS	56

1 Introduction

Breeze is aimed at IT managers and people who deploy complex applications and engineers designing and optimising complex workflows. It gives you the information you need to solve software build and installation issues quickly. Breeze was originally developed by Ellexus, now an Altair company.

Even when the software and IT infrastructure have been well designed, installation and configuration problems often arise, particularly on distributed systems. Files stored in the wrong place can affect the performance of your application and everything else on the cluster. Breeze profiles your applications so you can be sure that they are using your compute resources as they should.

Breeze works by tracing and profiling application dependencies. It is sometimes referred to as a 'system debugger', to be used when debugging scripted flows or third-party applications. From build and installation to customer services and tuning, Breeze forms a common platform to enable IT managers and product engineers to work together and solve problems more quickly.

When an application works on one system but not on another, Breeze can record what the applications are actually doing and highlight the differences. Or you can use Breeze to see in detail what your customer has set up and quickly find out what is wrong.

Breeze is designed to be up and running as quickly as possible. You don't need to change anything about your programs or your system, just follow these simple steps to access your program trace in minutes.

In this guide you will learn how to analyse complex build processes and large applications, and you will learn how to optimise your program for your compute resources so that you can be sure to get the most from your system.

1.1 Tracing and Profiling

Breeze traces application arguments, environment and dependencies. This means you can troubleshoot build or installation issues and resolve missing files or libraries.

Breeze also records I/O patterns so that you can understand how your programs are using the network and file system. This means you can resolve performance problems and assess the ability of your application to scale in parallel environments.

1.2 Using Breeze Remotely

Breeze can be used interactively, but you can also trace and profile programs via the command line and across multiple execution hosts. Tracing applications using the command line utility does not require a license. This makes Breeze perfect for tracing applications on a job scheduling computer cluster, or using Breeze for customer services, because you can log the data and analyse it later on your own machine.

1.3 The Breeze Automation Platform

The Breeze Automation Platform gives you complete access to the trace and profile data via a command-line API so you can write your own custom queries and automate quality checks in your system.

1.4 Summary of the Breeze Workflow

1.4.1 Tracing Your Application

The first step in using Breeze is to trace the application. This can be done in the Breeze UI but it is more common to do this using the command line. Tracing an application at the command line does not require a license.

Assuming the Breeze bin has been added to \$PATH and the command you want to trace is, for example

```
ls -l /tmp
```

to run this command under trace using default settings you would simply type

```
$ trace-program.sh -f <output_dir> ls -l /tmp
```

which would place the trace files generated in the specified directory <output_dir>.

Please see the [Installation](#) section for more information on installing Breeze and the [Command Line Options](#) section for more details on the available options when tracing an application.

1.4.2 Analysing a Trace with the Breeze UI

Once again assuming the Breeze bin directory has been added to \$PATH and the trace you wish to analyse is stored in the directory <breeze_trace> the Breeze UI can be started by typing

```
$ breeze.sh
```

If not already configured, on first use you will be prompted to accept the user license agreement and set up details of where to find your license. Please see section [Entering License Location](#) for more details on how to configure the license.

Once the UI has started you can load your trace by selecting File > Import Saved Trace from the drop down menus.

Select the <breeze_trace> directory in the file explorer dialog and click OK. This directory can contain a Breeze trace in either the binary or ASCII formats. If you import a binary trace Breeze will automatically generate the ASCII version for the trace in <breeze_trace>/decoded as part of the import process.

If the trace is large you will be offered the choice to use a faster import that leaves out some of the data or to import data from a specific time window only.

Once the import is complete you can start to explore your data. The various views are explained in detail in chapter [Breeze GUI View Details](#) below.

1.4.3 Analysing a Trace with Breeze Automation Platform

As an alternative to using the Breeze UI you can generate the same data in flat files that can be used to include Breeze data in a scripted workflow.

If you have not already configured your license settings you will need to do this first.

The simplest way to configure your license is to start the Breeze UI and fill in the settings when prompted to do so. Please see section [Entering License Location](#) for more details on how to configure the license.

As before assuming the Breeze bin directory has been added to \$PATH and the trace you wish to analyse is stored in the directory <breeze_trace>, a complete set of Breeze Automation Platform (Breeze AP) output can be generated using the command

```
$ breezeAP.sh <breeze_trace> -all -p all
```

This command will store the generated output files in the <breeze_trace> directory.

Please see section [Breeze AP Command Line Options](#) for more details on the available options when processing a trace using Breeze AP.

2 Installation

Go to [Altair One](#) and register for an account. You can then log in and download Breeze and the Altair license server.

We have 3 formats of package, a “.tar.gz”, “.deb” and “.rpm”. Select whichever is easiest for you to work with.

It is important to note that Breeze assumes that it is available on the same, canonical path on every machine when following jobs from one execution host to another.

You should make sure that you choose the right version of Breeze (32 or 64 bit) for the machine you want to run it on.

2.1 Tar file

To install, just untar the product on any filesystem which is accessible from all the hosts running workloads which you wish to trace.

2.2 Debian package

Install using your package manager. This will install Breeze to /opt/altair/breezeAP_2025.2.0_x86_64 by default.

e.g.

```
$ sudo dpkg -i breeze_2025.2.0-0_amd64.deb
```

To install to a different location use the --install-dir=dir argument to dpkg.

2.3 RPM package

Install using your package manager. This will install Breeze to /opt/altair/breezeAP_2025.2.0_x86_64 by default.

e.g.

```
$ sudo rpm --install breeze-2025.2.0-0.x86_64.rpm
```

To install in a different location use the --prefix=<path> argument to rpm.

2.4 Path setup

For simplicity this guide assumes that once Breeze has been installed it has been added to the user's path and can be started without specifying a fully qualified path to the installation location, but this is not required by Breeze.

That is, if Breeze has been installed in /apps/altair/breezeAP_latest_x86_64 it is assumed that the user has altered their \$PATH to include this directory either by manually typing

```
$ export PATH=$PATH:/apps/altair/breezeAP_latest_x86_64
```

or adding a similar command to their account login scripts.

2.5 Required Items

The following items should be found on any standard Linux installation. Breeze will detect them on start up and warn you if it detects a potential problem.

- You will need to have GDK+ 2.2 installed. We recommend at least GTK+ 2.8. This is part of the minimum install with most Linux distributions.

2.6 Licensing

Although generating a Breeze trace does not require a license, the main Breeze U/I and Breeze Automation Platform need access to a valid license to run.

Breeze uses Altair units based licensing by default. Breeze will checkout 30 units whilst running. You will require a license that has the HWHPCBreeze feature for this to work.

For backwards compatibility floating Altair Breeze licenses can still be used if you set and export the following environment variable before running `BREEZE_UNITS=false`. This will checkout a BREEZE license.

Legacy Ellexus licenses were of two different types: Node-locked or Floating. Both types of licenses were provided as a plain text ASCII file saved with a `.lic.json` file extension.

2.6.1 Floating Licenses and the Altair License Server

In order to use a floating license you will need the Altair License Manager (ALM) corresponding to your architecture. This can be downloaded from AltairOne, please see the Breeze Installation Guide for details.

2.6.2 Legacy Ellexus Node-locked Licenses

If you have been provided with a node-locked license this file must be placed on the host whose details match those specified within the license, in a directory readable by all potential users of Breeze.

2.6.3 Legacy Floating Licenses and the Ellexus License Server

Please see the Breeze Installation Guide for installation instructions for the legacy Ellexus license server.

3 Tracing an Application at the Command Line

Tracing an application is done from the standard shell command prompt.

Generating a trace in this way does not require any special permissions or license, and can be done by any user authorised to run the application under investigation. As a result Breeze can be used for application support. The Breeze tracing engine can be freely downloaded from [Altair One](#) by a customer who is having an issue, and used to generate an application trace that they can send to you for analysis.

To trace and profile an application you just type `trace-program.sh -f <output_dir>`, followed by your command and arguments. For example, if you wish to trace the command

```
ls -l /tmp
```

you would simply type

```
$ trace-program.sh -f <output_dir> ls -l /tmp
```

which would place the trace files generated in the specified directory `<output_dir>`. If the output directory specified in the `-f` option exists and already contains trace data, the script will display a warning message and exit.

By default Breeze will output any errors to `stderr`, including a final "trace complete" message. In most cases this does not cause any issue for the traced application but occasionally this can interfere with normal operation, most commonly where a secondary job is submitted, such as via `ssh`, and the output of that command is read by the launching application. Breeze's output, including the "trace complete" message, can be redirected to a separate log file by use of the `--errlog` option (see section [Command Line Options](#) below).

3.1 How tracing works

Breeze works by wrapping up your application in a special library designed to replace the interface between your application and the Linux kernel. This allows us to intercept library calls without modifying your system or your application.

3.2 Command Line Options

The following section lists all the valid command line options used by `trace-program.sh`. You can also see all the options offered by `trace-program.sh` by running the script with no options. All options to `trace-program.sh` must be specified before the command to be traced.

```
--post-trace=<post-trace-command>
```

```
-c <post-trace-command>
```

Execute a post-trace command after the traced program has finished.

The post-trace command itself won't be profiled, traced or monitored. You can use this command to run a short post-processing script, or to create a flag file, e.g., `--post-trace="touch /path/to/flag/file"`. If the post-trace command doesn't finish within 10 minutes, it will be killed.

```
--errlog=<filename>
```

Record Breeze error messages in the specified file. If this option is not set, errors will be sent to `stderr`.

```
--output <output directory>
```

```
-f <output directory>
```

The directory that trace data will be written to, and which is used by Breeze for temporary storage. Data will be written to a timestamped sub-folder. If not specified data will be output to \$HOME/breezetrcs.

```
--relocate <output directory>
```

Directory where trace data will be copied after the run has finished. May be used to speed up execution time of the program under trace by logging to local storage, and transferring the data to network storage afterwards.

```
--child=<[-bsub][,-lsbatch][,-lsrun][,-pbsdsh][,-pbs_tmrsh][,-qsub]          [,-rsh][,-sbatch][,-srun][,-ssh][,-su][,-sudo]>
```

```
--child=<yes|no>
```

```
-r
```

This option controls whether or not Breeze will follow an application to a new execution host.

The option can be specified as either a comma separated list of unsupported launching commands each prefixed with a - or one of all or no.

The value all is equivalent to listing all valid job launching commands.

By default, when this command line option is not specified, all valid job launching commands are supported.

Setting this option to no disables tracing of any child jobs.

The currently supported list of commands recognised by this option is bsub, lsbatch, lsrun, pbsdsh, pbs_tmrsh, qsub, rsh, sbatch, srun, ssh, su and sudo.

The new host must have an identical Breeze installation in the same directory as the first machine, and the trace output directory must be located on a shared file system that is mounted in the same place on each machine.

```
--child-job=yes
```

```
--child-job
```

Track child jobs. When a one or more child jobs are launched from a top-level command/script then the top-level job waits for all the child jobs to complete. This option is off by default.

```
--shell=<shell path>
```

```
-s <shell path>
```

Path to your shell. This is used in tracing interactive sessions executed using su, ssh, and similar programs.

3.3 Grouping I/O operations

Breeze collects data on sets of I/O calls. These functions are aggregated into the following groups:

accept	accept
access	access, chdir, readlink, realpath, stat, ...
connect	connect
create	creat, open (if file is created), tmpfile, mkdir, ...
delete	remove, rmdir, unlink, ...
fschange	chmod, link, rename, ...
glob	glob, glob64
open	open, opendir, ...
read	fgets, fread, mmap, read, readdir, recv, scanf, ...
seek	lseek, fseek, rewind, ...
write	error, fwrite, printf, putc, send, warn, write, ...

3.4 Tracing Applications on Remote or Multiple Hosts

Breeze currently supports tracing applications on remote hosts using `bsub`, `qsub`, `rsh`, `ssh`, `srun`, `sbatch`, `lrun`, and `lsbatch`.

The initial `trace-program.sh` script can be submitted to supported job schedulers such as `bsub` or `qsub` directly as long as the Breeze installation is available via the same path on all possible execution host nodes.

In addition, if the traced program runs a command on a new execution host via one of the supported commands, Breeze will attempt to re-write the command so that this task will also be traced. The output directory used for the command on the new execution host will be created under the output directory specified by the initial `-f` option, which must therefore be available on all possible execution host nodes. This new output directory will be named:

```
<output_directory>/childtrace-<hostname>-<PID>-<UID>
```

If the command was submitted as part of a job array the array index of the job under trace will be appended giving a full output directory specification of:

```
<output_directory>/childtrace-<hostname>-<PID>-<UID>-<array_index>
```

3.5 Altair Grid Engine version 2022.2 and above

Version 2022.2 and above of Altair Grid Engine includes Breeze integration.

To enable Breeze for a queue, use `qconf` to change the `execd_params` configuration item. See the manual page for `sge_conf(5)`. The following three parameters are mandatory:

- `AGE_BREEZE_MODE`:
 - Set to `ALWAYS` to turn Breeze on for all jobs on this queue.
 - Set to `DEFAULT_ON` for Breeze to be enabled for all jobs on the queue unless the `AGE_BREEZE` environment variable is set to 0 (with `qsub -v`).
 - Set to `IF_REQUESTED` for Breeze to be enabled for a job on the queue only when the `AGE_BREEZE` environment variable is set to 1 (with `qsub -v`).
 - Set to `NEVER` (the default value) to prevent Breeze being enabled on the queue.
- `AGE_BREEZE_INSTALL_PATH`: set to the full path of the directory where Breeze has been installed.
- `AGE_BREEZE_STARTER_METHOD`: set to the full path of the Breeze `trace-program.sh` script in the `bin` directory. If not set when profiling is requested, Altair Grid Engine will use a value relative to the Breeze installation path: `AGE_BREEZE_INSTALL_PATH/trace-program.sh` **Note that this path is the old path Breeze used to be packaged as. In order for Grid Engine to find this, please run the `compatibility-symlinks.sh` script found in the `sbin` folder of your Breeze install with the `create` parameter. This will create symlinks to simulate the old Breeze directory structure.**
- `AGE_BREEZE_OUTPUT_DIRECTORY`: set to the full path of a directory where Breeze will write its output trace files.

Note: if Altair Mistral is also enabled for a job, then only Breeze profiling will happen. Breeze and Mistral profiling cannot both be used on the same job.

3.5.1 Job submission with no shell option

When you submit a job with no `shell` option for quick execution, AGE executes the command directly as shown below.

```
$ qsub -q test.q -shell n -b y sleep 3600
Your job 513 ("sleep") has been submitted

$ ps aux | grep sge_shepherd
sge_execd(9548)-+-sge_shepherd(59616)---sleep(59617)
```

Breeze profiling with `trace-program.sh` introduces a shell for command execution, even if the job is submitted with no `shell` option. Breeze modifies the job submission options by prepending `-b n` and including `trace-program.sh` before the command to profile the remotely executing commands, as shown below.

```
$ ./bin/trace-program.sh -f ~/breezetraces qsub -q test.q -shell n -b y sleep 3600
Your job 514 ("sleep") has been submitted

$ ps aux | grep sge_shepherd
sge_execd(9548)-+-sge_shepherd(60152)---514(60154)---sleep(60173)

$ qstat -j 514 | grep submit_cmd
qsub -b n -q test.q -shell n
/opt/altair/breezeAP_trunk_x86_64/bin/trace-program.sh -r -q --trace=yes
--child-env BREEZE_INSTALL_DIRECTORY=/opt/altair/breezeAP_trunk_x86_64
...
...
sleep 3600
```

3.6 Altair Accelerator & Flowtracer

3.6.1 Integration using a Breeze environment configuration

If more control over tracing is needed, you can take advantage of the fact that both Accelerator and Flowtracer can set user-specified environment variables when they run jobs. Since Breeze can be started by setting appropriate environment variables, this provides a basic mechanism for tracing such jobs.

An example script (`BREEZE.start.tcl`) that can be used to set these environment variables is included in the `docs/samples/flowtracer` directory. Just modify the Breeze path in this file and then save it as `BREEZE.start.tcl` in the Flowtracer/Accelerator environment directory (`$VOVDIR_LOCAL/environments`). You should then be able to add `+BREEZE` to the environment for each task in the flow that you wish to trace with Breeze. Alternatively set `$VOV_ENV_DIR` and save `BREEZE.start.tcl` there.

3.6.2 Using Bypass Mode to ignore scheduler I/O

Accelerator and Flowtracer perform some I/O of their own, which is generally not interesting when using Breeze to analyse I/O of user jobs. The `BREEZE_BYPASS_PROGRAMS` environment variable can be used to avoid tracing this scheduler I/O, by setting it to the parent directory of the Accelerator or Flowtracer installation (usually named

in \$VOVDIR). The BREEZE_BYPASS_DISABLE variable can be used to turn tracing back on when a child job is started.

The following configuration is suggested:

```
export BREEZE_BYPASS_PROGRAMS=$(dirname $VOVDIR)
export BREEZE_BYPASS_DISABLE=VOV_JOBID
unset BREEZE_BYPASS_TO_NEW_TRACE
```

In the settings above Breeze will turn off tracing when a program or script is called from the directory above VOVDIR and it will turn back on once VOV_JOBID is set.

In general, if the BREEZE_BYPASS_PROGRAMS environment variable is set to a list of programs and directories, then any program which matches an entry in the list will be run in bypass mode. For example, `export BREEZE_BYPASS_PROGRAMS="emacs, /usr/local/"`, would run emacs and any program in /usr/local/, or a sub-directory such as /usr/local/bin, in bypass mode. For directories, ensure that the entry is an absolute path, whereas binaries should be listed by name as shown in the example.

If BREEZE_BYPASS_TO_NEW_TRACE is set then each time we leave bypass mode we start a separate trace, but this is not recommended.

3.7 Limitations

To trace a compound command such as `command1 && command2` or a pipeline such as `command1 | command2`, you must quote the command in order to prevent the shell from interpreting `command1` as an argument to `trace-program.sh` and piping its output into `command2`. For example:

```
$ trace-program.sh -f <output_directory> "command1 | command2"
```

The other option is to wrap the entire command in a shell. For example:

```
$ trace-program.sh -f <output_directory> sh -c \
"cd /apps; ./io_command | command2"
```

It is important to note that Breeze will not automatically detect compound commands when re-writing job submissions to remote hosts.

Another option is to add your command to a script and trace that script:

```
$ cat test.sh
#!/bin/bash

HELLO=THREE /usr/bin/echo $HELLO
```

```
$ trace-program.sh -f <output_directory> ./test.sh
```

Alternatively, you can source `trace-program.sh` to set up the environment so that all subsequent commands are traced. Then execute the commands you would like to profile and exit the shell session:

```
$ source trace-program.sh -f <output_directory>
$ cd /apps
$ ./io_command | command2
$ exit
```

3.8 Tracing Memory-Mapped Files

When tracing applications that map files into memory with `mmap`, Breeze traces the initial `mmap` operation if it is backed by a file. Any subsequent operations on the memory area itself are not traced. For example, when an application calls `mmap`, Breeze will trace the read/write operation for the file in question. If the application subsequently reads from or writes to the memory area, Breeze will not trace the memory I/O operations.

If an application calls `mmap` with `MAP_ANONYMOUS` flag (i.e., the mapping is not backed by any file), Breeze will not trace the `mmap` operation. Breeze also doesn't trace `munmap` operation, which removes an existing mapping.

3.9 Removing Confidential Information from Trace Output

It is possible that while tracing an application Breeze may have captured information you do not want to share with the team that will analyse the trace output, such as confidential file names.

By default Breeze creates binary files as this is more space efficient, however it is possible to convert this binary output into plain text using the `decode-trace.sh` script which can be found in the `bin` directory of the installation. The plain text files can then be easily processed to remove confidential information.

The script takes two parameters:

```
$ decode-trace.sh <input_directory> [output_directory]
```

If the output directory is not defined, the script will place the decoded trace in

```
<input_directory>/decoded
```

The `<input_directory>` should be a Breeze trace output directory. This will either be the directory passed as the `-f` option to a `trace-program.sh` command or a trace directory automatically created as the result of running a command on an alternate execution remote host as described in section [Tracing Applications on Remote or Multiple Hosts](#) above.

All strings, names and variables in the trace are listed in the file called `strings` in the top-level of the decoded trace directory structure. This file can be edited with any plain text file editor, enabling the user to change any confidential values.

The `strings` file is a simple CSV format file containing two columns. The first column is a numeric string ID number that is used to identify this string in all other trace files. The second column is the string captured within the traced application.

When editing the `strings` file it is important that no string value is completely removed. Instead, remove sensitive information by editing values. White-space is not allowed within this file and should be replaced with the `#` character, although it is recommended for simplicity that you try to avoid using white-space in your replacement strings.

The following is a cut down example of a `strings` file:

```
1,2.12.0
2,a7ac0988c4305c4617b19dab203d70107b982b56
3,2.17
4,Linux
5,www.altair.com
6,3.10.0-693.5.2.el7.x86_64
7,x86_64
8,/home/alice/breezeAP_latest_x86_64/tests/21/testscript.sh
9,/usr/bin/readlink
```

```
10, head#-n#1  
11, /usr/bin/grep
```

Assume that for security reasons you do not wish to share either user or host names with the team analysing the trace. In this case lines 5 and 8 should be updated.

Although there is no restriction on the text used, other than use of whitespace as described above, it is recommended that data is replaced with a substitution that should make sense in context, for example

```
5, www.EXAMPLE.com  
6, 3.10.0-693.5.2.el7.x86_64  
7, x86_64  
8, /home/USERNAME/breezeAP_latest_x86_64/tests/21/testscript.sh
```

Once all confidential data has been updated, the plain text version of the trace can be sent to the team that will analyse the trace, in place of the original, unmodified binary output.

4 Using the Breeze GUI

Start Breeze with the `breeze.sh` script available at the top level of the installation.

Although Breeze has a Java GUI, it comes with its own version of Java so you don't need to have Java installed to run it. You do need to have write permissions in the current working directory from which you launch Breeze, and you also need to have enough RAM to run a Java Eclipse application. Usually 1 GB of RAM is sufficient.

4.1 Command Line Options

The following section lists all the valid command line options used by `breeze.sh`.

`--tmp=<directory>`

Sets the directory where Breeze will store temporary trace data to `<directory>`. If the `--tmp` option is not used then Breeze will store temporary trace data in `/tmp`.

`--mem=<size>`

Sets the maximum heap size of the Java Virtual Machine to 1 GigaByte. Use this option if you are working with big traces and get `OutOfMemoryError`. The size argument must only contain numbers and may end with one of the following units: k, K, m, M, g, G.

`-h`

Displays a usage information message.

4.2 Entering License Location

The first time the Breeze UI is started you will be prompted to enter your license details. You will also be prompted to update these details if your configured license is invalid for any reason, for example it is not possible to contact the configured license server.

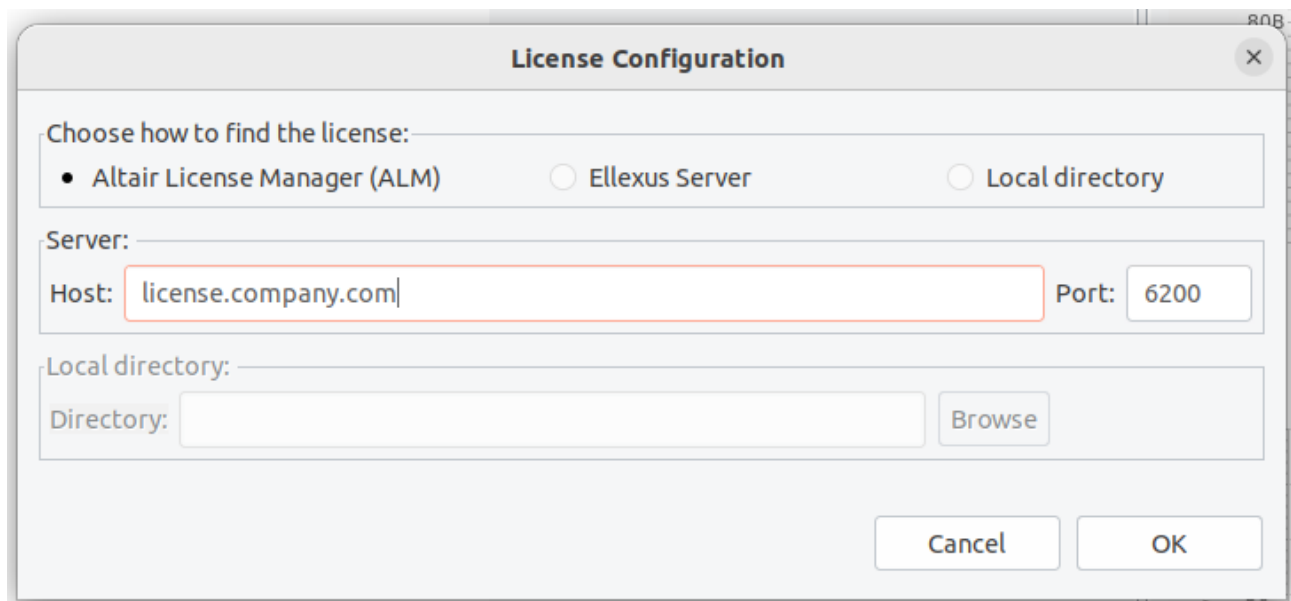


Figure 1: Breeze license configuration dialog

In most cases Breeze should be configured for use with a license server (see section [Floating Licenses and the Altair License Server](#)). To do this make sure the "License server" option is selected, place the fully qualified

domain name (or IP address) of your license server in the “Host” field, and the port on which the server is listening for license requests in the “Port” field. The port will be 6200 by default. If you do not know the location of your license server please contact your local administrator for the correct details.

Alternatively, if you have been provided with a license file (see section [Legacy Ellexus Node-locked Licenses](#)) select “Local directory” and then use the “Browse” button to select the directory that contains your license. Please note that the license file must be named with a `.lic.json` file extension to be recognised.

4.3 Importing a Saved Trace

To import a trace select `File > Import Saved Trace` from the drop down menus.

Select the directory that contains your trace and click OK. You will then be asked whether you wish to load “All trace data” or “Fast import mode - program and profiling data only”. If you select fast mode Breeze will not import all events, and as a result the event view and files views will have missing data. However every file opened will still be listed.

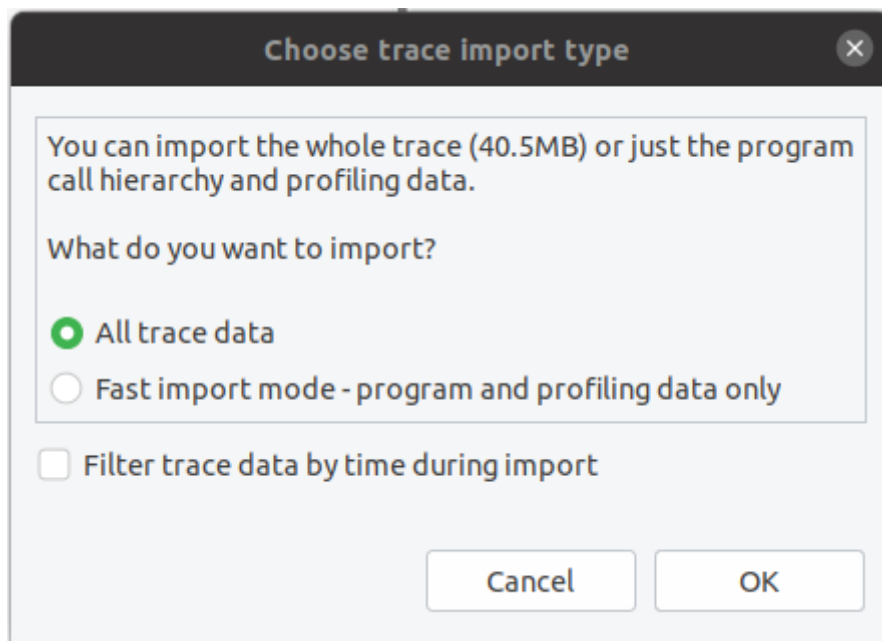


Figure 2: Import trace dialog

There is also an option “Filter trace data by time-range during import”. This will decode and open the trace, determine the start and end time, and then allow you to select a time range to be imported. Data before this time range is heavily filtered, only including program forking and execing. Data after this time range is not imported at all.

4.4 Importing a Saved trace tar file

If you have a trace created with the `--relocate` option then the trace will be in a `.tar.gz` archive. These can be imported directly by selecting `File > Import Trace Tar File` from the drop down menus.

You will then be able to select the compressed tar file that contains the trace. The tar file will be extracted and you will be then be presented with the same dialog as when importing a trace above.

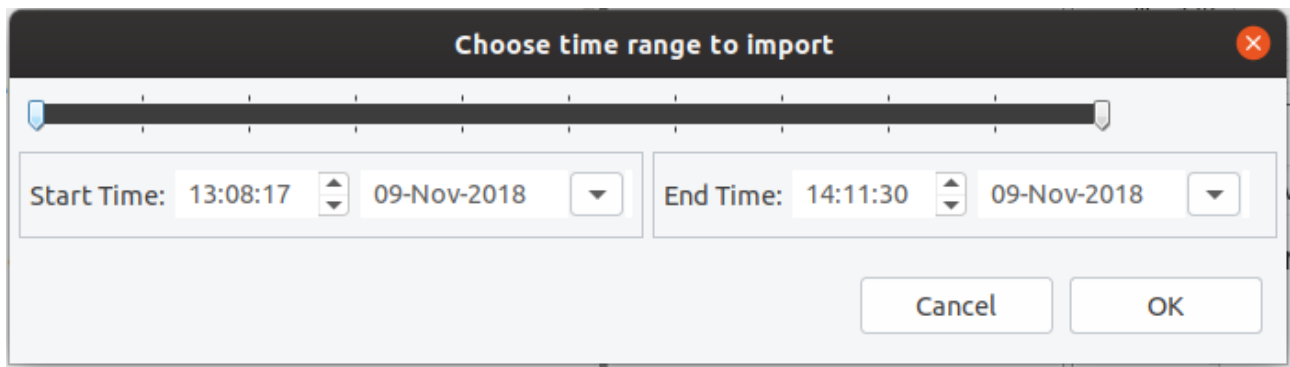


Figure 3: Select trace import time range dialog

This option is designed to work with tar files created in this way, and assumes that the tar file name matches the extracted file name. If trying to import tar files created any other way the extracted trace may not be found automatically.

5 Breeze GUI View Details

5.1 I/O Summary

This is the view that is displayed by default. It presents a breakdown of the I/O done by the application that was traced. Each I/O call is classified in one of the following categories:

Bad I/O: these are I/O operations that are considered detrimental to your file system. It regroups the following sub-categories:

- Small Read and Write;
- Failed I/O: here, failed Open and Close calls are also included;
- Zero-byte I/O;
- Backward Seek;
- Open and Closes on unused files;
- Stat on unused files;
- Failed Network I/O.

Medium I/O: these are I/O operations that could impact the performance of your file system. It is made of:

- Forward Seek;
- Stat on used files;
- Open and closes on files used a little;
- Delete Operations;
- Successful Sync operations;
- Large Reads and Writes.

Good I/O: these are the patterns that should take full advantage of your file system capabilities. It regroups the following:

- Open and Closes on files used a lot;
- Successful network I/O;
- Medium Sized Read and Write.

The threshold for deciding whether an operation was “small” or “large” and whether a file was used “a little” or “a lot” is defined by `BREEZE_PROFILE_SMALL_IO` and defaults to 32,768B (32kB) and `BREEZE_PROFILE_LARGE_IO` and defaults to 100MB.

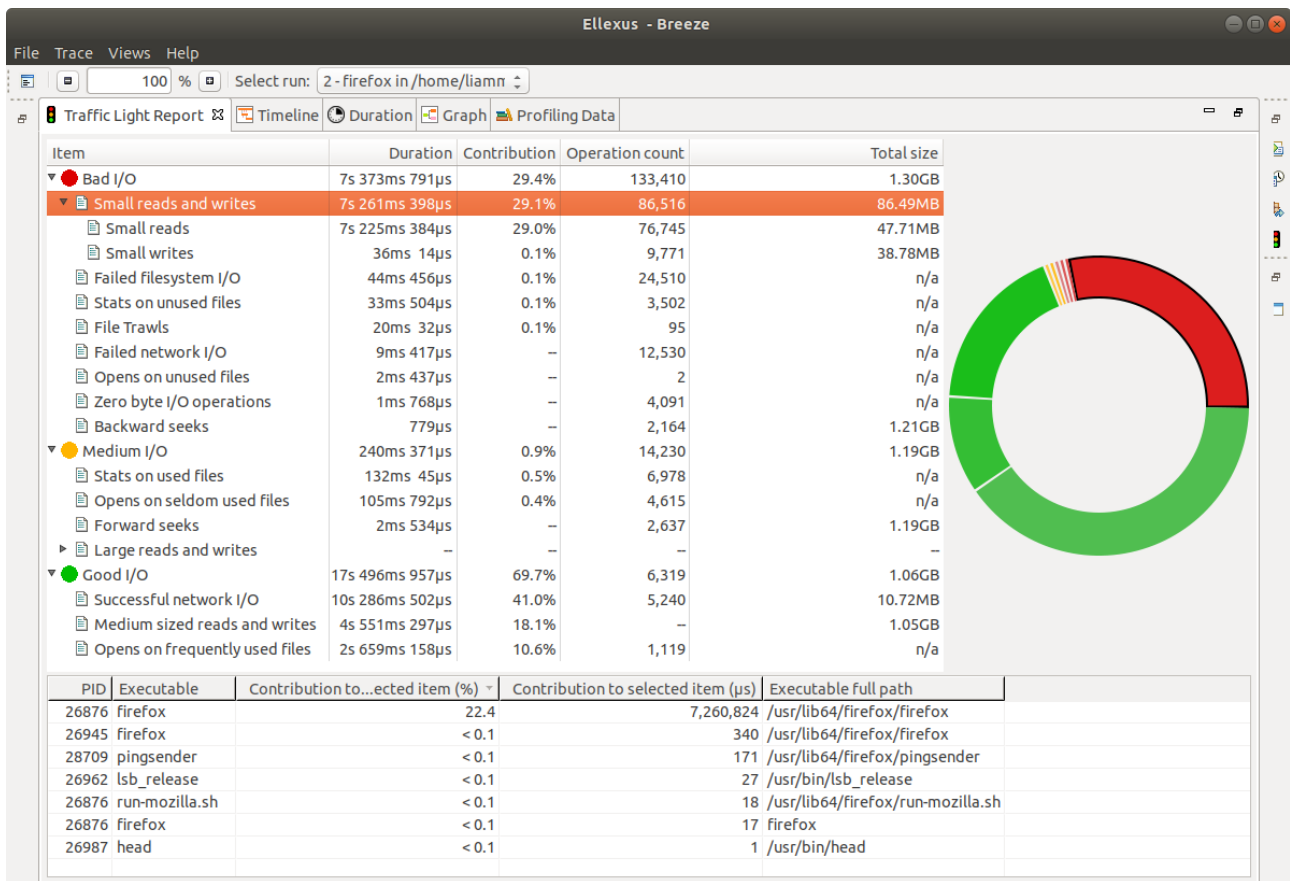


Figure 4: I/O Summary view

5.2 Timeline View

The **Timeline View** shows you a tree of processes with timing bars. The **Timeline View** is displayed in the main central pane by default.

The zoom level can be controlled either by using the mouse wheel while holding 'ctrl' key or by the zoom control on the application tool bar. In the **Timeline View** it is also possible to scale to a particular program. Right click on the program and choose 'Select and zoom to fit in Timeline View' and the **Timeline View** will zoom so that the selected bar fills the window.

5.3 Duration View

If you want to find information about which of your programs are taking the most time, you can use the **Duration View**. The **Duration View** is displayed in the main central pane by default. This view sorts program by duration by default showing the longest first. The sort order can be changed by clicking on column headers.

You'll find a right-click menu for each table item similar to that in the **Timeline View** so that you can find more details about any suspicious programs you have identified.

It is also possible to filter the elements of the table by specifying a time in the filter text box. The filter will recognise dates specified in either the highlighted string date format or as a unix time since epoch. Activating the filter will show on the table only the processes that were active at the time specified.

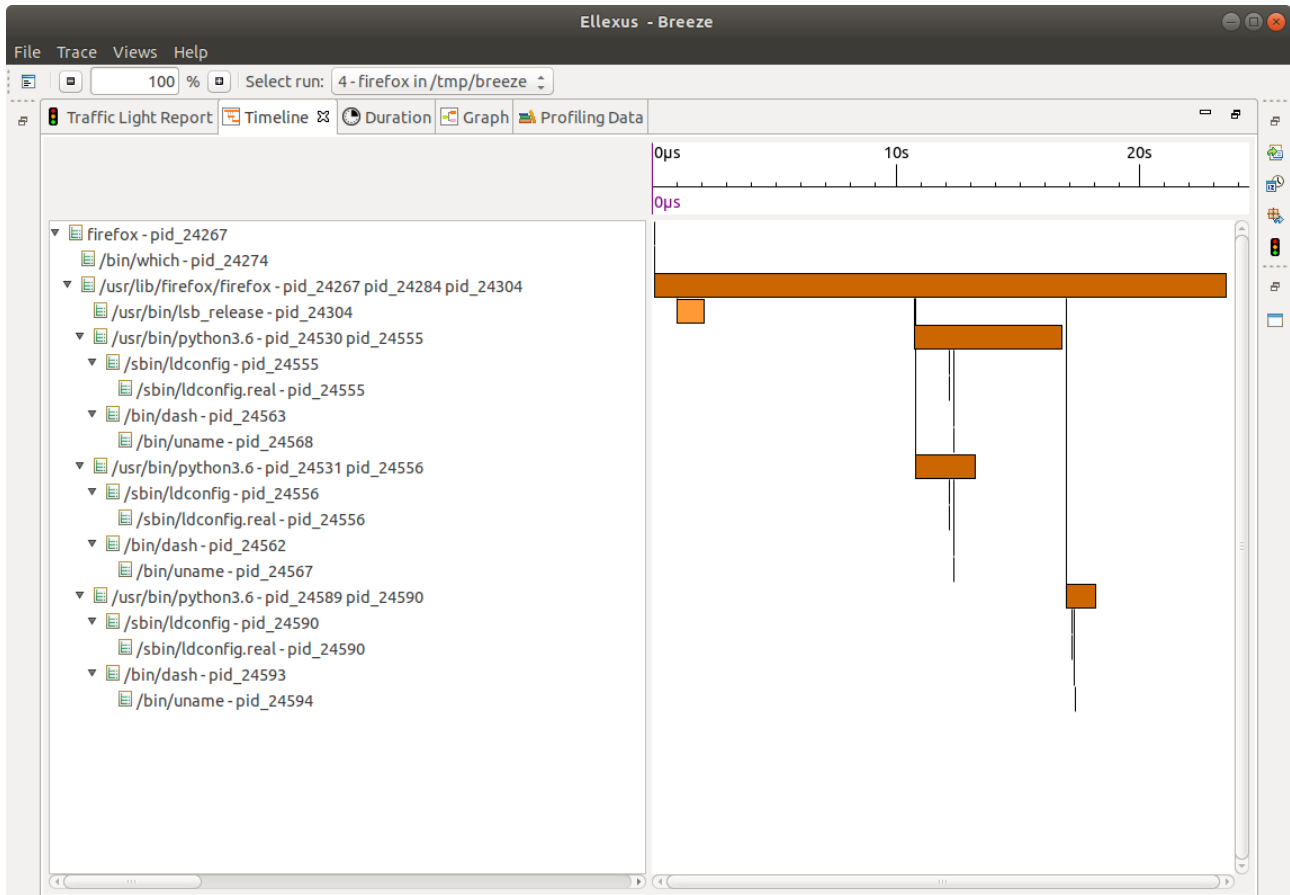


Figure 5: Timeline

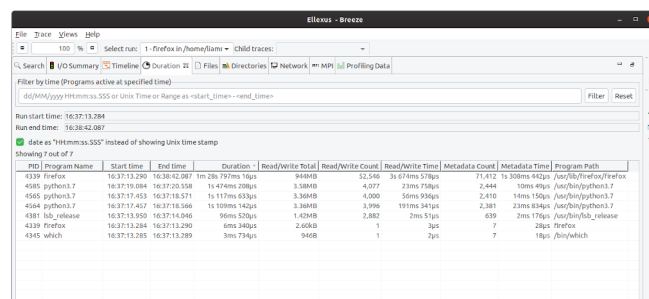


Figure 6: Duration - shows all the processes that make up the job sorted by duration

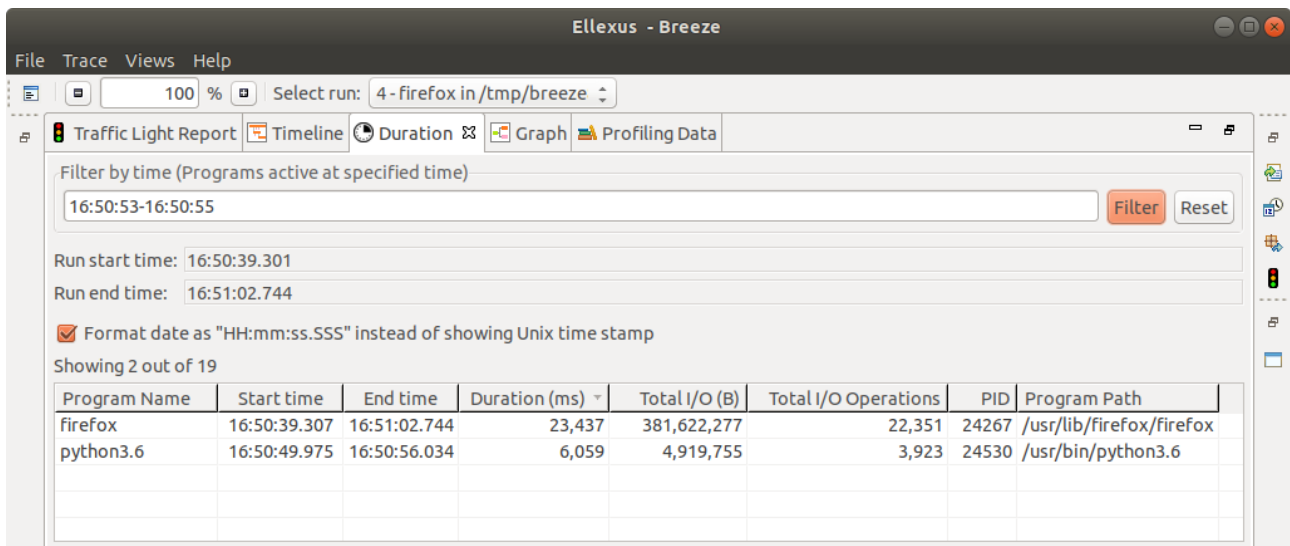


Figure 7: Filtering what is shown in the duration view by time

5.4 Profiling Data View

The Profiling Data View tab displays the I/O profiling data collected over time.

By default the File-system I/O Counts mode will be displayed for all process ID's. As can be seen from the screenshot above, the top area controls the actual profiling data currently being displayed, below that is a graph of the data and then the actual data in a table.

You can change what I/O statistics are displayed by choosing the required profiling mode from the Profiling Mode drop down box. It is also possible to view statistics for an individual process ID by choosing the process from the PID drop down box.

Some of the I/O statistics views are further divided by call name (i.e. File-system I/O latency) and the Call Name drop-down box will be enabled so that profiling data for each call name can be viewed.

Memory Usage statistics only display data for an individual process and will choose the first process in the PID list if one is not already selected.

Host CPU statistics are collected for the compute host that the Breeze trace was collected on. As these are per-host numbers they are not split per-process.

All the views allow to select multiple lines in the table and with one or more lines selected a right-click pop-up menu will allow the user to navigate to the duration view with a time filter already set that spans the time frame of the selected rows.

The graph displays selected aspects of the table data and usually aggregates the data. Clicking on the bars in the graph will highlight the first line in the table that contributed to that bar.

Right click on a bar in the graph and a pop-up menu will be displayed with the option to show the data in the duration view. Choosing the option will display the duration view with the appropriate time filter set.

5.5 Profile Program

Once a process has been selected in the **I/O Summary** or **Timeline** then the Profile Program view will be displayed. This shows profiling data for the selected process.

The top of the view has the name and total runtime of the process.

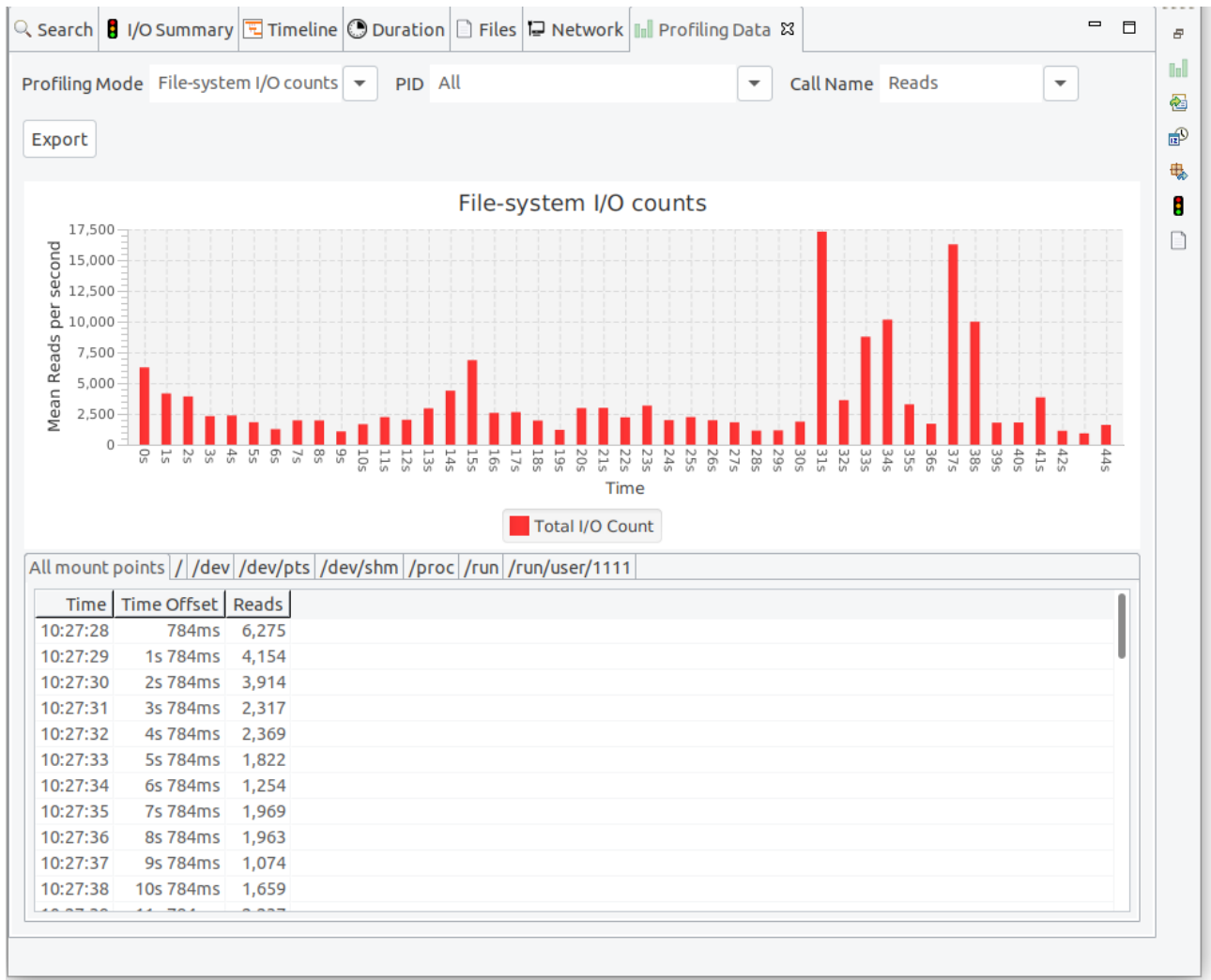


Figure 8: Profile view showing the number of read operations by mount point over time

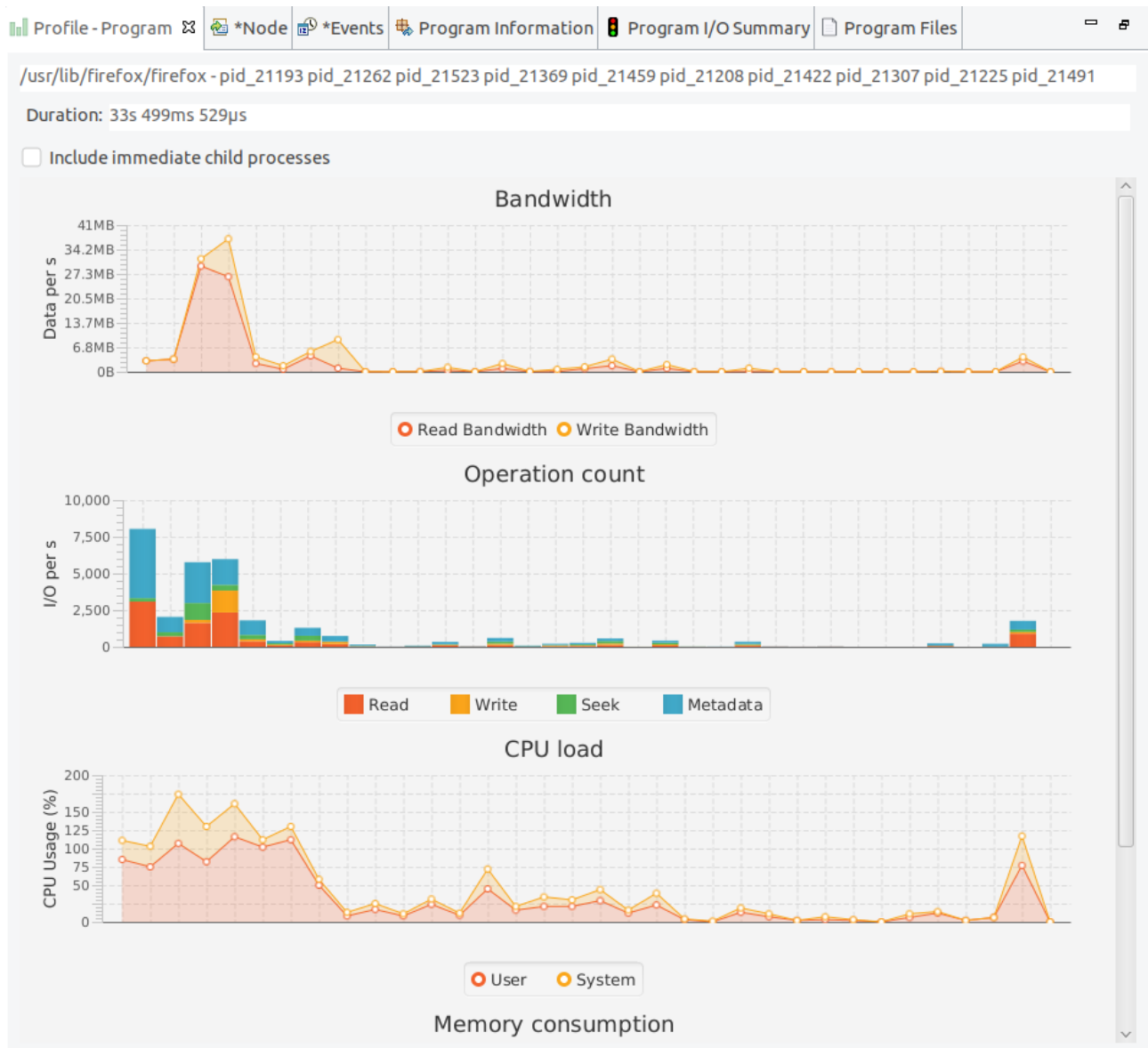


Figure 9: Profile program showing operations over time for the selected program

There is also an option “Include sub-process tree”, when this is checked the processes that were forked or execed by this process (child processes) and all of their sub-processes are included in the graphs.

There are four graphs - all against time. The graphs are aligned, so that you can see how the data is related. Hovering the mouse over the graphs will display the underlying data. The graphs are as follows:

5.5.1 Bandwidth

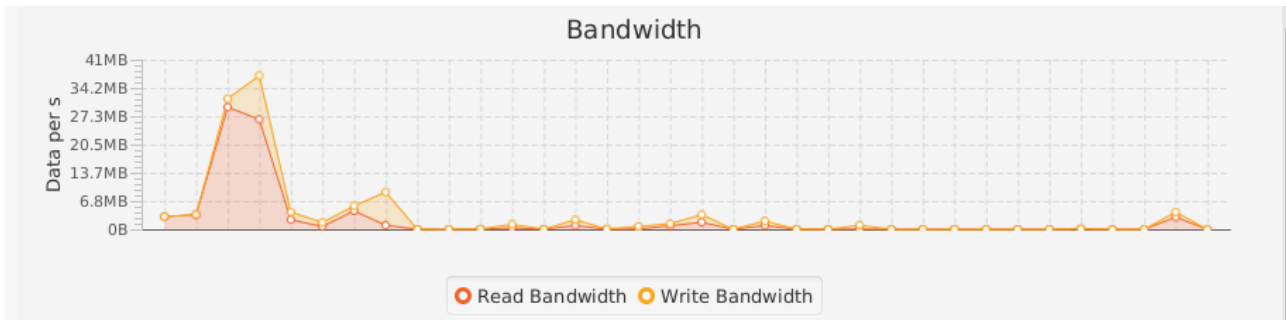


Figure 10: Program profile showing read and write bandwidth over time for the selected program

This graph shows the amount of data in read and write operations over time. The data is normalised, so that it reports the amount of data transferred per second. More detail can be seen in the [Profiling Data View](#), File System I/O Counts and selecting the process that you are interested in the “Bytes Read” and “Bytes Written” tabs.

5.5.2 Operation Count

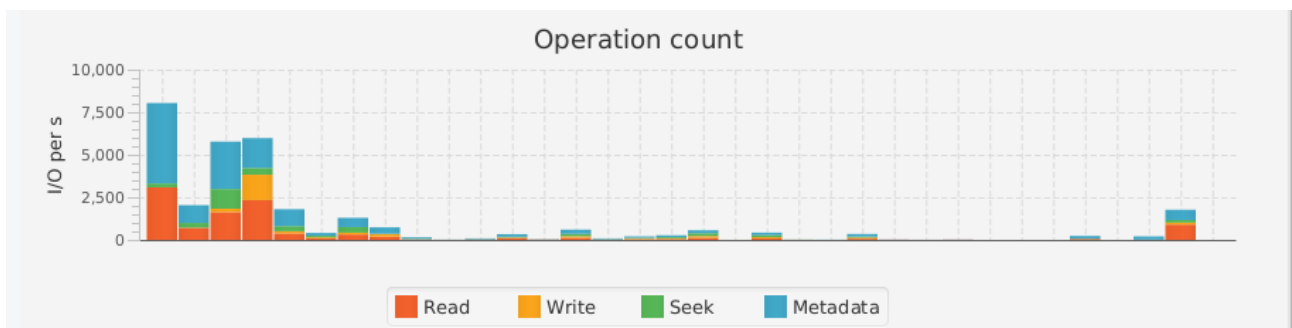


Figure 11: Program profile showing the number of I/O operations over time grouped by type for the selected program

This graph shows the number of operations performed over time. These are split into reads, writes, seeks and metadata operations. Metadata operations consist of the following (Opens, Globs, Deletes, Stats and Metadata changes).

More detail can be seen in the [Profiling Data View](#), File System I/O Counts and selecting the process that you are interested.

5.5.3 CPU Load

This graph shows the mean CPU load per second over time. This is displayed as a percentage. If the process used multiple cores it is expected that this will be greater than 100% for CPU bound processes.

More detail can be seen in the [Profiling Data View](#), CPU Load and selecting the process that you are interested.

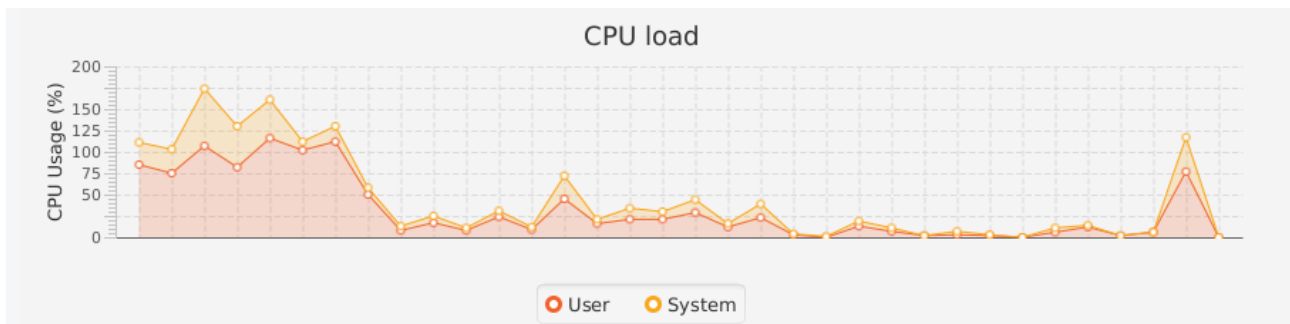


Figure 12: Program profile showing CPU load per second over time for the selected program

5.5.4 Memory Consumption

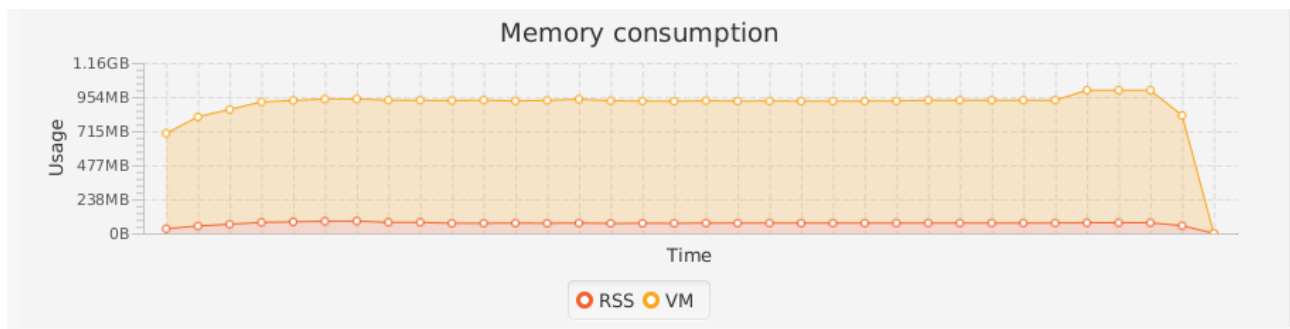


Figure 13: Program profile showing the memory usage (RSS and VM) per second over time for the selected program

This graph shows the memory usage over time. Split into Resident Set Size (RSS) and Virtual Memory (VM).

More detail can be seen in the [Profiling Data View](#), Memory Consumption and selecting the process that you are interested.

5.6 Node View

Once a process has been selected in either the [Timeline View](#) or [Duration View](#) the Node View will show details of all files and connections used by that process for all the link types configured in Trace > Preferences > Graph Nodes.

Only link types that are actually observed in the trace will be visible in the Node View and fork events will only be listed separately if the "Fold forks" option is not selected in Trace > Preferences > Graph Options.

Successful operations are listed separately from failed operations. The contents of the Node View are displayed in a standard collapsed tree format with each branch contents sorted alphabetically.

5.7 Event View

The Event View displays actions performed by a selected program or on a selected file. Selecting a program or file in the following views will automatically populate the Event View with the appropriate data.

To view the calls made by a program, click on that program in either the [Timeline View](#) or [Duration View](#). If the Event View is not currently visible either select the Event View tab, typically in the right hand panel, or choose Event View from the Views menu bar.

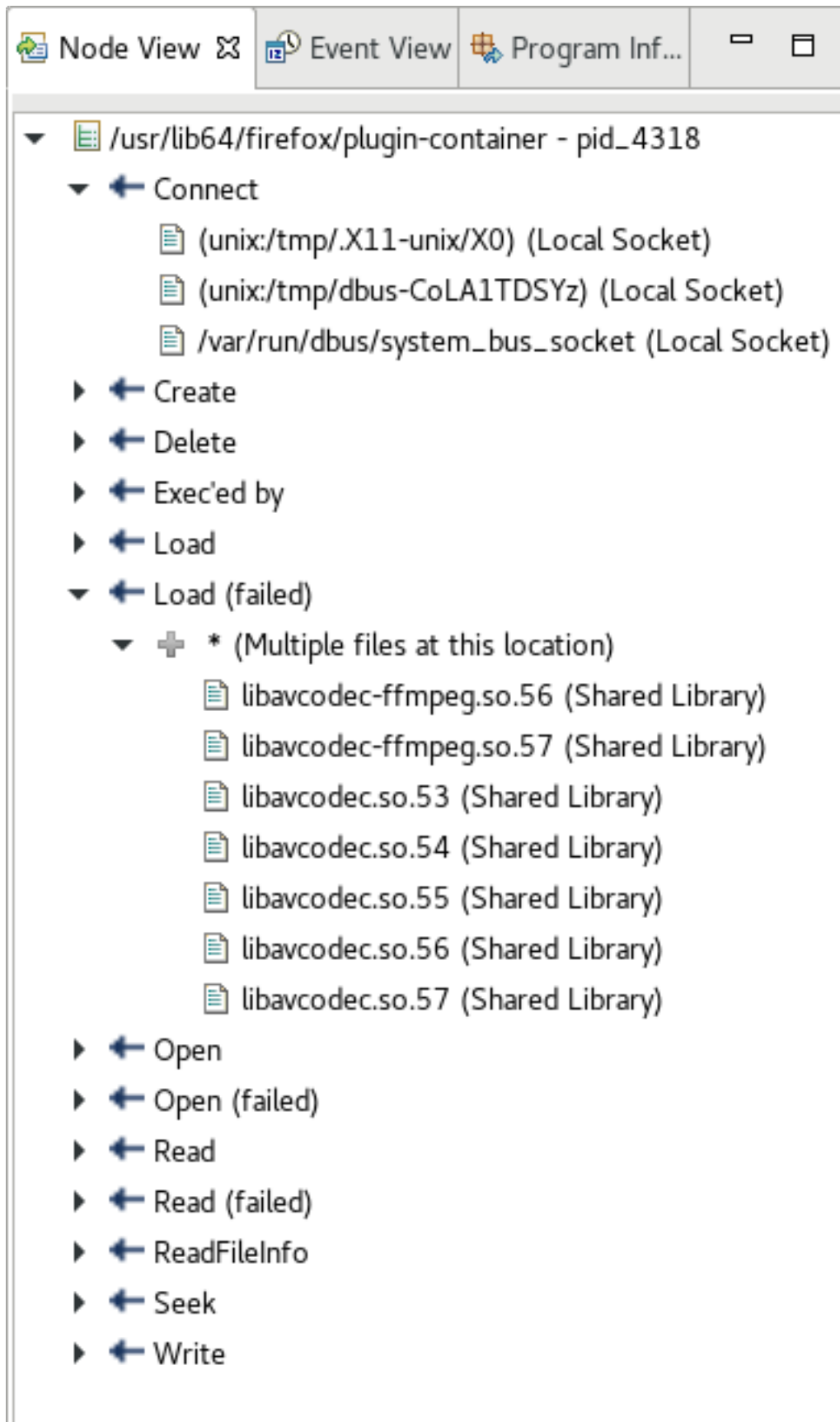


Figure 14: Node view

To view the actions performed on a file select the file in the **Files View**. The Event View will display the actions performed on the file and the programs that executed them.

The name of the selected program or file is displayed in a text box at the top of the Event View window.

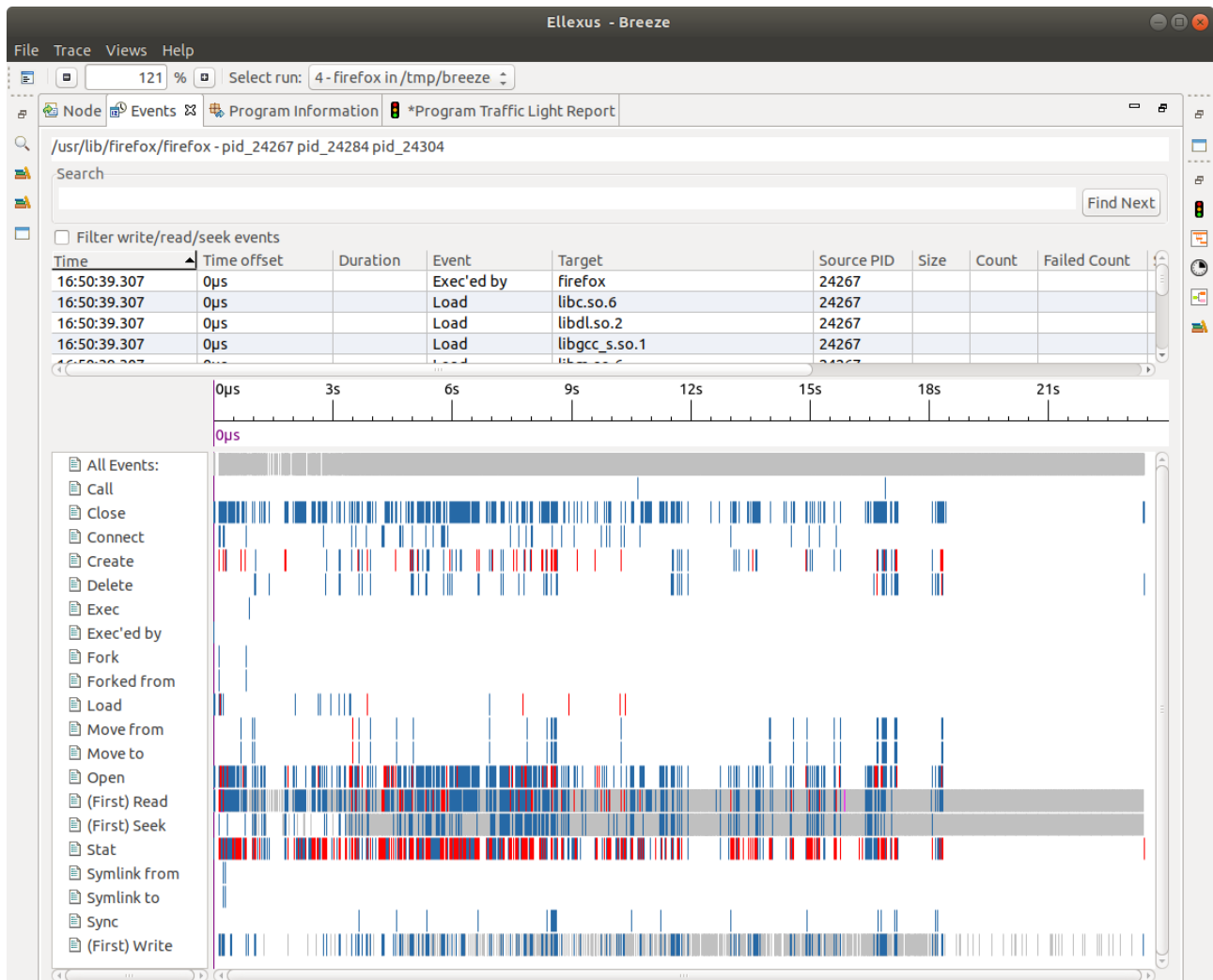


Figure 15: Events View

The Event view is composed two main parts. A table and a time-line graph. If a program has been selected, the table shows all calls made by that program. If a file has been selected, the table shows all accesses to that file. Below the table is a time-line graph, this is a visual summary of the information in the table. The table and graph are re-sizeable.

The table is initially sorted on the Time Offset column. Clicking on the header of the currently sorted column reverses the sort order and clicking on another column header will sort the table on that column.

The time-line graph initially scales to show all events in the space available, however it is possible to change the scale of the time-line by using the mouse wheel while holding 'ctrl' key. It is *not* affected by the zoom control on the application tool bar.

The search box can be used to search for any term in the table to skip to a particular event, action or time.

When an event is selected in the table, the time the selected event occurred is highlighted in the event time-line with a vertical purple bar. Similarly clicking on the time-line will select the nearest event to the selected time in the table.

Failed events are displayed in the table in a red italic font and the Failed? column will contain a red X icon. The time-line will display a red vertical bar.

If an event has an associated back trace then the table will show a green tick icon in the Backtrace column. Double clicking on the cell in the Backtrace column or selecting 'Show Backtrace' from the right-click pop-up menu will display the back trace in a dialog box.

Breeze records only the first read, write or seek of each file in detail so only these will be shown as blue or red bars. Subsequent reads and writes are aggregated and shown in grey in the table with the prefix (profile) and on the time line in grey or salmon.

All read, write and seek events can be filtered out of the view by checking "Filter write/read/seek events". If the trace was generated with the `--trace=all-io` option the "Filter write/read/seek events" check-box will be checked by default to maintain performance. Unchecking the filter option will add the omitted events to the view.

This view can be useful in identifying slow response times, for example a gap in the Event View time-line next to a connect call may indicate that a program is hanging on a particular network operation.

5.8 Program Information View

The Program Information View contains a summary of the environment used by the selected process. This view is displayed in the right pane by default and can also be opened as a pop-up window by double clicking on an orange program node in the [Timeline View](#).

The Program Information View is split into three tabs. The first tab "Arguments" shows how the process was invoked, including the complete command line used and any serious errors that were encountered.

The second tab "System Environment" provides details of all the Environment Variables set. By default this tab will show the complete environment as it was when this particular process was started but there are also options to only show Environment Variables that have been changed, either since the initial command traced or only those changed by the immediate parent process. This is useful to help track down issues where bad configuration is causing a workflow to fail.

The final tab "System Libraries" shows a list of all the shared libraries loaded by the process during the traced run. This is broken down into those libraries loaded at start-up, those loaded dynamically as the process was run and libraries that failed to load for some reason. Failed library loads can be examined in more detail using the [Event View](#).

5.9 Search View

Even if a program is not shown you can search for it in the Search View which is displayed in the left pane by default. You can click on a search result to see more information in the [Node View](#) and [Event View](#), or you can right click for more options.

5.9.1 Using the Search View to find files, libraries and network nodes

You can also use the Search View to locate particular files and network nodes. If you select a search result, the programs that used that resource will be listed in the [Node View](#) so it is easy to find out which files have been used by a particular program and which programs have used a particular file.

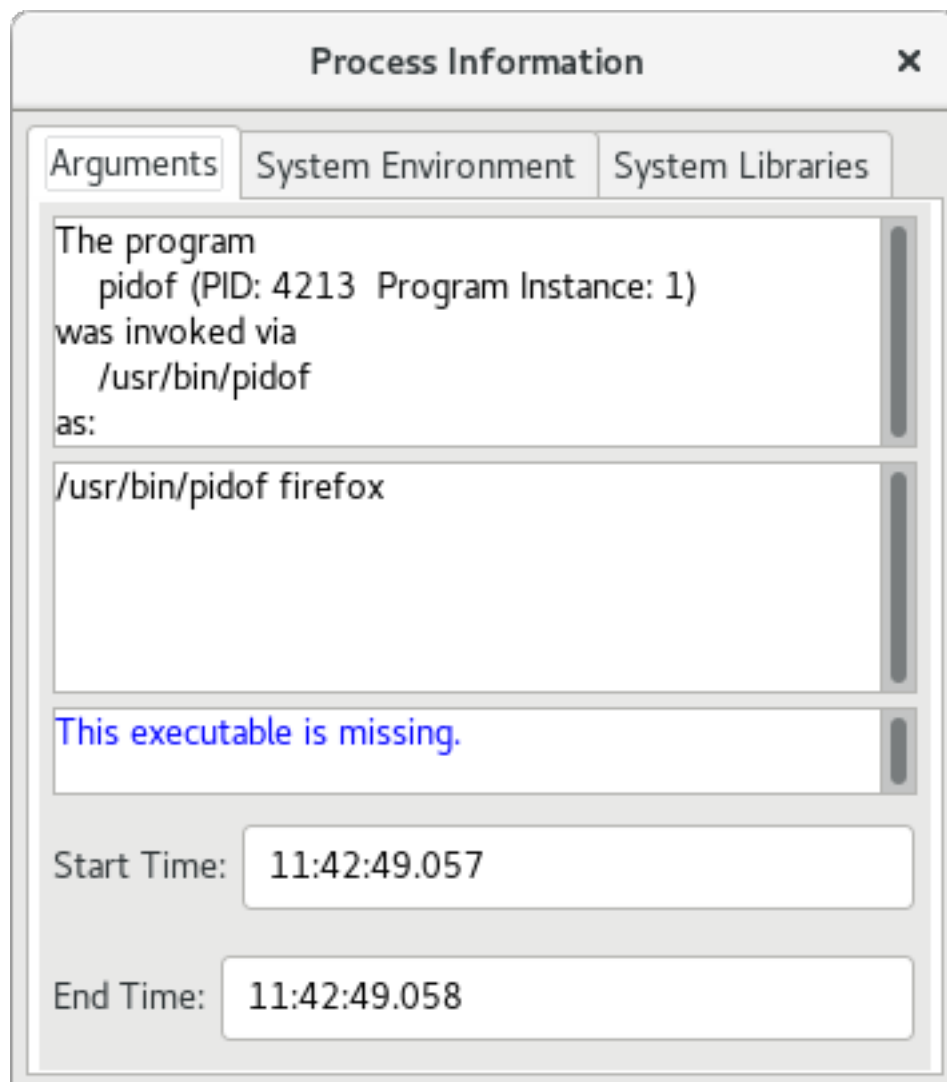


Figure 16: Process Information - Arguments

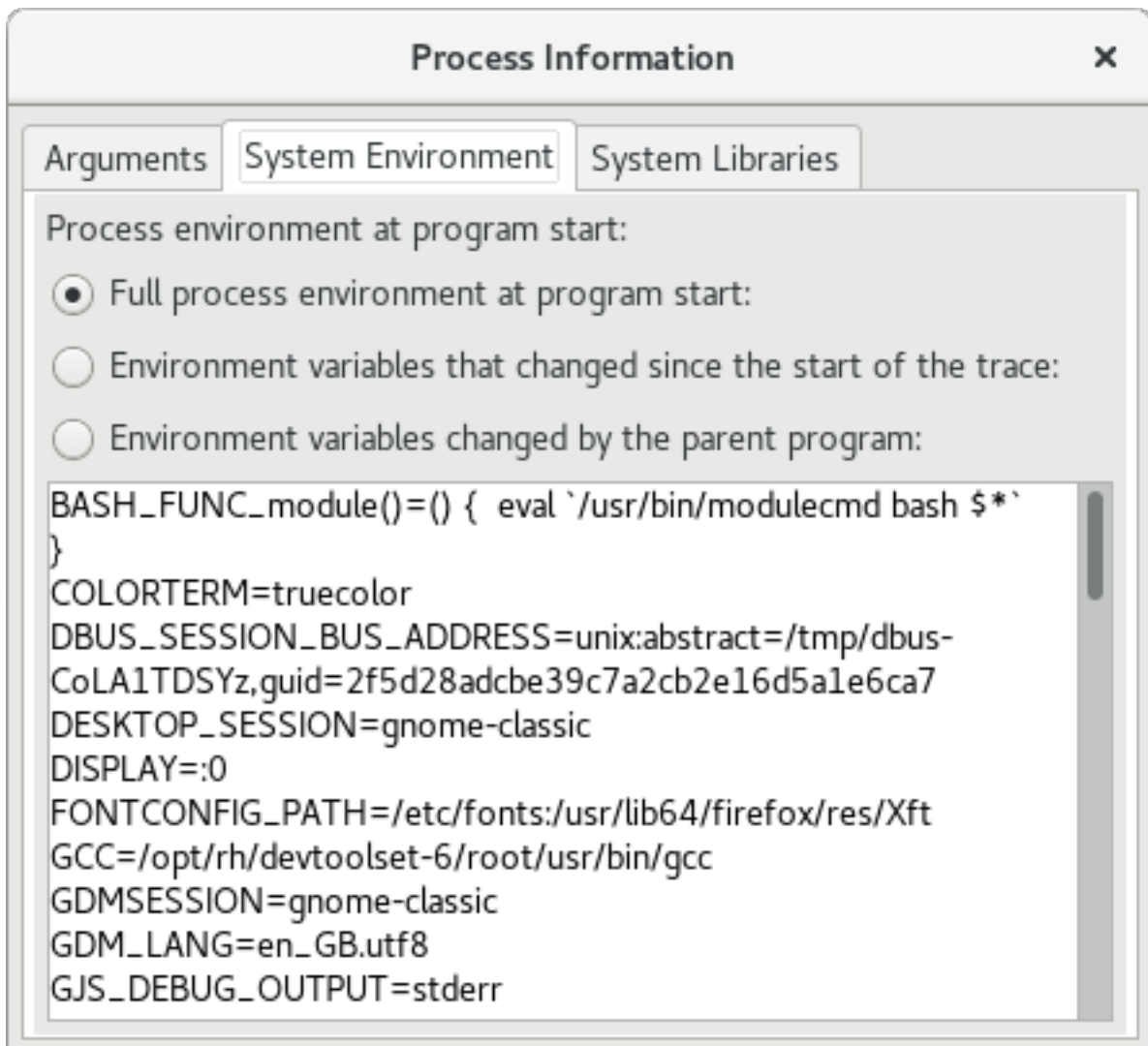


Figure 17: Process Information - System Environment

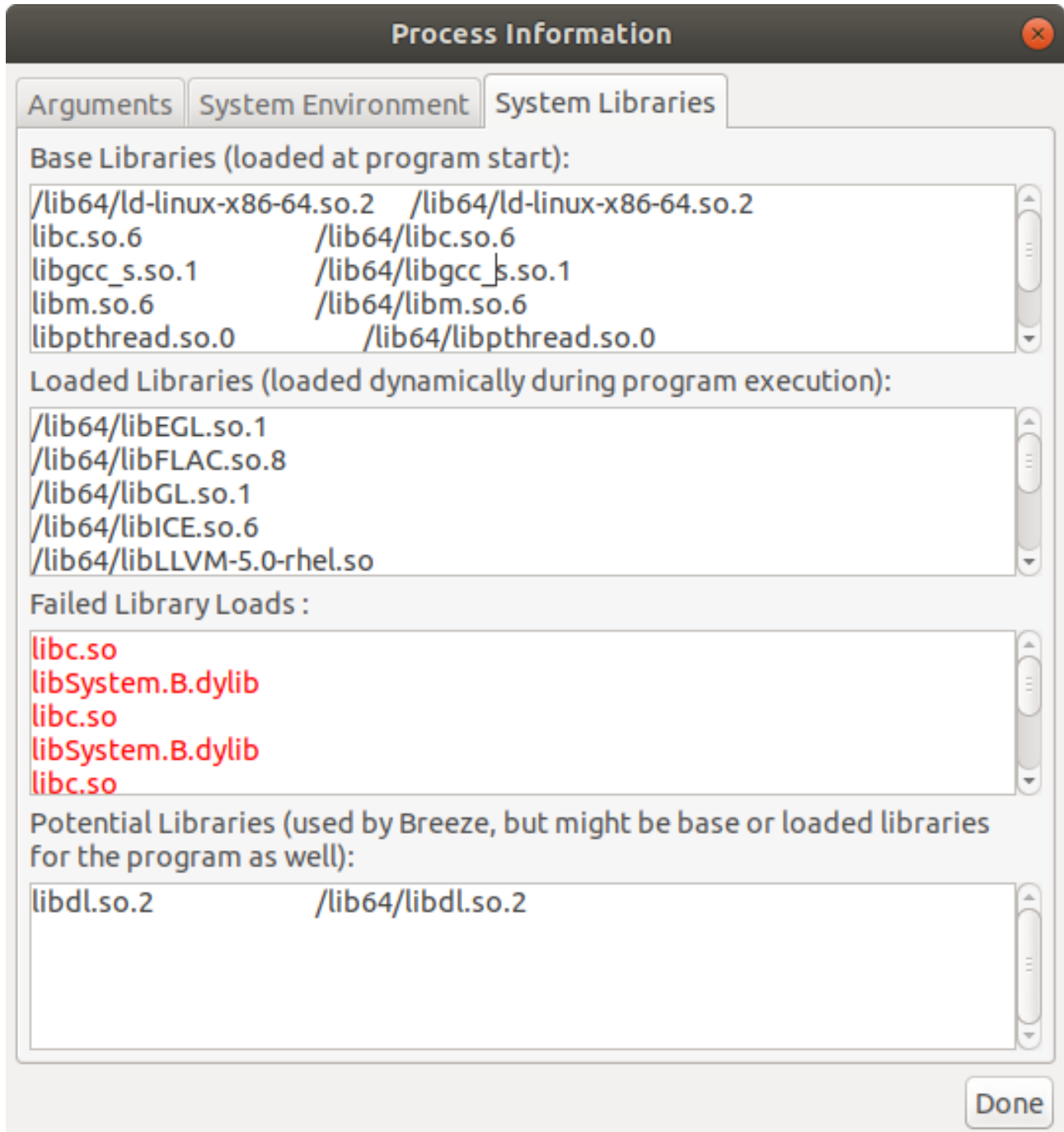


Figure 18: Process Information - System Libraries

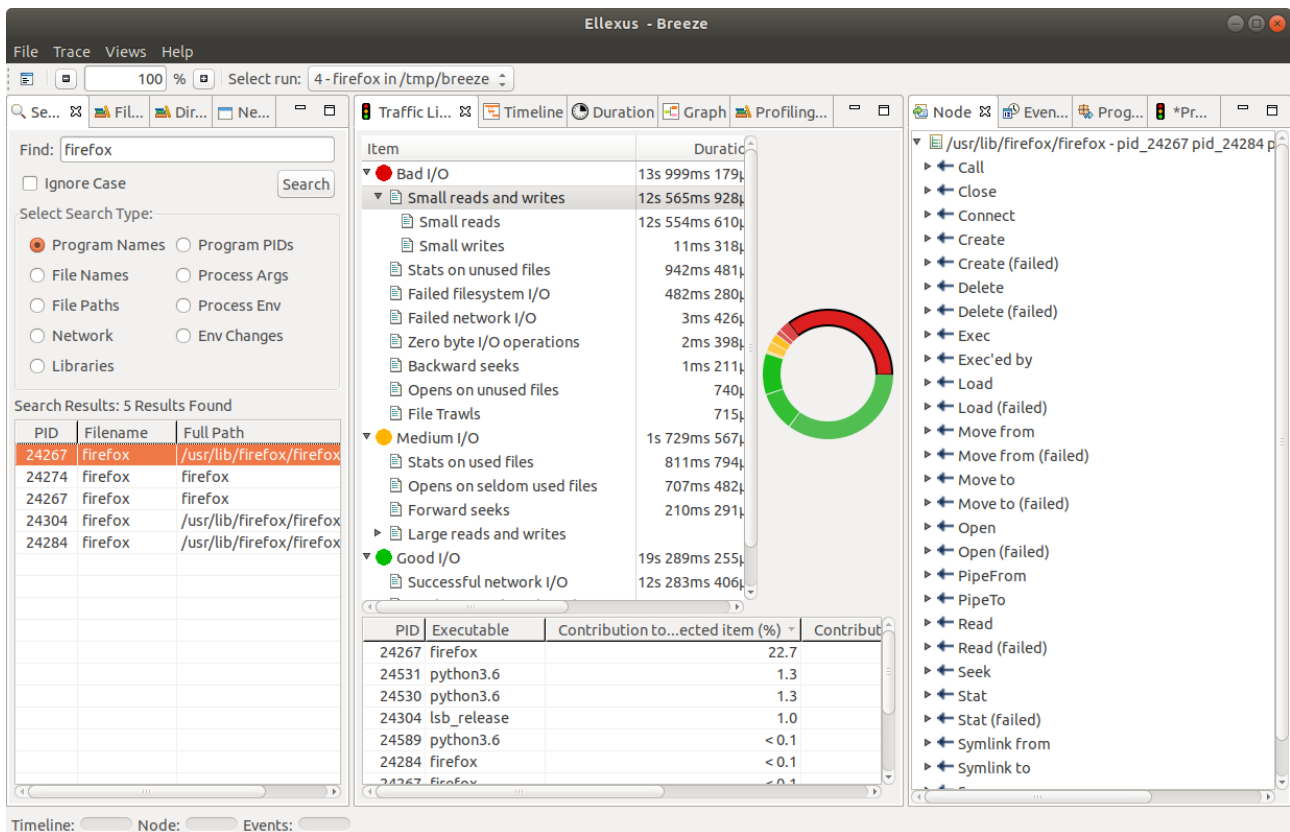


Figure 19: Search View

5.10 Files View

The Files View will show you a list of all the files used by your application and is displayed in the left pane by default. This can help you to see which files need to be shipped with your application, or to see whether any illegal or user-specific files have been used by mistake.

The table has a double header row, the first row represents a group of related information that can be expanded by double-clicking on the group header text.

The entire table can be sorted by clicking on the appropriate second level column header regardless of whether the group is expanded or not. Expanded groups can be re-collapsed by double-clicking the group header.

5.10.1 Group By

Initially the view will display summarised data on the number of various I/O operations carried out on each file. The data can be grouped as follows:

5.10.1.1 File

The data in Breeze is collected per file and this shows all of the data. This is particularly useful for dependency analysis or to help find bottlenecks.

5.10.1.2 Directory

All the operations on the files inside a directory are aggregated. This can be useful for applications that work on a large number of files per directory (e.g. databases).

Location	Read	Small Read	Write	Small Write	Seek	Open	Create	Stat	Delete	mmap	Sync	Load	Large Read	Large Write	Small Seek
# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call	# Call
/usr/bin/	2,222	2,221	0	0	0	0	1	0	31	0	0	0	0	0	0
/home/richard/.mozilla/firefox/j386tj5b.default/...	1,140	1,140	22	22	1,162	3	0	56	0	0	0	2	0	0	445
/home/richard/.mozilla/firefox/j386tj5b.default/...	1,004	996	0	0	1,004	1	0	1,002	0	0	0	0	0	0	996
/proc/21467/mountinfo	922	922	0	0	0	0	0	0	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	838	670	0	0	4	2	0	12	0	0	0	0	0	0	1
/proc/21467/smaps	508	508	0	0	0	0	0	0	0	0	0	0	0	0	0
/dev/	460	458	0	0	0	2	0	0	0	0	0	0	0	0	0
/usr/lib/python3/dist-packages/	460	458	0	0	0	2	0	6	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	357	350	0	0	0	7	0	2	0	0	0	0	0	0	0
anon_inode[eventfd]	243	243	406	405	0	0	0	0	0	0	0	0	0	0	0
/usr/lib/python3.8/	206	205	0	0	0	1	0	44	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	227	182	0	0	4	2	0	12	0	0	0	0	0	0	1
/usr/share/applications/	153	152	0	0	0	1	0	1	0	0	0	0	0	0	0
/usr/lib/python3.8/encodings/	127	126	0	0	0	1	0	5	0	0	0	0	0	0	0
/etc/fonts/conf.d/	107	106	0	0	0	1	0	9	0	0	0	0	0	0	0
/etc/fonts/conf.avail/	96	95	0	0	0	1	0	4	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	91	84	0	0	0	7	0	27	0	0	0	0	0	0	0
/proc/21467/maps	66	66	0	0	0	0	0	0	0	0	0	0	0	0	0
/home/richard/.mozilla/firefox/j386tj5b.default/...	57	56	0	0	0	1	0	0	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	51	50	0	0	0	1	1	100	1	0	0	0	0	0	0
/usr/lib/python3.8/lib-dynload/	48	47	0	0	0	1	0	6	0	0	0	0	0	0	0
/home/richard/.mozilla/firefox/j386tj5b.default/...	48	46	0	0	0	2	0	0	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	58	45	0	0	4	2	0	12	0	0	0	0	0	0	1
/home/richard/.mozilla/firefox/j386tj5b.default/...	44	38	0	0	44	1	0	44	0	0	0	0	0	0	36
/dev/dri/	42	35	0	0	0	7	0	2	0	0	0	0	0	0	0
/usr/share/drirc.d/00-mesa-defaults.conf	40	35	0	0	0	5	0	0	0	0	0	0	0	0	0
/usr/share/X11/locale/locale.alias	35	34	0	0	0	2	0	0	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	38	29	0	0	4	2	0	12	0	0	0	0	0	0	1
/home/richard/.mozilla/firefox/j386tj5b.default/...	25	25	13	13	38	3	0	65	0	0	0	2	0	0	36
/home/richard/.mozilla/firefox/j386tj5b.default/...	25	25	46	46	71	3	3	9	0	0	4	0	0	0	65
/usr/share/pixmaps/	26	25	0	0	0	1	0	2	0	0	0	0	0	0	0
/usr/share/hunspell/	26	24	0	0	0	2	0	4	0	0	0	0	0	0	0
/home/richard/.cache/event-sound-cache.tdb.30...	23	23	0	0	22	1	0	2	0	0	0	0	0	0	22
/usr/share/icons/	23	22	0	0	0	1	0	2	0	0	0	0	0	0	0
/home/richard/.mozilla/firefox/j386tj5b.default/...	22	21	0	0	0	1	1	0	0	0	0	0	0	0	0
/etc/drirc	25	20	0	0	0	5	0	0	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	20	18	0	0	0	2	1	4	0	0	0	0	0	0	0
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	15	15	0	0	15	3	0	52	0	0	0	0	0	0	15
/home/richard/.cache/mozilla/firefox/j386tj5b.d...	18	14	0	0	18	1	0	23	0	0	0	0	0	0	13
/home/richard/.mozilla/firefox/j386tj5b.default/...	15	14	0	0	0	4	0	7	0	0	3	0	0	0	0
Total	13,233	12,100	3,413	2,892	4,973	3,269	1,135	10,620	795	956	69	183	0	0	3,916

Figure 20: Files View

5.10.1.3 Mount Point

All the operations on the files on a mount point are aggregated. This can be useful to check that your application used the right areas of the file-system, e.g. local vs. shared storage.

5.10.2 Summary Column

By default the groups of columns are collapsed, and only the # Call column is visible for each group. If you wish to see a different column without expanding each group you can choose this from the drop down. The options are:

- # Call
- Amount
- Total Latency
- Max Latency
- # Failures

See below for the definitions of these columns.

5.10.3 Filter By Path

You can filter which files/directories/mount points are shown in the Files View using the Filter By Path (supports regular expressions) box. The full path will be filtered using the regular expression that you enter.

Text without special characters will match any part of the filename. Should you wish to match only the start of file path start your filter with ^.

e.g.

/bin/ - will match all files in both /bin and /usr/bin
 ^/bin/ - will match all files in only /bin

5.10.4 Location Group

The Location column group gives information about the file this row's data is related to. When collapsed the group displays only base file names. When expanded this group displays the mountpoint containing the file, the file type, file basename and full file specification.

5.10.5 Read Group

The Read column group gives you information about all the read calls performed against the specified file. When collapsed the group only displays the number of read calls made to this file. When expanded this group displays the raw number of read calls made on the file, the total number of Bytes read, the total latency of these calls in μ s, the maximum latency of a single read call and the total number of failed calls.

5.10.6 Small Read Group

Small (< 32kB) Read			
# Call	Amount (B)	Total Latency (μ s)	Max Latency (μ s)

Figure 21: Files View - Small Read Group

The Small Read column group gives you the same information as the **Read Group** but only reports information relating to reads that were smaller than 32kBytes.

When collapsed the group only displays the number of read calls made to this file. When expanded this group displays the raw number of read calls made on the file, the total number of Bytes read, the total latency of these calls in μ s, and the maximum latency of a single read call. The count of failed read calls is not displayed as read size is recorded from the number of bytes successfully read which is, by definition, not available for a failed call.

5.10.7 Large Read Group

Large (\geq 100MB) Read			
# Call	Amount (B)	Total Latency (μ s)	Max Latency (μ s)

Figure 22: Files View - Large Read Group

The Small Read column group gives you the same information as the **Read Group** but only reports information relating to reads that were larger than 100MBytes.

5.10.8 Write Group

The Write column group gives you information about all the write calls performed against the specified file. When collapsed the group only displays the number of write calls made to this file. When expanded this group displays the raw number of write calls made on the file, the total number of Bytes written, the total latency of these calls in μ s, the maximum latency of a single write call and the total number of failed calls.

5.10.9 Small Write Group

Small (< 32kB) Write			
# Call	Amount (B)	Total Latency (µs)	Max Latency (µs)

Figure 23: Files View - Small Write Group

The Small Write column group gives you the same information as the **Write Group** but only reports information relating to write that were smaller than 32kBytes.

When collapsed the group only displays the number of write calls made to this file. When expanded this group displays the raw number of write calls made on the file, the total number of Bytes written, the total latency of these calls in µs, and the maximum latency of a single write call. The count of failed write calls is not displayed as read size is recorded from the number of bytes successfully written which is, by definition, not available for a failed call.

5.10.10 Large Write Group

Large (≥ 100MB) Write			
# Call	Amount (B)	Total Latency (µs)	Max Latency (µs)

Figure 24: Files View - Large Write Group

The Large Write column group gives you the same information as the **Write Group** but only reports information relating to write that were larger than 100MBytes.

5.10.11 Seek Group

The Seek column group lists information about all the seek calls performed against the specified file. When collapsed the group only displays the number of seek calls made within this file. When expanded this group displays the raw number of seek calls made within the file, the total seek distance in Bytes, the total latency of these calls in µs, the maximum latency of a single seek call and the total number of failed calls.

5.10.12 Small Seek Group

Small (< 32kB) Seek			
# Call	Distance (B)	Total Latency (µs)	Max Latency (µs)

Figure 25: Files View - Small Seek Group

The Small Seek column group gives you the same information as the **Seek Group** but only reports information relating to seeks that moved less than 32kBytes within the file.

When collapsed the group only displays the number of seek calls made within this file. When expanded this group displays the raw number of seek calls made within the file, the total seek distance in Bytes, the total latency of these calls in µs, and the maximum latency of a single seek call. The count of failed seek calls is not displayed as seek size is calculated as the difference of positions within the file after the seek has completed successfully which, by definition, is not available for a failed call.

5.10.13 Other Operation

The Other Operation column group gives you information non I/O operations that targeted this file. When collapsed the group only displays the total number of non-I/O operations performed on this file. When expanded this group displays the total number of Open, Create, Load, Stat and Delete calls, with each category spilt into counts of successful and failed calls as well as their latencies. In addition, if any seek operations were performed the read/seek ration is calculated and displayed to give an indication of how random read access was to the file.

5.11 Program Files View

Shows the same data as the files view, but only for the selected program.

5.12 Network View

You can also get an overview of the I/O that was done over the network with the Network View. This view is displayed in the left pane but is not opened automatically by default. The view can be displayed by selecting Views > Network View from the drop down menus.

The data shown in the Network View lists information about connections to specific network locations broken down by IP address and port number. Where the connection is initiated on the execution host detail for all local ephemeral ports is aggregated, for example in the screenshot below all the connections from the execution host with IP address 10.33.100.166 to the DNS server on 10.33.0.40 port 53 are listed as a single line even though each unique connection will have been initiated with a different local port number. Similarly, inbound connection data from remote ephemeral ports will be aggregated into a single line.

Location	Accept	Bind	Connect	Listen	Read	Write
Address	# Call	# Call	# Call	# Call	# Call	# Call
10.33.100.84 -> 64.233.167.189:443	0	0	1	0	14	2
10.33.100.84 -> 216.58.206.110:80	0	0	5	0	10	10
10.33.100.84 -> 104.86.110.209:80	0	0	1	0	1	1
10.33.100.84 -> 192.0.76.3:443	0	0	2	0	45	12
::1:35663	0	1	0	0	0	0
10.33.100.84 -> 172.217.23.10:443	0	0	1	0	16	4
10.33.100.84 -> 52.26.196.116:443	0	0	1	0	14	4
10.33.100.84 -> 209.132.180.180:443	0	0	1	0	22	3
::1:43371	0	1	0	0	0	0
10.33.100.84 -> 93.184.220.29:80	0	0	1	0	2	2
10.33.100.84 -> 216.58.206.74:443	0	0	2	0	50	17
10.33.100.84 -> 143.204.95.106:443	0	0	1	0	18	4
10.33.100.84 -> 46.17.166.12:443	0	0	11	0	405	48
10.33.100.84 -> 216.58.206.101:443	0	0	2	0	280	22
10.33.100.84 -> 46.235.225.189:443	0	0	7	0	216	38
10.33.100.84 -> 216.58.206.97:443	0	0	1	0	24	8
10.33.100.84 -> 23.205.220.128:80	0	0	2	0	3	3
::1:53621	0	1	0	0	0	0
10.33.100.84 -> 23.205.220.33:80	0	0	1	0	1	1
10.33.100.84 -> 93.93.129.174:443	0	0	5	0	281	45
10.33.100.84 -> 192.0.77.32:443	0	0	1	0	27	7
10.33.100.84 -> 216.58.206.110:443	0	0	7	0	1646	94

Figure 26: Network view

Initially the view will display summarised data on various network connections established by this job. The table has a double header row, the first row represents a group of related information that can be expanded by double-clicking on the group header.

The entire table can be sorted by clicking on the appropriate second level column header regardless of whether the group is expanded or not. Expanded groups can be re-collapsed by double-clicking the group header.

Each group expands as follows.:

5.12.1 Location Group

Location ⌵			
Local Address	Local Port	Remote Address	Remote Port

Figure 27: Network view - location group

The Location column group gives information about the connection this row's data is related to. When collapsed the group displays a string representation of all the data. When expanded this group displays the local IP address and port, and remote IP address and port as separate columns for more precise sorting options.

For outbound connections the local port field may be empty when several connections using different ephemeral high port numbers have been aggregated. Similarly the remote port field may be empty for inbound connections.

5.12.2 Accept, Bind, Connect and Listen Groups

The four column groups labelled Accept, bind, Connect and Listen contain the same column headings for their respective functions.

When collapsed each group only displays the number of the related function calls made on the specified connection. When expanded the group displays the raw number of related function calls made on the connection, the total latency of these calls in μ s, the maximum latency of a single call and the total number of failed calls.

5.12.3 Read and Write Groups

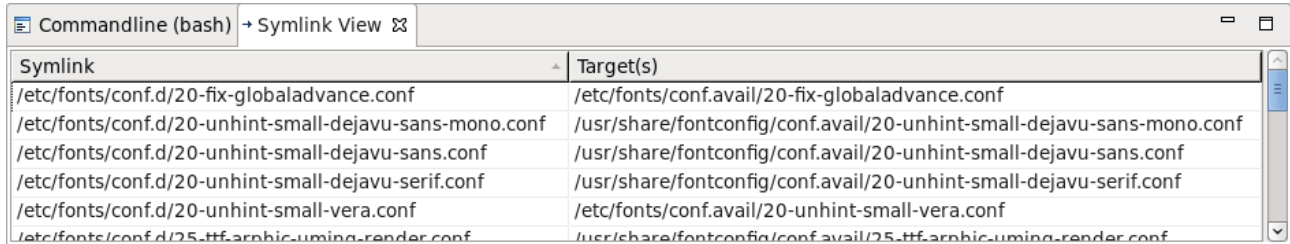
The Read and Write column groups gives you information about all the read and write calls performed against the specified connection. When collapsed the group only displays the number of the respective function calls made on this connection. When expanded this group displays the raw number of related function calls made on the connection, the total number of Bytes read or written, the total latency of these calls in μ s, the maximum latency of a single write call and the total number of failed calls.

5.13 Program Network View

Shows the same data as the network view, but only for the selected program.

5.14 Symlink View

The Symlink View (opened from the Views Menu) will show you what symlinks are opened or read by your application. You will get the paths of the links on the left column and the paths of the targets on the right column. In some cases, Breeze might not be aware of the target of the link, in which case it will be clearly indicated. If your application uses the same link for different targets over time, you will see one line per target.



Symlink	Target(s)
/etc/fonts/conf.d/20-fix-globaladvance.conf	/etc/fonts/conf.avail/20-fix-globaladvance.conf
/etc/fonts/conf.d/20-unhint-small-dejavu-sans-mono.conf	/usr/share/fontconfig/conf.avail/20-unhint-small-dejavu-sans-mono.conf
/etc/fonts/conf.d/20-unhint-small-dejavu-sans.conf	/usr/share/fontconfig/conf.avail/20-unhint-small-dejavu-sans.conf
/etc/fonts/conf.d/20-unhint-small-dejavu-serif.conf	/usr/share/fontconfig/conf.avail/20-unhint-small-dejavu-serif.conf
/etc/fonts/conf.d/20-unhint-small-vera.conf	/etc/fonts/conf.avail/20-unhint-small-vera.conf
/etc/fonts/conf.d/25-ttf-arphic-uming-render.conf	/usr/share/fontconfig/conf.avail/25-ttf-arphic-uming-render.conf

Figure 28: Symlink view

6 Profiling file I/O

We are now going to look at profiling file system and network I/O. This will help you to find performance bottlenecks and improve the scalability of your application. Breeze profiling data can help you detect when your application is storing files in the wrong place or accessing your shared file system inefficiently. Breeze will also tell you what kind of performance your application is seeing from your storage infrastructure.

The profiling data is displayed by default, but can also be accessed via Views > Profiling Data View.

In each view, there is a “Show in Duration View” button, which will switch to the Duration View and set the selected time as filter. This is very useful to have a better understanding of which programs were active at that specific time and contributed in the statistics.

When showing profile data for a specific PID and switching to the Duration View via the available button, it might happen that the table does not show this specific PID. This happens when the option “Fold forks to combine multi-process programs into one node” is set, so the program that triggered the profile log is folded in another node. To have a full overview of all the programs, the option mentioned above could be deselected.

6.1 File-system mount points

Breeze collects statistics per mount point, along with a catch-all which displays the combined data across all mount points.

6.2 File-system I/O Counts

This option tells Breeze to count the number of times the file system is accessed. Breeze will measure the number of open, seek, read or write calls, as well as the number of bytes read or written and the seek distance.

Other operations recorded are stat calls which check permissions of files, and file system changes such as move or delete. The counts are reported every second.

6.3 File-system I/O Latency

Breeze can measure the time taken to access the file system. It will collect separate statistics for each of the file-system mount point.

The first two columns show the minimum and maximum latency in microseconds. The next column headings are the latency ranges. The table contains counts for each of these ranges.

6.4 Network I/O Count and Latency

These options provide the same sort of information as the file-system I/O counts and latency, but for the network. Breeze will record information for every network location.

6.5 Error Code Counts and Zero-Byte I/O Counts

Breeze can show you any errors returned by I/O calls and the number of zero byte reads and writes. Some errors and zero-byte operations are expected as they are used by programs to understand permissions and to test the

existence of files. Lots of errors, or unusual errors, may indicate a more serious problem. Each error value (errno) is counted separately, so you can diagnose any network or file system problems you are having.

6.6 Memory Consumption

If enabled, Breeze will report the memory usage of the program being profiled. This records the virtual memory (VM) and resident set size (RSS) as an approximation for how much memory is used. The total memory used by all processes is not available for this measure as the use of shared memory can significantly affect the result, so instead you must specify the process ID of the program you want to see the data for.

6.7 Job CPU Usage

This mode shows you various information about how CPU is stressed by your application. Displayed information includes User and System load, and voluntary and involuntary context switches.

6.8 Host CPU Usage

This mode shows you how the compute host was performing whilst your application was running. Displayed information is the User, Nice and System load as well as the time spent in I/O wait.

The Nice figure is the percentage time the CPU spent running user processes marked as nice (i.e. low priority). This is the same figure that `top` refers to as `ni`.

The I/O wait figure is the percentage time the CPU spent idle whilst waiting for I/O operations to complete, this is the same figure that `top` refers to as `wa`.

The Steal time figure is the percentage time the CPU was made unavailable to the operating system. This normally only makes sense for virtual machines, this is the same figure that `top` refers to as `st`. If you are running on a cloud instance it may be that your virtual machine was running out of CPU credits if this percentage is high.

6.9 Symlink Chain Counts

A long chain of symbolic links could impact on performance. If this option is enabled, each file opened will be checked to see if it is in a chain of symbolic links.

6.10 File-system Trawls

A program will often look in a number of directories to find a particular file or program, and in the worst case it may search the entire file system before it finds it. For example, a program might attempt to access a file within each directory on the `PATH` until it eventually finds the file it is looking for. On distributed file systems this can affect not only the performance of that program, but also the performance of all the other programs running on the cluster. We refer to such a sequence of calls as a 'trawl'.

In more detail, if Breeze sees a specific file system call fail at least 4 times, without any other intervening calls, it considers this to be a 'trawl'. The trawl is ended either by a successful call of the same type, or a different call. Breeze records the number of failed calls in the trawl, the name of the file associated with the final failed call, and the total time taken by the whole sequence of failed calls.

6.11 I/O Sizing

This option provides information about file-system I/O counts, operation size and latency of read,write and seek operations by mount point for different operation size ranges over time.

This allows you to identify the size of operations over time in your job. This will help identify jobs that would benefit from internal caching to increase operation size – and can be used as a measure of how the filesystem is performing from the applications view point.

The size ranges used are

4B, 16B, 64B, 256B, 1kB, 4kB, 16kB, 64kB, 256kB, 1MB, 4MB, 16MB, 64MB, 256MB, 1GB

7 Guide to MPI Applications

Breeze can profile MPI applications in the same way as it profiles single-node applications. Breeze will by default automatically wrap each rank with the profiling technology. In Breeze multiple ranks on a single machine will be aggregated into one trace. MPI ranks that run on other machines will have their own traces, grouped by machine. So if an MPI application has 32 ranks on two machines, Breeze will create one top level trace and one child trace, and each trace will have 32 ranks. Each rank is displayed as a separate process.

In Breeze you can load child traces using the drop down menu at the top of the UI/.

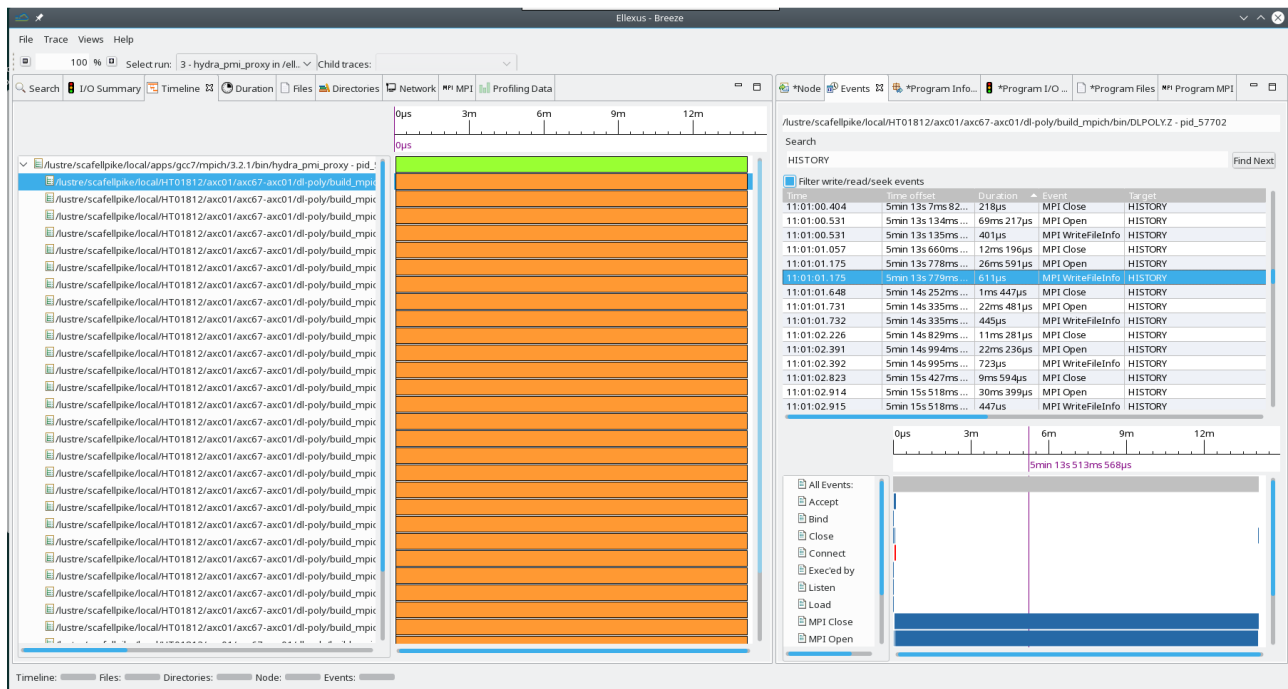


Figure 29: An example of the Timeline view with an MPI application

7.1 Aggregating I/O data for the whole job

Traces cannot be aggregated in the Breeze GUI, but the data from multiple traces can be aggregated with the following options in Breeze Automation Platform:

```
# All the file dependencies for the whole application (XML format)
$ breezeAP.sh <trace_path> -k -y dependencies -f xml -o <output_dir>
# Summary of all files used and their access counts (CSV format)
$ breezeAP.sh <trace_path> -k -y files -f csv -o <output_dir>
# Summary of all files used and their access counts, the MPI and
# network operations (XML format)
$ breezeAP.sh <trace_path> -k -y files,mpi,network -f xml -o <output_dir>
```

7.2 My job ran on 1000 hosts, which trace file should I look at?

The largest trace file will be for the ranks that did the most I/O. If the traces are all the same size then it is likely that I/O is evenly distributed across ranks and you can open any trace to see a representative sample of the I/O patterns.

7.3 Viewing MPI file I/O in Breeze

MPI file I/O calls eventually result in POSIX file I/O, which Breeze displays in the usual way.

7.4 Tracking MPI comms I/O between ranks in Breeze

MPI comms I/O between ranks in a single MPI job will show up in Breeze as Network I/O, shown in the Network View and Profile Data View.

8 Breeze Automation Platform and Exporting Data

The Breeze Automation Platform offers a full solution to integrate Breeze into your compute infrastructure. It enables you to convert trace data straight into XML and CSV output for automatic processing. This allows run checks over thousands of traced programs.

You can run automatic checks on your IT set-up to be assured of high-quality configuration and long-term system compliance. The Breeze Automation Platform gives you full access to tracing and profiling data so you can write your own custom queries. By integrating the Breeze Automation Platform into your working practices, you can resolve issues before they become problems.

For more information about how to run the Breeze Automation Platform, simply run `breezeAP.sh` with no arguments.

8.1 Breeze Automation Platform Licensing

Before you can run Breeze Automation Platform you must have a valid license set up, please see `breeze_installation.pdf` for details.

8.2 Breeze AP Command Line Options

The following section lists all the valid command line options used by `breezeAP.sh`. You can also see all the options offered by `breezeAP.sh` by running the script with no options.

```
breezeAP.sh <input trace> [<input trace>]... --all [-t <temp dir>]
               [-o <output dir>] [-c <preferences file>] [-p <profile data type>]
               [--licqueue] [--mem <size>]
```

```
breezeAP.sh <input trace> [<input trace>]... -y <output types>
               -f <output formats> [-t <temp dir>] [-o <output dir>]
               [-c <preferences file>] [-p <profile data type>] [-h <home dirs>]
               [-r <restricted_dirs>] [--licqueue] [--mem <size>]
```

The first synopsis is used to parse the input trace(s) and output trace data in all formats.

The second synopsis allows you to parse the input trace(s) and output trace data for each of the type options you provided with the '-y' option. The output format for each type must be provided using the '-f' option.

In both cases, `<input trace>` denotes a folder containing the trace (comprised of multiple files). `<input trace>` can be a glob pattern with the following special characters:

- '*' matches zero or more characters in a filename;
- '{}' specifies a collection of sub-patterns, e.g. '{one,two,three}' matches either 'one', 'two' or 'three';
- '[]' matches any character between the brackets, e.g. '[one]' matches either 'o', 'n' or 'e'. You can specify ranges with '-' and negate things with '!';
- '?' matches exactly one character;
- '\' suppresses the special nature of the immediately following special character, causing it to be interpreted as a literal character and part of the file name. To specify '\' itself as a literal character use '\\'.

It is recommended that the file pattern is placed within quotes to prevent shell expansion.

If an output directory is not specified with the '-o' option, the output files will be written in a directory called 'breezeap' and located under <input trace>. This means that if you don't have write permission on <input trace>, you must use the '-o' option to specify an alternate output directory. The name of the output file is generated from the name of <input file> and the output type and format, e.g. input.trace-files.xml.

When several traces are specified on the command line, the '-o' option is required to specify an output directory. One common file will be produced for output types that support data aggregation (currently: dependencies, files, mpi and network), and it will be named after the output type and format (e.g. files.xml). For other output types, one file per trace will be produced, and their names will be derived from the corresponding input file and the output type and format (e.g. input.trace-programs.xml).

OPTIONS:

-a
--all

This options produces every possible output in every compatible format. If your trace(s) contain(s) profile data, it will be output.

--output-type=<output types>
-y <output types>

This option allows you to specify which type of output you want. <output types> must be a comma-separated list of acceptable type. The acceptable types are:

- **io-summary**: a summary of the trace including the I/O duration, I/O count and size statistics. There will be one line for each trace file;
- **files**: a list of file dependencies with I/O statistics for each file;
- **network**: a list of network I/O statistics;
- **programs**: a list of programs with program information and actions;
- **events**: a list of programs each with a list of actions ordered by time;
- **profile**: exports all of the profiling statistics collected by Breeze. These are: I/O counts and latency for both file and network I/O, error counts, zero byte I/O, file-system trawls, memory consumption symlink chain counts, host CPU and job CPU;
- **dependencies**: a list of file dependencies with no additional information. This mode is faster than the other export options when used by itself;
- **symlink**: a list of symlinks and their target(s);
- **duration**: a list of programs with the duration they were running for and summary I/O statistics;
- **healthcheck**: a healthcheck report with detailed statistics on I/O and dependencies.

Each type can appear several times if several output formats are needed.

```
--output-format=<output formats>
-f <output formats>
```

This option lets you specify which output format you want for each type of output given with the '-y' option. <output formats> must be a comma-separated list of acceptable format. Acceptable formats are:

- text, txt: human readable text, for events and healthcheck types;
- xml: XML
- xsd: XML schema, for files, network, programs, events, dependencies and mpi types;
- csv: comma-separated file, for files, network, profile, dependencies and mpi types;
- html: HTML, for healthcheck.

For a given output, position of its type in <output types> and position of its format in <output formats> must be the same.

```
--tmp=<temporary directory>
-t <temporary directory>
```

This specifies where Breeze will store its temporary files. These can be quite large as they contain post-processed trace information. By default Breeze will make a temporary directory in /tmp, but if you have insufficient storage there you may want to use this option to specify an alternative location.

```
--preferences-file=<preferences file>
-c <preferences file>
```

Use this preferences file instead of the default settings. If this option is not used Breeze will search for a file called breeze-preferences.properties in your home directory followed by the Breeze installation directory. If the file isn't found BreezeAP will use its internal default values.

```
--output-directory=<output directory>
-o <output directory>
```

An output directory may be specified for the output files. This directory must be writable if it exists. If it is not specified, the output files will be written in a directory called breezeap, inside <input trace>. This option is mandatory when specifying several traces on the command line.

```
--profile-data-type=<all | total | separate>
-p <all | total | separate>
```

When outputting profile data, you can choose to export just the 'total' for all PIDs in the run, a 'separate' file for each PID, or 'all' data with both total and separate PIDs. Breeze will output 'total' by default.

```
--child-traces
-k
```


This option allows you to include all the data from remote traces in a combined trace output, as if it was part of the main trace. This means that processes run on different host machines would be included as if they were run on the initial machine. This option is only supported for the following output types: io-summary, dependencies, files, mpi and network.

```
--limit=<limit in GB>
--limit <limit in GB>
```

This option allows you to change the limit used for importing traces (default 10GB). Traces over this limit size will error. The size is specified in GB - so a 25GB trace could be imported by setting `--limit 25`.

```
--home=<home directories>
-h <home directories>
```

Specify what directories should be considered as “home directory” when running the healthcheck diagnosis (useful when the trace was generated on another machine). It can be one of:

- a comma-separated list of paths. In this case, the paths cannot contain commas (`,`);
- a path to a file containing a list of paths, each on its own line. In this case, the file path must be prefixed with `file:`, e.g. `--home=file:/tmp/home_dirs`. In this file, leading whitespace is ignored, and lines starting with `#` (comments) and blank lines are skipped.

If not specified, the current home directory will be used. Note that this is meaningful only with the healthcheck mode.

```
--restricted=<restricted directories>
-r <restricted directories>
```

Specify what directories should be considered as restricted when running the healthcheck diagnosis. It can be one of:

- a comma-separated list of paths. In this case, the paths cannot contain commas (`,`);
- a path to a file containing a list of paths, each on its own line. In this case, the file path must be prefixed with `file:`, e.g. `--restricted=file:/tmp/r_dirs`. In this file, leading whitespace is ignored, and lines starting with `#` (comments) and blank lines are skipped.

If not specified, this section of the report will not be populated. Note that this is meaningful only with the healthcheck mode.

```
--licqueue
```

When used in conjunction with a license server BreezeAP will wait for a valid license, rather than exiting if none are available.

```
--mem=XG
--mem XG
```

Specify the amount of memory allocated to the JVM. The java default will be used if not specified. The `--mem` option argument must match the following pattern: `[1-9][0-9]*[GgMmKk]` For instance: `--mem=1024k`, `--mem=1G`, `--mem=1073741824`. Note that if no unit is provided, bytes is assumed.

8.2.1 Examples:

Output all combinations of output type and format for the given trace:

```
$ breezeAP.sh input.trace --all
```

Output one io-summary file, with one line for each trace listed, including any child traces. This is a good way to start looking at a large MPI trace that used lots of machines. The data will be output to a single csv file:

```
$ breezeAP.sh /path/to/trace -k -y io-summary -f csv -o /path/to/my_out
```

Exports just the dependencies for the traces listed, including any child traces. The dependencies export is faster than other modes. The data will be output to an xml file:

```
$ breezeAP.sh '/path/to/app_trace[0-9]*' -y dependencies -f xml -k \  
-o /path/to/my_out
```

Export the file dependencies for the traces listed, including any child traces, and for each file the I/O patterns will be included. The data will be output to an xml file:

```
$ breezeAP.sh '/path/to/app_trace[0-9]*' -y files -f xml -k -o /path/to/my_out
```

Create a Breeze Healthcheck HTML report:

```
$ breezeAP.sh input.trace -y healthcheck -f html
```

9 Breeze Healthcheck

9.1 Introduction

Breeze Healthcheck automatically generates an I/O report for your application.

9.2 Installing Breeze Healthcheck

Breeze Healthcheck is included in the Breeze download.

Before you can run Breeze Healthcheck you must have a valid license set up, please see `breeze_installation.pdf` for details.

9.3 Generating a Healthcheck Report

To generate a report for an application called “app” in `~/myapplication` you would `cd` to the `bin` directory of your application and run `breeze-healthcheck.sh`. For example:

```
$ cd ~/myapplication
$ breeze-healthcheck.sh -f ~/tmp ./app
```

The `-f` command line option specifies a directory where Breeze Healthcheck will store its temporary files. Breeze Healthcheck will report an error and exit if this directory already exists.

This will run your application, tracing its use. At the end of this process the trace files that have been generated are analysed to create the Healthcheck report. This will be created in the output directory you specified e.g.

```
~/tmp/app-healthcheck/
```

In this folder there will be up to 4 files created:

- `healthcheck_report.html`: the main Healthcheck report;
- `report.txt`: the main Healthcheck report in text format;
- `dependencies.txt`: a list of dependencies that your application used;
- `network.csv`: information about the network locations that your application accessed, this file is only created if network access is detected;
- `programs.csv`: detailed information about each process in your application.

The report is human readable and has recommendations for changes that could be made to your application.

9.4 Creating an Empty Sample Report

You can generate the output documentation for Healthcheck by running:

```
$ breeze-healthcheck.sh --docs
```

This will generate a full `report.txt` file, so you can see what could potentially be reported when running your application under Breeze Healthcheck. Where necessary, explanations should inform you as to what is reported, and how.

9.5 Healthcheck Command Line Options

The main command line options are described below. To get a full list of options, run the `breeze-healthcheck.sh` script without any option.

`-f <output_directory>`

Specify the directory in which Breeze Healthcheck will store its data. Breeze Healthcheck will not generate a report if this option is not specified.

`--home <home_dirs>`

When generating a Healthcheck report it is possible to specify which directories should be treated as the users' home directory. By default Healthcheck uses the current users home directory if this is not specified. This could be useful for monitoring directories that programs should not be run from in a production environment. To do this specify the `--home` command line option with a comma separated list of paths as follows:

```
$ breeze-healthcheck.sh --home \
/home/staff/user1,/home/contractors/user2 \
-f ~/tmp ./app
```

`--traceonly`

Optional. If specified, Breeze Healthcheck generates traces but doesn't create the report.

This option allows you to run the report generation later. Using the following command:

```
$ breeze-healthcheck.sh <trace directory> -f <output_directory>
```

`--thresholds <threshold-file>`

Optional. Specify the path to the file where thresholds are defined. The file should exist and be readable. Without this option, Breeze Healthcheck will look for a file named 'healthcheck_thresholds' in your home directory and in its etc directory, and use default hard-coded values in last resort.

This file allows you to configure the output of the report, you can adjust the thresholds over which data is reported to suit your environment/applications.

`--restricted <restricted_dirs>`

Optional. Where `<restricted_dirs>` can either be:

- a comma separated list. In this case, the paths cannot contain commas (',');
- a path to a file containing a list of paths, each on its own line. In this case, the file path must be prefixed with `file:`, e.g. `--restricted=file:/tmp/r_dirs`. In this file, leading whitespace is ignored, and lines starting with `#` (comments) and blank lines are skipped.

Specify what directories should be considered restricted when creating the Healthcheck report. Any file accessed or program/script run from these directories will be listed in the report. If not specified, this section of the report will be empty.

`--limit <max-trace-size>`

Optional. Specify the maximum trace size that can generate a Healthcheck trace in gB. Defaults to 10gB. The argument must only contain numbers.

`--mem <max-heap-size-for-JVM>`

Optional. Sets the maximum heap size of the Java Virtual Machine. Use this option if you are working with big traces and get `OutOfMemoryError`. The argument must only contain numbers and may end with one of the following units: k, K, m, M, g, G.

10 Container Support

10.1 Singularity

Breeze will trace workloads in Singularity and Apptainer containers by default. This will add a number of bind paths to each singularity container, so that Breeze is able to read the configuration files and run the executables that it normally would. If these files are all in one area of your filesystem you can minimise the number of paths that are bound by setting the following environment variable to that path:

`BREEZE_CONTAINER_BIND_PATH`

Note that Breeze will also attempt to add paths it needs to function to the bind path, most notably the output directory where the trace files are saved. It is possible for this to fail when the output directory is in `/tmp`, as Singularity will try to bind mount both `/tmp` and `/var/tmp` to `/tmp` in the container by default. In order to avoid this, it is recommended to either set the output directory to somewhere other than `/tmp` or to change this Singularity behaviour in the Singularity config file, usually `/usr/local/etc/singularity/singularity.conf`.

In order to work around changes to Singularity from v3.6.0 onwards we assume that the first application will be a 64-bit executable. If this is not the case then an error will be output and the executable will not be traced. If this causes any problems please contact support.

10.2 Docker

Breeze does not automatically trace workloads in Docker containers by default – this feature is planned for a future release. You can still trace these jobs manually by mounting Breeze in the docker container and running `trace-program.sh` as normal.

11 Troubleshooting

11.1 Turning off Breeze tracing for specific binaries

It is possible to disable Breeze tracing for specific binaries, this is done by setting an environment variable `BREEZE_NOTRACE_PROGRAMS`. This should contain a comma separated list of the full path, directory or binary name. If specifying a directory, ensure you use the absolute path.

e.g.

```
export BREEZE_NOTRACE_PROGRAMS="ssh,rsh"
```

Will match ssh and rsh, for example `/usr/bin/ssh`.

```
export BREEZE_NOTRACE_PROGRAMS="/bin/,/usr/bin/"
```

Will match locally installed binaries such as `/bin/sh`.

```
export BREEZE_NOTRACE_PROGRAMS="/home/user/dev/project/bin/failing_binary"
```

Will only match the binary that is causing you problems.

By default, Breeze won't trace programs with any of these names: `blaunch`, `bsub`, `lsgrun`, `lrun`, `qsub`, `qrsh`, `sbatch`, `srun`. This is to prevent it from tracing cluster scheduling infrastructure. You can remove a program from this list by including it in `BREEZE_NOTRACE_PROGRAMS` preceded by a `-` sign:

```
export BREEZE_NOTRACE_PROGRAMS="-srun"
```

These removal entries can be combined with additions in the obvious way:

```
export BREEZE_NOTRACE_PROGRAMS="-srun,failing_binary,/usr/bin/ssh,-qsub"
```

11.2 KDE on CentOS 7

There is a problem with the default GTK+ 2 theme in CentOS 7. When running Breeze you may encounter some odd window sizing and errors similar to:

```
java: /build/builddd/gtk2-engines-oxygen-1.4.5/src/animations/
oxygencomboboxdata.cpp:87:
void Oxygen::ComboBoxData::setButton(GtkWidget*)Assertion
`!_button._widget' failed.
```

The simplest solution is to change the theme in System Settings -> Application Appearance -> GTK+ Appearance - you can then change the theme from Oxygen to Adwaita.

For more detail about other work-arounds please see the [Eclipse release notes](#).

11.3 X-Forwarding to MacOS

If you are running Breeze on a remote linux system and using X-Forwarding over ssh from a Mac with XQuartz then you may run into this issue.

Some parts of Breeze GUI use JavaFX for rendering, this requires a feature (igl) that is disabled by default in MacOS. The following steps enable it:

- at the Terminal CLI on Mac run `defaults write org.macosforge.xquartz.X11 enable_iglx -bool true`
- restart xquartz