Breeze TraceOnly User Manual

2.13.3

Table of Contents

1 Introduction

Breeze HPC is a tool used to solve deployment, tuning issues when installing and running complex Linux applications.

Breeze TraceOnly is a small download that lets you trace applications and send them to someone who has a full Breeze license. You can't look at the data without a Breeze license, but if your software vendor does, then you can send them data so they can work out what the problem is.

Breeze TraceOnly traces application arguments, environment and dependencies for use in troubleshooting build or installation issues and resolving problems caused by missing files or libraries.

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

2 Installation

Download the latest version of Breeze TraceOnly from our website and extract it somewhere sensible. Please make sure that you download the appropriate version of Breeze TraceOnly (32 or 64bit) for the machine you want to run it on.

Breeze TraceOnly does not require any special permissions or license and can be run by any user authorised to run the application under investigation.

3 Tracing an application

Breeze TraceOnly is run using the **trace-program.sh** script available at the top-level directory of the installation.

To trace and profile an application you just type '**trace-program.sh -f <output directory>**' followed by your command and arguments. For example:

```
$ ./trace-program.sh -f ~/traceOutput make all
```

If the output directory specified in the **-f** option exists and already contains trace data the script will display a warning message and exit.

3.1 Command line options

The following section lists all the valid command line options used by **trace-program.sh**. All options to **trace-program.sh** must be specified before the command to be traced.

--bash-aliases=<alias file>
-ab <alias file>
Supply a file of bash alias definitions. Breeze needs the definitions in order to trace aliases.

A suitable alias file can be generated by running the following command in bash before running this script:

```
$ alias > alias.txt
```

--post-trace=<post-trace-command>
-c <post-trace-command>
Execute a post-trace command after program under trace has finished.

The command itself won't be profiled, traced, monitored or throttled. 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/log/file"**. If the command doesn't finish within 10 minutes, it will be killed.

--log=<filename>
-l <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 tracing data will be written to, and which is used by Breeze TraceOnly for temporary storage. This option is required.

--profile=<yes|no>
-p
This option turns profiling on or off. When enabled Breeze collects many types of statistics on the operation of the programs under trace. Profiling is on by

default, but turning this off may speed up tracing and reduce the size of the output. The exact set of statistics that are collected is controlled by environment variables described in Profiling Options.

--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.

--remote=<[bsub][,qsub][,rsh][,ssh][,srun][,sbatch][,lsrun][,lsbatch]>
--remote=<yes|no>
-r
This options 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 supported job launching commands or one of **yes** or **no**. The value **yes** is equivalent to listing all valid job launching commands and is the default value for this option. Setting this option to **no** disables tracing of any child jobs.

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

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.

--remote-job=yes
--remote-job
Track remote jobs. When a one or more remote child jobs are launched from a top-level command/script then the top-level job waits for all the remote 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.

--stat=<yes|no>
-S
By default calls in the stat family (**stat**, **fstat** and **lstat**) are not traced and profiled. Turning this on may slow down tracing and increase the size of the output.

--tcsh
-t
Run the command to be traced in a **tcsh** shell

--tcsh-aliases=<alias file>
-at <alias file>
Supply a file of **tcsh** or **csh** alias definitions. Breeze needs the definitions in order to trace aliases.

A suitable alias file can be generated by running the following command in **tcsh** or **csh** before running this script:

```
$ alias > alias.txt
```

--trace=<all-io|yes|no>

This option turns tracing on or off. Tracing is on by default.

The value **all-io** enables complete I/O tracing. With **--trace=all-io**, Breeze TraceOnly collects data on all reads, writes, and seeks in addition to the standard tracing data. Whereas in the default tracing mode (**--trace=yes**), only the first read, write, and seek operation for each file is recorded. N.B. Using **--trace=all-io** option may considerably slow down tracing and may increase the size of the output significantly - enabling profiling (on by default) will give most of the required information with a lower overhead.

--variant=<mpich|mvapich|ompi>

This option selects Breeze variant, which enables additional tracing functionality.

Currently supported values enable MPI I/O tracing for MPICH (**--variant=mpich**), MVAPICH (**--variant=mvapich**) and OpenMPI (**--variant=ompi**) applications.

3.2 Profiling Options

The exact set of statistics that are collected is controlled by environment variables. These environment variables are summarised below.

BREEZE_PROFILE_BUCKETS

A comma separated list of buckets.

Breeze TraceOnly aggregates file system statistics over specified subsets of the file system, which we refer to as "buckets".

A bucket may be any file or directory. If a bucket name contains a comma it must be escaped with a single backslash "\" character.

Defaults to all top-level directories in your file system and all active mount points.

BREEZE_PROFILE_BUCKET_STATS

Boolean, "1" for on, "0" for off.

When set to "1" Breeze TraceOnly collects the following statistics.

First, counts of the number of calls to functions that use the file system. These functions are aggregated into the following groups:

Group	Calls
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, ...

Second, counts of the number of bytes read and written and the seek distance.

Each of these statistics is aggregated for each of the file system buckets configured by `BREEZE_PROFILE_BUCKETS` (see above).

Defaults to "1" for on.

BREEZE_PROFILE_TIME_INTERVAL

An integer value that specifies how often statistics are reported.

By default, time intervals are assumed to be given in milliseconds, but you can explicitly use the unit "us" for microseconds, "ms" for milliseconds, or "s" for seconds.

Defaults to “1000ms” (1 second).

BREEZE_PROFILE_NETWORK_STATS

Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly collects counts of calls to functions that use the network. These functions are aggregated into the following groups:

Group	Calls
accept	accept
bind	bind
connect	connect
listen	listen
read	read, recv, ...
write	write, send, ...

These statistics are aggregated by each remote address accessed.

Defaults to “1” for on.

BREEZE_PROFILE_BUCKET_LATENCY

Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly measures the time taken by function calls that use the file system.

These functions are aggregated into the groups described under `BREEZE_PROFILE_BUCKET_STATS` above (accept, access, connect, fschange, glob, open, read, write).

Breeze collects maximum and minimum latencies, and counts of calls that fall into each of the latency ranges configured by `BREEZE_PROFILE_TIME_RANGES` (see below), for each of the file system buckets configured by `BREEZE_PROFILE_BUCKETS`.

Defaults to “1” for on.

BREEZE_PROFILE_NETWORK_LATENCY

Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly measures the time taken by function calls that use the network.

These functions are aggregated into the groups described under `BREEZE_PROFILE_NETWORKS_STATS` above (accept, bind, connect, listen, read, write).

Breeze collects maximum and minimum latencies, and counts of calls that fall into each of the latency ranges configured by `BREEZE_PROFILE_TIME_RANGES` (see below), for each remote address accessed.

Defaults to “1” for on.

BREEZE_PROFILE_TIME_RANGES

A comma separated list of time interval boundaries.

When `BREEZE_PROFILE_BUCKET_LATENCY` or `BREEZE_PROFILE_NETWORK_LATENCY` is turned on, Breeze aggregates counts of calls that fall into a set of time ranges (count of calls taking less than 1us, count of calls taking 1-10us, ...).

Each time interval boundary must be specified as an integer value. If not specified the interval is assumed to be given in milliseconds, but you can explicitly use the unit "us" for microseconds, "ms" for milliseconds, or "s" for seconds.

For example, if you set:

```
BREEZE_PROFILE_TIME_RANGES=1us,1ms,1s
```

Then there are four ranges defined: ≤1us, 1us-1ms, 1ms-1s and >1s.

Breeze TraceOnly will accept up to 15 values for this setting (hence up to 16 ranges).

Defaults to “`1us,10us,100us,1ms,10ms,100ms,1s,10s,100s,1000s`”.

BREEZE_PROFILE_FAILED_IO

Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly collects counts of function calls that failed.

These functions are aggregated into the groups described above (accept, accesss, bind, connect, fschange, glob, listen, open, read, seek, write).

Each of these statistics is aggregated for each of the file system buckets configured by `BREEZE_PROFILE_BUCKETS` (see above), and for each remote address (in the case of network functions).
The failures are further aggregated by error number (errno).

Defaults to “1” for on.

BREEZE_PROFILE_FS_TRAWL
Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly identifies cases when a program "trawls" the file system, testing many non-existent file system paths in succession.

File system trawls can happen when the environment is poorly configured, for example if the PATH has many elements, and so programs have to search many places to find the files that they need. On distributed file systems this can cause serious performance degradation.

Breeze defines a "trawl" as being an uninterrupted sequence of `BREEZE_PROFILE_TRAWL_LENGTH` (see below) or more failed calls to the same function. The trawl is ended either by a successful call of that function, or by a call to a different function.

Breeze records the number of failed calls in the trawl, the name of the file associated with the final failed call, and the time taken by the whole sequence of failed calls.

Defaults to “1” for on.

BREEZE_PROFILE_TRAWL_LENGTH
An integer value that specifies the minimum number of failed calls that Breeze considers to be a "trawl". See `BREEZE_PROFILE_FS_TRAWL` above.

Defaults to “4”.

BREEZE_PROFILE_MEMORY_USAGE
Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly reports the memory used by the program being profiled.

Breeze records the "total program size" (reserved virtual memory) and the "resident set size" (mapped memory) as reported by /proc/[pid]/statm. See "man proc(5)" for details.

Defaults to “1” for on.

BREEZE_PROFILE_SYMLINK_COUNT
Boolean, “1” for on, “0” for off.

When set to “1” Breeze TraceOnly counts the number of symbolic links that have to be followed to resolve each file system path used by the program under trace.

Breeze aggregates counts of file system operations by the length of the symlink chain, up to `BREEZE_PROFILE_SYMLINK_DEPTH` (see below).

Defaults to “1” for on.

BREEZE_PROFILE_SYMLINK_DEPTH

An integer value that specifies the maximum length of a chain of symbolic links that Breeze TraceOnly will follow. See `BREEZE_PROFILE_SYMLINK_COUNT` above.

Defaults to “5”.

BREEZE_PROFILE_CPU_USAGE

Boolean, “1” for on, “0” for off.

When `BREEZE_PROFILE_CPU_USAGE` is turned on, Breeze TraceOnly reports the CPU time used by the program being profiled.

Breeze TraceOnly records the "user CPU time" and the "system CPU time" as a percentage value. It may reach more than 100% on multi-core CPUs within multi threaded applications. It also records "voluntary context switches" and "involuntary context switches". The values represent the delta to the last measurement.

Defaults to “1” for on.

3.3 Tracing applications on remote hosts

Breeze TraceOnly currently supports tracing applications on remote hosts using **bsub**, **qsub**, **rsh**, **ssh**, **srun**, **sbatch**, **lsrun**, 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 TraceOnly installation is available via the same path on all possible remote host nodes.

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

```
<output directory>/remotetrace-<hostname>-<PID>
```

Additionally 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>/remotetrace-<hostname>-<PID>-<array index>
```

3.4 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 TraceOnly will not automatically detect compound commands when re-writing job submissions to remote hosts.

Alternatively, you can source **trace-program.sh**, execute the commands you want to trace and exit the shell:

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

3.5 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 would then read/write into 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 deletes an existing mapping.

4 Removing confidential information from trace output

It is possible that while tracing an application Breeze TraceOnly 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 TraceOnly 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 top-level directory of the installation.

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 TraceOnly output directory. This will either be the directory passed as the **-f** option to a **trace-program.sh** command or a trace directory created as the result of running a command on a remote host (as described in section 3.3 Tracing applications on remote 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 which allows the user to change any confidential values.

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 binary output.