Mistral User Manual

2.13.1

Table of Contents

1 Introduction

Mistral is a tool used to report on and resolve I/O performance issues when running complex Linux applications on high performance compute clusters.

Mistral is a small download that allows you to monitor application I/O patterns in real time, and log undesirable behaviour using rules defined in a configuration file called a contract.

2 Installation

Extract the Mistral product archive that has been provided to you somewhere sensible. Please make sure that you use the appropriate version of Mistral (32 or 64bit) for the machine you want to run it on.

Mistral requires a license, please contact Ellexus if you do not already have a valid Mistral license.

The environment variable `MISTRAL_LICENSE` must be set to the location of your license, which can be one of:

a) The pathname of a specific license file.
b) The pathname of a directory which contains one or more license files.

Mistral will attempt to detect the installation directory correctly on start up however some job schedulers, e.g. Univa Grid Engine, use a spool directory that can break this detection. In this case the environment variable `MISTRAL_INSTALL_DIRECTORY` must be set to the directory used for installation.

There are two flavours of Mistral, designed to be used with either `/bin/bash` or `/bin/[t]csh` as interpreter.

If Mistral is intended to be used with a job scheduler all required environment variables must be available in all interactive and non-interactive shells. It is recommended that global environment variable settings be added to `/etc/bashrc` or `/etc/cshrc` and individual user settings to the user's `.bashrc` or `.cshrc` file.

3 Configuring Mistral

3.1 Configuring contract and log locations

Mistral determines what events to log and/or throttle by using contract files. Mistral uses two types of contracts, local and global. This enables administrators to define global settings for the entire system while also allowing for the creation of tuned settings for specific workloads. The following environment variables configure the locations Mistral uses for contract and log files.

It is not necessary to configure both global and local contracts but at least one valid contract / log pair must be defined. When testing it may be preferable to just use one contract, either local or global, for simplicity.

Environment Variable	Description
MISTRAL_CONTRACT_MONITOR_GLOBAL	The name and location of the global monitoring contract file.
MISTRAL_CONTRACT_MONITOR_LOCAL	The name and location of the local monitoring contract file.
MISTRAL_CONTRACT_THROTTLE_GLOBAL	The name and location of the global throttling contract file.
MISTRAL_CONTRACT_THROTTLE_LOCAL	The name and location of the local throttling contract file.
MISTRAL_LOG_MONITOR_GLOBAL	The name and location of the file in which Mistral will log violations of global contract rules.
MISTRAL_LOG_MONITOR_LOCAL	The name and location of the file in which Mistral will log violations of local contract rules.
MISTRAL_LOG_THROTTLE_GLOBAL	The name and location of the file in which Mistral will log throttling events triggered by a violation of a global contract rule.
MISTRAL_LOG_THROTTLE_LOCAL	The name and location of the file in which Mistral will log throttling events triggered by a violation of a local contract rule.
MISTRAL_LOG_BACKTRACE	The name and location of the file in which Mistral will log backtraces taken when a rule is violated.
MISTRAL_BACKTRACE_SAMPLE	If set to a non zero value, as well as recording backtraces when a rule is violated, Mistral will record a backtrace for each **MISTRAL_BACKTRACE_SAMPLE**‘th I/O operation.
MISTAL_TRAFFIC_LIGHT_LOG	The name and location of the file in which Mistral will log traffic light numbers at the end of a run.
MISTRAL_TRAFFIC_LIGHT_PER_PROCESS	Unset - don’t log per-process traffic light

Environment Variable	Description
	entries (default). 1: Log per-process traffic light entries.

3.2 Configuring MPI I/O support

Mistral supports monitoring and throttling MPI I/O for selected MPI variants. MPI I/O support is not enabled by default. You can enable MPI I/O support by setting `MISTRAL_VARIANT` environment variable to one of the following values:

`mpich`	Enables Mistral MPI I/O support for MPICH.
`mvapich`	Enables Mistral MPI I/O support for MVAPICH.
`ompi`	Enables Mistral MPI I/O support for OpenMPI.

3.3 Contract specification

Contracts are configuration files that specify I/O limits for a process.

If any monitoring rule limit is exceeded a log message is output indicating which process broke the limit and by how much.

If a throttling rule limit is exceeded a log message is output indicating the process that contributed most to breaking the limit and all job processes are rate limited until the job falls within the defined rule.

3.3.1 Contract Header

The first line of the file specifies the contract type and the time frame in the format

```
<VERSION>,<CONTRACT-TYPE>,<TIMEFRAME-PERIOD><TIMEFRAME-UNIT>
```

where:

`<VERSION>` is the contract format version number which must be “2” for this release of Mistral;

`<CONTRACT-TYPE>` is either `monitortimeframe` or `throttletimeframe`;

`<TIMEFRAME-PERIOD>` is the length of the time frame for all rules in this contract, specified as an integer, followed by;

`<TIMEFRAME-UNIT>` which must be “ms” for milliseconds or “s” for seconds.

For example:

```
2,monitortimeframe,15s
```

3.3.2 Comment Lines

Blank lines and lines starting with “#” are ignored and can be used for adding comments to a contract file.

3.3.3 Contract Rules

Each remaining line specifies a rule in the format

```
<LABEL>,<PATH>,<CALL-TYPE>,<SIZE-RANGE>,<MEASUREMENT>,<THRESHOLD><UNIT>
```

where:

<LABEL> is the name of this rule. It appears in log entries related to this rule. It is an arbitrary string of the letters a-z (in lower or upper case), the digits 0-9, the underline character ("_") or a hyphen ("-").

<PATH> is an absolute file system path. The rule applies to function calls on paths starting with this value. Mistral de-references all relative paths and symbolic links therefore this path must be fully resolved.

Note that this is a purely string-based comparison. For example, the path "`/usr/lib32/libm.a`" matches a rule where `<PATH>` is "`/usr/lib`". This is useful when many paths share a prefix, so for example it is possible to set `<PATH>` to "`/tmp/output/test-`" in order to match "`/tmp/output/test-1`" and so on. To restrict the path to a particular directory, it must be specified with a trailing "**/**", for example "`/usr/lib/`".

<CALL-TYPE> is the set of call types to which the rule applies. It must specify one or more of these call types:

access	Calls that access file system meta data (stat, readlink, etc.).
create	Calls that create new files (open, creat, mkdir, etc.).
delete	Calls that delete files (remove, rmdir,unlink, etc.).
fschange	Calls that update file system meta data (chmod, rename, etc.).
open	Calls that open existing files (open, fopen, opendir, etc.).
read	Calls that read data from the file system (read, fgets, mmap, readdir, recv, scanf, etc.).
seek	Calls that update the current position within a file (fseek, lseek, rewind, etc.).
write	Calls that write data to the file system (write, error, printf, putc, send, warn, etc.).

When a rule applies to multiple call types, join them with "`+`" signs. For example, "`read+write`" matches calls that either read or write data.

If you enable MPI I/O support (see section 3.2 Configuring MPI I/O

support), following MPI I/O specific call types can also be used:

mpi_access	Calls that access MPI file meta data (MPI_File_get_amode, MPI_File_get_size, etc.).
mpi_create	Calls that create new MPI files (MPI_File_open with MPI_MODE_CREATE mode).
mpi_delete	Calls that delete MPI files (MPI_File_delete).
mpi_fschange	Calls that update MPI file meta data (MPI_File_set_atomicity, MPI_File_set_size, etc.).
mpi_open	Calls that open existing MPI files (MPI_File_open without MPI_MODE_CREATE mode).
mpi_read	Calls that read data from MPI files (MPI_File_read, MPI_File_iread_at, MPI_File_read_all_begin, etc.).
mpi_seek	Calls that update the current position within an MPI file (MPI_File_seek, MPI_File_seek_shared).
mpi_sync	Calls that synchronize MPI file data (MPI_File_sync).
mpi_write	Calls that write data to MPI files (MPI_File_write, MPI_File_iwrite_at, MPI_File_write_all_begin, etc.).

<SIZE-RANGE> specifies the range of sizes that match this rule. A size range may only be specified for rules with any combination of the call types “read”, “write”, and “seek” (other types of call have no associated size). A size range is specified in the format:

`<SIZE-MIN><SIZE-MIN-UNIT>-<SIZE-LIMIT><SIZE-LIMIT-UNIT>`

meaning that a matching size must be at least `<SIZE-MIN>` but lower than `<SIZE-LIMIT>`. The `<SIZE-MIN-UNIT>` and `<SIZE-LIMIT-UNIT>` are the corresponding units, and must be one of the following:

- “`B`” Bytes
- “`kB`” Kilobytes (1,000 bytes)
- “`MB`” Megabytes (1,000,000 bytes)
- “`GB`” Gigabytes (1,000,000,000 bytes)

For example, a size range of “`1kB-4kB`” matches reads (or writes) with 1000 ≤ size < 4000. (Note the asymmetric bounds: these make it easier to specify non-overlapping ranges.)

`<SIZE-MIN><SIZE-MIN-UNIT>` may be omitted, in which case the value 0 is used. `<SIZE-LIMIT><SIZE-LIMIT-UNIT>` may be omitted, in which case there is no upper limit.

If a rule is to apply to all of the specified operations regardless of size, or size is not applicable to one or more of the call types specified in the rule this field must be set to “`all`”.

<MEASUREMENT> is the type of data being measured. The list of valid measurement types differs between monitoring throttling rules. For monitoring rules it must be one of:

bandwidth	Amount of data processed by calls of the specified type in the time frame. This applies only to “read” and “write” calls.
count	The number of calls of the specified type in the time frame.
seek-distance	Total distance moved within files by calls of the specified type in the time frame. This applied only to “seek” calls.
max-latency	The maximum duration of any call of the specified type in the time frame. See section 3.3.4 Latency Sampling.
mean-latency	The mean duration of any call of the specified type in the time frame, provided the number of calls is higher than the value of `MISTRAL_MONITOR_LATENCY_MIN_IO`. See section 3.3.4 Latency Sampling.
total-latency	The total duration of time spent in calls of the specified type in the time frame, provided the number of calls is higher than the value of `MISTRAL_MONITOR_LATENCY_MIN_IO`. See section 3.3.4 Latency Sampling.
memory	The amount of memory used by the job. This is calculated using the Resident Set Size (RSS) of the individual processes. The <PATH>, <CALL-TYPE> and <SIZE-RANGE> fields should be left blank.
user-time	The amount of CPU time used by the job, while executing user code. The <PATH>, <CALL-TYPE> and <SIZE-RANGE> fields should be left blank.
system-time	The amount of CPU time used by the job while executing system code. The <PATH>, <CALL-TYPE> and <SIZE-RANGE> fields should be left blank.
cpu-time	The amount of CPU time used by the job while executing either user or system code. The <PATH>, <CALL-TYPE> and <SIZE-RANGE> fields should be left blank.

For throttling rules the only valid measurements are “`bandwidth`”, “`count`”, “`total-latency`”, “user-time”, “system-time” and “cpu-time” as described above.

<THRESHOLD> is the limit for this rule. If the measured data exceeds `<THRESHOLD>` in `<TIMEFRAME>`, then the violation is logged. Monitoring contracts

allow 0 and throttling contracts allow 1 as the lowest limit.

<UNIT> is the unit for <THRESHOLD>. When <MEASUREMENT> is "bandwidth", "seek-distance" or "memory", this must be one of:

"B"	Bytes
"kB"	Kilobytes (1,000 bytes)
"MB"	Megabytes (1,000,000 bytes)
"GB"	Gigabytes (1,000,000,000 bytes)

When <MEASUREMENT> ends with "-latency" or "-time", this must be one of:

"us"	Microseconds
"ms"	Milliseconds
"s"	Seconds

When <MEASUREMENT> is "count", this must be one of:

<blank>	Exact number of calls
"k"	Thousands of calls
"M"	Millions of calls

For example:

```
red,/mnt/net/abc,write,all,bandwidth,100MB
```

3.3.3.1 Monitoring rules

Monitoring rules within the same contract are grouped by <PATH>, <CALL-TYPE> and <MEASUREMENT>. If multiple rules in a group have been violated simultaneously, only the rule with the highest <THRESHOLD> is logged.

For example, consider the contract:

```
2,monitortimeframe,1s
#LABEL,PATH,CALL-TYPE,SIZE-RANGE,MEASUREMENT,THRESHOLD
Red,/mnt/net/abc,write,all,bandwidth,1MB
Yellow,/mnt/net,write,all,bandwidth,10MB
Green,/mnt/net/abc,write,all,bandwidth,10kB
#Black,/mnt/net/abc,write,all,bandwidth,1kB
```

In this example, if the application writes more than 10kB/s in subdirectories of "/mnt/net/abc" the Green rule is violated and logged. If it writes more than 1MB/s in "/mnt/net/abc", the Green and Red rules are violated but only the Red rule is logged. If it writes more than 10MB/s in "/mnt/net/abc", the Red, Yellow and Green rules all match, but only the Red and Yellow rules are logged. The Black rule is never logged, because it has been commented out with "#".

3.3.3.2 Throttling rules

If a path matches multiple rules in the throttle contract file, then all of the limitations apply. For example, if the contract file contains:

```
thr_root_r,/,read,all,bandwidth,10000000B
thr_usr_r,/usr,read,all,bandwidth,500000B
thr_usrlib_r,/usr/lib,read,all,bandwidth,90000B
```

Then a read from "`/usr/lib/libc.so`" is subject to all three bandwidth limitations, whereas a read from "`/etc/passwd`" is subject only to the first.

3.3.4 Latency Sampling

Latency measurements incur a larger processing overhead than simple count or bandwidth operations. Such measurements are also subject to greater variability in value. To limit the impact of these problems Mistral implements measurement sampling on any latency rules defined in a monitoring contract.

Latency sampling is controlled via three environment variables. All three variables must be set to a positive integer if defined.

Environment Variable	Description
MISTRAL_MONITOR_LATENCY_SAMPLE	The sampling rate. If set to n Mistral will only measure the latency of every nth operation. Setting this to 1 will cause the latency of all I/O operations to be measured. Defaults to 10.
MISTRAL_MONITOR_LATENCY_MIN_IO	The minimum number of I/O operations that must be seen in a single time frame for the sample to be considered statistically significant. Defaults to 100.
MISTRAL_MONITOR_LATENCY_MAX_IO	The minimum number of I/O operations that must be seen in a single time frame for the sample to be considered statistically significant. Defaults to 100.

Latency measurements are not made if no latency rules are defined.

The minimum and maximum counts defined above are applied individually to each `<CALL-TYPE>` class. For example, using default configuration where a minimum of 100 operations must be seen before rules are applied, if 150 `read` operations are sampled in a single time frame any defined latency rules against `read` events will be applied. If, during that same time frame, only 75 `write` operations are seen any latency rules defined against `write` events will not be applied.

It is important to note that any `total-latency` rules defined are only compared to the sampled I/O operations and are subject to the maximum sample size count limit defined.

3.3.5 Adjusting contracts

It is possible to update contracts for running jobs and can be particularly useful to increase thresholds to prevent excessive logging. How this is done differs between global and local contracts.

Global contracts are assumed to be configured with high “system threatening” rules that should not be frequently changed. These contracts are intended to be maintained by system administrators and will be polled approximately once a minute for changes on disk.

Local contracts can be update dynamically during a job execution run by the use of an update plug-in. Using an update plug-in is the only way to modify the local contracts in use by a running job. If an update plug-in configuration is not defined Mistral will use the same local configuration contracts throughout the life of the job.

Please see section 6 for details on the configuration and use of plug-ins.

3.4 Log Entries

Log entries are output in the following format:

```
<TIME-STAMP>,<LABEL>,<PATH>,<CALL-TYPE>,<SIZE-RANGE>,<MEASUREMENT>,
<MEASURED-DATA>/<TIMEFRAME-PERIOD><TIMEFRAME-UNIT>,
<THRESHOLD>/<TIMEFRAME-PERIOD><TIMEFRAME-UNIT>,<HOSTNAME>,<PID>,<CPU>,
<COMMAND-LINE>,<FILE-NAME>,<JOB-GROUP-ID>,<JOB-ID>,
<MPI-WORLD-RANK>,<BACKTRACE>
```

Where the field definitions are as follows:

<TIME-STAMP> is either the end of the time frame where the violation occured (monitoring contract) or when the first rule was violated in the current time frame (throttling contract). The time-stamp is in ISO 8601 format with microsecond precision (`YYYY-MM-DDThh:mm:ss.ffffff`).

<LABEL> is copied from the violated rule.

<PATH> is the path that caused `<MEASURED-DATA>` to exceed `<THRESHOLD>`.

<CALL-TYPE> is copied from the violated rule.

<SIZE-RANGE> is copied from the violated rule.

<MEASUREMENT> is copied from the violated rule.

<MEASURED-DATA> is the data rate of the job that exceeded the limit.

<THRESHOLD> is copied from the violated rule.

<TIMEFRAME-PERIOD> is copied from the violated rule.

<TIMEFRAME-UNIT> is copied from the violated rule.

<HOSTNAME> is the name of the host where the rule was violated on. The hostname includes the domain name.

<PID> is the id of the process in the job that performed the most I/O that contributed to violating the rule.

<CPU> is the number of the CPU where the process (PID) was running on. If the process was multi-threaded, this is the CPU where the thread that violated the rule was running on.

<COMMAND-LINE> is the name of the process in the job that performed the most I/O that contributed to violating the rule. It includes the parameters for the execution.

<FILE-NAME> is the path to the file being accessed when the rule is

exceeded.

<JOB-GROUP-ID> is the job group identifier for the job group that violated the rule.

<JOB-ID> is the job identifier for the job that violated the rule.

<MPI-WORLD-RANK> is the MPI world rank number. This is only set for MPI applications when MISTRAL_VARIANT environment variable has been set to one of the supported MPI variants.

<BACKTRACE> is a the number of the backtrace associated with the I/O operation which violated the rule.

3.4.1 Example Log Entries

The following is an example of a rule violation log entry:

```
2015-01-30T14:30.108355,red,/mnt/net/abc,write,all,bandwidth,102MB/
15s,1MB/15s,foo.bar.com,1234,1,/mnt/tool/bin/abc -d
-e,/mnt/net/abc/file,5,5,7
```

If the <PATH> in one rule is a subdirectory of the <PATH> in another rule, then a single process accessing the subdirectory may violate both rules e.g.

```
2015-01-30T14:30.108355,red,/mnt/net/abc,write,all,bandwidth,102MB/
15s,1MB/15s,foo.bar.com,1234,0,/mnt/tool/bin/abc -d
-e,/mnt/tool/abc/file2,5,5,7
2015-01-30T14:30.108469,yellow,/mnt/net,write,all,bandwidth,102MB/15s,
10MB/15s,foo.bar.com,1234,0,/mnt/tool/bin/abc -d
-e,/mnt/tool/abc/file1,5,5,7
```

Although violated throttling rules will cause Mistral to slow the I/O operation of all processes within a job, any I/O operation that is already in progress when throttling is applied will complete without any modification by Mistral.

As a result the I/O rate measured may still exceed the defined limit even under throttling. The actual I/O rate that was achieved when applying the throttle is output in the <MEASURED-DATA> field.

3.5 Traffic Light Log

Traffic light mode is disabled by default under Mistral. It can be enabled by setting **`MISTAL_TRAFFIC_LIGHT_LOG`**. Mistral will collect aggregated statistics about the type of I/O that was performed. This has been split into three categories, good (green), medium (yellow) and bad (red).

By default only per-job entries are logged. Per-process entries are logged if **`MISTAL_TRAFFIC_LIGHT_PER_PROCESS=1`** environment variable has been set.

3.5.1 Traffic Light Log format

Log entries are output in the following format with one entry per job:

```
<TIME-STAMP>,<RUN-TIME><IO-TIME><%IO-TIME><IO-CALLS>
<RED-TIME>,<%RED-TIME>,<RED-CALLS>,<%RED-CALLS>,
<YELLOW-TIME>,<%YELLOW-TIME>,<YELLOW-CALLS>,<%YELLOW-CALLS>,
<GREEN-TIME>,<%GREEN-TIME>,<GREEN-CALLS>,<%GREEN-CALLS>,
<JOB-GROUP-ID>,<JOB-ID>
```

Where the field definitions are as follows:

`<TIME-STAMP>`	Time when this log entry was created. We use ISO 8601 format with microsecond precision: YYYY-MM-DDThh:mm:ss.ffffff.
`<RUN-TIME>`	Wallclock runtime of this job (µs).
`<IO-TIME>`	Time spent doing I/O calls (µs).
`<%IO-TIME>`	% of runtime that was spent on I/O.
`<IO-CALLS>`	Total number of I/O calls.
`<RED-TIME>`	Time spent doing bad I/O (µs).
`<%RED-TIME>`	% of total I/O time that is bad I/O.
`<RED-CALLS>`	Number of bad I/O calls.
`<%RED-CALLS>`	% of total I/O calls that are bad I/O.
`<YELLOW-TIME>`	Time spend doing medium I/O (µs).
`<%YELLOW-TIME>`	% of total I/O time that is medium I/O.
`<YELLOW-CALLS>`	Number of medium I/O calls.
`<%YELLOW-CALLS>`	% of total I/O calls that are medium I/O.

<GREEN-TIME>	Time spent doing good I/O (µs).
<%GREEN-TIME>	% of total I/O time that is good I/O.
<GREEN-CALLS>	Number of good I/O calls.
<%GREEN-CALLS>	% of total I/O calls that are good I/O.
<JOB-GROUP-ID>	Job group identifier.
<JOB-ID>	Job identifier.

3.5.2 How red, yellow and green percentages are calculated

Each I/O call has a duration measured in microseconds. Once the call is categorised under bad, medium or good I/O, we accumulate the call duration to get the time spent in red, yellow and green I/O operations. In addition we need to measure the total time the application spent doing I/O. The percentages are then simply calculated as:

% Red time = (Time spent in bad I/O ops) / (Total time spent in I/O ops)
% Yellow time = (Time spent in medium I/O ops) / (Total time spent in I/O ops)
% Green time = (Time spent in good I/O ops) / (Total time spent in I/O ops)

We don't calculate the percentages against the total wallclock runtime, because the application spends time also doing CPU intensive tasks, memory I/O, synchronization (locks), sleeping, etc.

In similar fashion, we calculate the percentages using call counts:

% Red calls = (Number of bad I/O calls) / (Total I/O calls)
% Yellow calls = (Number of medium I/O calls) / (Total I/O calls)
% Green calls = (Number of good I/O calls) / (Total I/O calls)

We log the total time spent in I/O ops, which is:

Total time spent in I/O ops = Red time + Yellow time + Green time

and similarly for total number of I/O calls:

Total number of I/O calls = Red calls + Yellow calls + Green calls

We also log how much of the total running time was spent in I/O:

% I/O Time = (Total time spent in I/O ops) / (Total wallclock runtime)

For multi-threaded processes, the times and call counts are accumulated from each thread. Therefore the total time spent in I/O may be greater than the total wallclock runtime, and equally % I/O Time may be greater than 100%.

3.5.3 Rules for Bad I/O

Definition of bad I/O:

- Small reads or writes.
- Opens for files where nothing was written or read.
- Stats that succeeded on files that were not used.
- Failed I/O.
- Backward seeks.
- Trawls of failed I/O where we include the whole time from the first fail to the last fail or the first success of the same type.
- Zero seeks, reads, writes.
- Failed network I/O.

3.5.4 Rules for medium I/O

Definition of medium I/O:

- Opens for files from which less than N bytes were read or written.
- Stats of files that were used later.
- Forward seeks.

3.5.5 Rules for good I/O

Definition of good I/O:

- Reads and writes greater than **`MISTRAL_PROFILE_SMALL_IO`**.
- Opens for files from which at least **`MISTRAL_PROFILE_SMALL_IO`** bytes were read or written.
- Successful network I/O.

4 Monitoring an application

Once Mistral has been configured it can be run using the `mistral` script available at the top level of the installation. To monitor an application you just type "`mistral`" followed by your command and arguments. For example:

```
$ ./mistral ls -l $HOME
```

By default any error messages produced by Mistral will be written to a file named `mistral.log` in the current working directory. Any errors that prevent the job running as expected, such as a malformed command line, will also be output to `stderr`.

This behaviour can be changed by the following command line options.

`--log=<filename>`
`-l <filename>`
: Record Mistral error messages in the specified file. If this option is not set, errors will be written to a file named `mistral.log` in the current working directory.

`-q`
: Quiet mode. Send all error messages, regardless of severity, to the error log. Command line options are processed in order, therefore this option must be specified first to ensure any errors parsing command line options are sent to the error log.

5 Example contracts

5.1 Monitoring Contract

Consider the following contract:

```
2,monitortimeframe,1s
#LABEL,PATH,CALL-TYPE,SIZE-RANGE,MEASUREMENT,THRESHOLD
High_reads,/usr/,read,all,bandwidth,1MB
High_reads_bin,/usr/bin/,read,all,bandwidth,5MB
Higher_reads_bin,/usr/bin/,read,all,bandwidth,50MB
High_create_lat,/tmp/,create,all,mean-latency,10ms
High_num_w,/home/,write,all,count,750
```

Examining each line individually:

`2,monitortimeframe,1s`

This line identifies the contract as containing monitoring rules that are applied over a time frame of 1 second.

`High_reads,/usr/,read,all,bandwidth,1MB`

This line defines a rule named "`High_reads`" and tells Mistral to generate an alert when the total amount of data read from `/usr/` exceeds 1MB within the 1s time frame.

If a monitored process were to read a 2MB file in `/usr/share/doc/` in less than a second, for example, this rule would be violated and a log message of the following form would be output:

```
2015-07-30T14:30.108355,High_reads,/usr/,read,all,bandwidth,2MB/
1s,1MB/1s,foo.bar.com,15392,0,/mnt/tool/bin/python script.py,
/usr/share/doc/glibc-common-2.17/README.timezone,3,6,7
```

`High_reads_bin,/usr/bin/,read,all,bandwidth,5MB`
`Higher_reads_bin,/usr/bin/,read,all,bandwidth,50MB`

These two lines define two additional rules named "`High_reads_bin`" and "`Higher_reads_bin`" respectively.

All reads in "`/usr/bin/`" will be tested against all three rules currently defined as a read under "`/usr/bin/`" is also a read under "`/usr/`".

If a process read 6MB of data in less than 1 second both the "`High_reads`" rule and the "`High_reads_bin`" rule would be violated. As the rules are defined on different paths a log message for both rule 1 and rule 2 will be output:

```
2015-07-30T14:30.108355,High_reads,/usr/,read,all,bandwidth,6MB/
1s,1MB/1s,foo.bar.com,15392,1,/bin/bash
script.sh,/usr/bin/data,3,6,7
2015-07-30T14:30.108469,High_reads_bin,/usr/
bin/,read,all,bandwidth,6MB/1s,5MB/1s,foo.bar.com,15392,1,/bin/
bash script.sh,/usr/bin/data,3,6,7
```

If a process read 60MB of data in less than 1 second all three currently defined rules would be violated, but only the first and third rule would be logged. This is because Mistral only logs the largest threshold violated when multiple rules are defined on the same "`path`", "`call-type`" and "`measurement`" as is the case with the "`High_reads`" and "`Higher_reads_bin`" rules:

```
2015-07-30T14:30.108355,High_reads,/usr/,read,all,bandwidth,60MB/
1s,1MB/1s,foo.bar.com,15392,0,/bin/bash
script.sh,/usr/bin/data,3,6,7
2015-07-30T14:30.108529,Higher_reads_bin,/usr/
bin/,read,all,bandwidth,60MB/1s,50MB/1s,foo.bar.com,15392,0,/bin/
bash script.sh,/usr/bin/data,3,6,7
```

`High_create_lat,/tmp/,create,all,mean-latency,10ms`

The rule labelled "`High_create_lat`" is only concerned with function calls that create file system objects ("`create`") under "`/tmp/`". In this case the latency of each call made during the time frame is accumulated and averaged over the total number of these calls, provided the number of calls within the time frame is higher than the value of **`MISTRAL_MONITOR_LATENCY_MIN_IO`**.

If at the end of the time frame this "`mean-latency`" is higher than "`10ms`" then a log message will be output, for example:

```
2015-07-30T15:10.108650,High_create_lat,/tmp/,create,all,
mean-latency,22ms,10ms,foo.bar.com,15537,1,/bin/bash
script.sh,/tmp/data,3,6,7
```

`High_num_w,/home/,write,all,count,750`

The rule labelled "`High_num_w`" is violated if the number of write calls exceeds 750.

```
2015-07-30T15:10.108669,High_num_w,/home/,write,all,count,863,
750,foo.bar.com,15537,1,/bin/bash script.sh,/home/data,3,6,7
```

5.2 Throttling Contract

Consider the following contract:

```
2,throttletimeframe,1s
#LABEL,PATH,CALL-TYPE,SIZE-RANGE,MEASUREMENT,ALLOWED
High_reads,/usr/,read,all,bandwidth,5MB
High_reads_bin,/usr/bin/,read,all,bandwidth,1MB
High_num_r,/home/,read,all,count,750
```

Examining each line individually:

`2,throttletimeframe,1s`

This line identifies the contract as containing throttling rules that are applied over a time frame of 1 second.

`High_reads,/usr/,read,all,bandwidth,5MB`

If a monitored job were to try and read a 6MB file in `/usr/share/doc/` in less than a second, for example, this rule would be violated. When Mistral identifies an I/O operation that would violate a throttling rule it will introduce a sleep long enough to bring the observed I/O back down to the configured limit and a log message of the following form will be output:

```
2015-07-30T14:30.108355,High_reads,/usr/,read,all,bandwidth,1MB/
1s,1MB/1s,foo.bar.com,15392,0,/mnt/tool/bin/python
script.py,3,6,7
```

`High_reads_bin,/usr/bin/,read,all,bandwidth,1MB`

The second rule in this contract is very similar to the first. Again it is monitoring read bandwidth but this time is only interested in reads that occur under "**`/usr/bin/`**" and will allow up to "`1MB`" of data to be read before the rule is violated.

In this case all reads in "`/usr/bin/`" will be tested against both the "`High_reads`" and "`High_reads_bin`" rules as a read under "`/usr/bin/`" is also a read under "`/usr/`".

If a process were to attempt to read 3MB of data in "`/usr/bin/`" in less than 1 second it would violate the "`High_reads_bin`" rule but would not violate the "`High_reads`" rule. Therefore the process would be limited to "`1MB/1s`" and up to three log messages similar to the one described above will be output depending on the number of function calls that are used to read the 3MB of data.

If the process instead attempted to read 6MB of data in less than 1 second both the currently defined rules would be violated. Even though the rules are defined on different paths in this case the most restrictive rule applies and again the process will be throttled to "`1MB/1s`" and up to 6 log messages generated by violations of the "`High_reads_bin`" rule will be logged.

High_num_r,/home/,read,all,count,750

The third rule does not care about how large each operation is, it is simply interested in the total number of times a call is made to a "`read`" operation. If a total of more than "`750`" read operations are performed within the time frame of 1 second under "`/home/`" then on the 751st read Mistral would introduce a sleep long enough to bring the data rate under "`750/1s`" and a log message of the following form would be logged:

```
2015-07-30T16:45.108469,High_num_r,/
home/,read,all,count,750/1s,750/1s,foo.bar.com,16601,1,/usr/
lib64/firefox/firefox,/home/data,1,1,7
```

6 Plug-ins

Currently two different plug-ins are supported.

6.1 Update Plug-in

The update plug-in is used to modify local Mistral configuration contracts dynamically during a job execution run according to conditions on the node and / or cluster. Using an update plug-in is the only way to modify the local contracts in use by a running job.

Global contracts are assumed to be configured with high “system threatening” rules that should not be frequently changed. These contracts are intended to be maintained by system administrators and will be polled periodically for changes on disk as described above. Global contracts cannot be modified by the update plug-in in any way.

If an update plug-in configuration is not defined Mistral will use the same local configuration contracts throughout the life of the job.

6.2 Output Plug-in

The output plug-in is used to record alerts generated by the Mistral application. All event alerts raised against any of the four valid contract types are sent to the output plug-in. The four contract types are:

- Global Monitoring
- Global Throttling
- Local Monitoring
- Local Throttling

If an output plug-in configuration is not defined Mistral will default to recording alerts to disk as described above. In addition if an output plug-in performs an unclean exit during a job Mistral will revert to recording alerts to a log file. This log file will use the log record format expected by the plug-in to allow for simpler recovery of the data at a later date.

6.3 Plug-in Configuration

On start up Mistral will check the environment variable **`MISTRAL_PLUGIN_CONFIG`**. If this environment variable is defined it must point to a file that the user running the application can read. If the environment variable is not defined Mistral will assume that no plug-ins are required and will use the default behaviours as described above.

When using plug-ins, at the end of a job Mistral will wait for a short time, by default 30 seconds, for all plug-ins in use to exit in order to help prevent data loss. If any plug-in processes are still active at the end of this timeout they will be killed. The timeout can be altered by setting the environment variable **`MISTRAL_PLUGIN_EXIT_TIMEOUT`** to an integer value between 0 and 86400 that specifies the required time in seconds.

The expected format of the configuration file consists of one block of configuration lines for each configured plug-in. Each line is a comma separated pair of a single configuration option directive and its value. Whitespace is treated as significant in this file. The full specification for a plug-in configuration block is as follows:

```
PLUGIN,<OUTPUT|UPDATE>
INTERVAL,<Calling interval in seconds>
PLUGIN_PATH,<Fully specified path to plug-in>
[PLUGIN_OPTION,<Option to pass to plug-in>]
…
END
```

6.3.1 PLUGIN directive

The `PLUGIN` directive can take one of only two values, “UPDATE” or “OUTPUT” which indicates the type of plug-in being configured. If multiple configuration blocks are defined for the same plug-in the values specified in the later block will take precedence.

6.3.2 INTERVAL directive

The `INTERVAL` directive takes a single integer value parameter. This value represents the time in seconds the Mistral application will wait between calls to the specified plug-in.

6.3.3 PLUGIN_PATH directive

The `PLUGIN_PATH` directive value must be the fully qualified path to the plug-in to be run e.g. `/home/ellexus/bin/output_plugin.sh`. This plug-in must be executable by the user that starts the Mistral application. The plug-in must also be available in the same location on all possible execution host nodes where Mistral is expected to run.

The `PLUGIN_PATH` value will be passed to `/bin/sh` for environment variable expansion at the start of each execution host job.

6.3.4 PLUGIN_OPTION directive

The `PLUGIN_OPTION` directive is optional and can occur multiple times. Each `PLUGIN_OPTION` directive is treated as a separate command line argument to the plug-in. Whitespace is respected in these values.

As whitespace is respected command line options that take parameters must be specified as separate `PLUGIN_OPTION` values. For example if the plug-in uses the option
“`--output /dir/name/`” to specify where to store its output then this must be specified in the plug-in configuration file as:

```
PLUGIN_OPTION,--output
PLUGIN_OPTION,/dir/name/
```

Options will be passed to the plug-in in the order in which they are defined.
Each `PLUGIN_OPTION` value will be passed to `/bin/sh` for environment variable expansion at the start of each execution host job.

6.3.5 END Directive

The `END` directive indicates the end of a configuration block and does not take any values.

6.3.6 Invalid Configuration

Blank lines and lines starting with “#” are silently ignored. All other lines that do not begin with one of the configuration directives defined above cause a warning to be raised.

6.3.7 Example Configuration

Consider the following configuration file; line numbers have been added for clarity:

```
 1    # File version: 2.9.3.2, modification date: 2016-06-17
 2
 3    PLUGIN,OUTPUT
 4    INTERVAL,300
 5    PLUGIN_PATH,/home/ellexus/bin/output_plugin.sh
 6    PLUGIN_OPTION,--output
 7    PLUGIN_OPTION,/home/ellexus/log files
 8    END
 9
10    PLUGIN,UPDATE
11    INTERVAL,60
12    PLUGIN_PATH,$HOME/bin/update_plugin
13    END
```

The configuration file above sets up both update and output plug-ins.
Lines 1-2 are ignored as comments. The first configuration block (lines 3-8) defines an output plug-in (line 3) that will be called every 300 seconds (line 4) using the command line

```
/home/ellexus/bin/output_plugin.sh --output "/home/ellexus/log files"
```

(lines 5-7). The configuration block is terminated on line 8.

The blank line is ignored (line 9).

The second configuration block (lines 10-13) defines an update plug-in (line 10) that will be called every 60 seconds (line 11) using the command line `/home/ellexus/bin/update_plugin`, (line 12), assuming `$HOME` is set to `/home/ellexus`. The configuration block is terminated on line 13.

7 Scheduler Integration

7.1 IBM Spectrum LSF

7.1.1 Launcher script

Create a script that defines the required environment variables and any default settings, for example:

```
#!/bin/bash
INSTALL=/apps/ellexus

export MISTRAL_INSTALL_DIRECTORY=${INSTALL}/mistral_latest_x86_64
export MISTRAL_LICENSE=${MISTRAL_INSTALL_DIRECTORY}

# This script hard codes a simple global contract but the following
# lines can be replaced with whatever business logic is required to
# set up an appropriate contract for the submitted job.
export MISTRAL_CONTRACT_MONITOR_GLOBAL=${INSTALL}/global.contract
export MISTRAL_LOG_MONITOR_GLOBAL=${INSTALL}/global.log

# Set up the Mistral environment. As we are doing this automatically
# on LSF queues set Mistral to only manually insert itself in rsh and
# ssh commands to other nodes.
source ${MISTRAL_INSTALL_DIRECTORY}/mistral --remote=rsh,ssh
```

This script should be saved in an area accessible to all execution nodes.

7.1.2 Define a Job Starter

For each queue that is required to automatically wrap jobs with Mistral add a `JOB_STARTER` setting that re-writes the command to launch the submitted job using the script created above. For example if the script above has been saved in `/apps/ellexus/mistral_launcher.sh` the following code defines a simple queue that will use it to wrap all jobs with Mistral:

```
# Mistral job starter queue
Begin Queue
QUEUE_NAME   = mistral
PRIORITY     = 30
INTERACTIVE  = NO
TASKLIMIT    = 5
JOB_STARTER  = . /apps/ellexus/mistral_launcher.sh; %USRCMD
DESCRIPTION  = For mistral demo
End Queue
```

Once the job starter configuration has been added the queues must be reconfigured by running the command:

```
$ badmin reconfig
```

To check if the configuration has been successfully applied to the queue the `bqueues` command can be used with the "`-l`" long format option which will list any job starter configured, e.g.

```
$ bqueues -l mistral

QUEUE: mistral
  -- For mistral demo

PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN SSUSP USUSP  RSV
 30    0  Open:Active       -    -    -    -     0     0     0     0     0    0
Interval for a host to accept two jobs is 0 seconds

 TASKLIMIT
 5

SCHEDULING PARAMETERS
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

SCHEDULING POLICIES:  NO_INTERACTIVE

USERS: all
HOSTS:  all
JOB_STARTER:  . /apps/ellexus/mistral_launcher.sh; %USRCMD
```

7.2 OpenLava

7.2.1 Launcher script

Create a script that defines the required environment variables and any default settings, for example:

```
#!/bin/bash
INSTALL=/apps/ellexus

export MISTRAL_INSTALL_DIRECTORY=${INSTALL}/mistral_latest_x86_64
export MISTRAL_LICENSE=${MISTRAL_INSTALL_DIRECTORY}

# This script hard codes a simple global contract but the following
# lines can be replaced with whatever business logic is required to
# set up an appropriate contract for the submitted job.
export MISTRAL_CONTRACT_MONITOR_GLOBAL=${INSTALL}/global.contract
export MISTRAL_LOG_MONITOR_GLOBAL=${INSTALL}/global.log

# Set up the Mistral environment. As we are doing this automatically
# on OpenLava queues set Mistral to only manually insert itself in rsh
# and ssh commands to other nodes.
source ${MISTRAL_INSTALL_DIRECTORY}/mistral --remote=rsh,ssh
```

This script should be saved in an area accessible to all execution nodes.

7.2.2 Define a Job Starter

For each queue that is required to automatically wrap jobs with Mistral add a `JOB_STARTER` setting that re-writes the command to launch the submitted job using the script created above.

For example if the script above has been saved in `/apps/ellexus/mistral_launcher.sh` the following code defines a simple queue that will use it to wrap all jobs with Mistral:

```
# Mistral job starter queue
Begin Queue
QUEUE_NAME   = mistral
PRIORITY     = 30
INTERACTIVE  = NO
JOB_STARTER  = . /apps/ellexus/mistral_launcher.sh; %USRCMD
DESCRIPTION  = For mistral demo
End Queue
```

Once the job starter configuration has been added the queues must be reconfigured by running the command:

```
$ badmin reconfig
```

To check if the configuration has been successfully applied to the queue the `bqueues` command can be used with the "`-l`" long format option which will list any job starter configured, e.g.

```
$ bqueues -l mistral

QUEUE: mistral
  -- For mistral demo

PARAMETERS/STATISTICS
PRIO NICE     STATUS       MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SSUSP USUSP  RSV
  30   0    Open:Active      -    -    -    -     0     0     0     0     0    0

Interval for a host to accept two jobs is 0 seconds

SCHEDULING PARAMETERS
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

SCHEDULING POLICIES:  NO_INTERACTIVE

USERS:  all users
HOSTS:  all hosts used by the OpenLava system
JOB_STARTER:  . /apps/ellexus/mistral_launcher.sh; %USRCMD
```

7.3 Univa Grid Engine

7.3.1 Launcher script

Create a script that defines the required environment variables and any default settings, for example:

```
#!/bin/bash
INSTALL=/apps/ellexus

export MISTRAL_INSTALL_DIRECTORY=${INSTALL}/mistral_latest_x86_64
export MISTRAL_LICENSE=${MISTRAL_INSTALL_DIRECTORY}

# This script hard codes a simple global contract but the following
# lines can be replaced with whatever business logic is required to
# set up an appropriate contract for the submitted job.
export MISTRAL_CONTRACT_MONITOR_GLOBAL=${INSTALL}/global.contract
export MISTRAL_LOG_MONITOR_GLOBAL=${INSTALL}/global.log

# Set the shell we need to use to invoke the submitted command
shell=${SGE_STARTER_SHELL_PATH:-/bin/sh}
if [ ! -x $shell ]; then
    # Assume that if the check failed $shell was not set to /bin/sh
    shell=/bin/sh
fi

shell_name=$(basename $shell)
if [ "${shell_name: -3}" = "csh" ]; then
    suffix=.csh
fi

# Check if a login shell is required
if [ "$SGE_STARTER_USE_LOGIN_SHELL" = "true" ]; then
    logopt="-l"
else
    logopt=""
fi

# Wrap the job with Mistral. As we are doing this automatically on UGE
queues
# set Mistral to only manually insert itself in rsh and ssh commands to
other
# nodes.
exec ${logopt} ${shell} "${MISTRAL_INSTALL_DIRECTORY}/mistral$suffix" \
    --remote=rsh,ssh "$@"
```

This script should be saved in an area accessible to all execution nodes.

7.3.2 Define a Starter Method

For each queue that is required to automatically wrap jobs with Mistral add a `starter_method` setting that points to the script created above. For example if the script above has been saved in `/apps/ellexus/mistral_launcher.sh` in order to add it to the existing queue "`mistral.q`" type the command:

```
$ qconf -mq mistral.q
```

This will launch the default editor (either `vi` or the editor indicated by the `EDITOR` environment variable). Find the setting for `starter_method` and replace the current value, typically "`NONE`", with the path to launcher script. Save the configuration and exit the editor. For example the following snippet of queue configuration shows the appropriate setting to use the file described above.

```
epilog                NONE
shell_start_mode      unix_behavior
starter_method        /home/ellexus/ugedemo/launch.sh
suspend_method        NONE
resume_method         NONE
```

It is important to note that a `starter_method` will not be invoked for `qsh`, `qlogin`, or `qrsh` acting as `rlogin` and as a result these jobs will not be wrapped by Mistral.

To check if the configuration has been successfully applied to the `qconf` command can be used with the "`-sq`" option to show the full queue configuration which will list any starter method configured, e.g.

```
$ qconf -sq mistral.q
qname                 mistral.q
hostlist              @allhosts
seq_no                0
load_thresholds       np_load_avg=1.75
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
qtype                 BATCH INTERACTIVE
ckpt_list             NONE
pe_list               make
jc_list               NO_JC,ANY_JC
rerun                 FALSE
slots                 1
tmpdir                /tmp
shell                 /bin/bash
prolog                NONE
epilog                NONE
shell_start_mode      unix_behavior
starter_method        /home/ellexus/ugedemo/launch.sh
suspend_method        NONE
resume_method         NONE
terminate_method      NONE
notify                00:00:60
owner_list            NONE
user_lists            NONE
xuser_lists           NONE
subordinate_list      NONE
complex_values        NONE
```

```
projects              NONE
xprojects             NONE
calendar              NONE
initial_state         default
s_rt                  INFINITY
h_rt                  INFINITY
d_rt                  INFINITY
s_cpu                 INFINITY
h_cpu                 INFINITY
s_fsize               INFINITY
h_fsize               INFINITY
s_data                INFINITY
h_data                INFINITY
s_stack               INFINITY
h_stack               INFINITY
s_core                INFINITY
h_core                INFINITY
s_rss                 INFINITY
h_rss                 INFINITY
s_vmem                INFINITY
h_vmem                INFINITY
```

7.4 Slurm

7.4.1 TaskProlog script

Create a Slurm TaskProlog script that prints out the required environment variables and any default settings, for example:

```
#!/bin/bash
INSTALL=/apps/ellexus
MISTRAL_INSTALL_DIRECTORY=${INSTALL}/mistral_latest_x86_64

echo "export MISTRAL_INSTALL_DIRECTORY=$MISTRAL_INSTALL_DIRECTORY"
echo "export MISTRAL_LICENSE=${MISTRAL_INSTALL_DIRECTORY}"

# This script hard codes a simple global contract but the following
# lines can be replaced with whatever business logic is required to
# set up an appropriate contract for the submitted job.
echo "export MISTRAL_CONTRACT_MONITOR_GLOBAL=${INSTALL}/global.contract"
echo "export MISTRAL_LOG_MONITOR_GLOBAL=${INSTALL}/global.log"

# If MISTRAL_VARIANT is set in the environment, use a Mistral variant
library.
# This is required if you plan to use Mistral with MPI applications.
if [[ -z "${MISTRAL_VARIANT}" ]]; then
    echo "export LD_PRELOAD=$
{MISTRAL_INSTALL_DIRECTORY}/dryrun/lib64/libdryrun.so"
else
    echo -n "export LD_PRELOAD="
    echo "${MISTRAL_INSTALL_DIRECTORY}/dryrun/lib64/libdryrun-$
{MISTRAL_VARIANT}.so"
fi
```

This script should be saved in an area accessible to all execution nodes.

7.4.2 Update Slurm configuration

Configure Slurm to use the above TaskProlog script by adding the following line in your slurm.conf file:

```
TaskProlog=/path/to/mistral/taskprolog.sh
```

Each execution host requires the same TaskProlog setting.

Finally, instruct all Slurm daemons to re-read the configuration file:

```
$ scontrol reconfigure
```

Now all jobs submitted with sbatch, srun and salloc commands use Mistral.

7.4.3 Running Mistral on a specific Partition

Rather than running Mistral on all jobs, Mistral can be configured to run only on specific Partitions. Simply surround the example in 7.4.1 with an if statement comparing the $SLURM_JOB_PARTITION variable, for example:

```
#!/bin/bash
if [ "$SLURM_JOB_PARTITION" == "mistral" ]; then
      INSTALL=/apps/ellexus
      MISTRAL_INSTALL_DIRECTORY=${INSTALL}/mistral_latest_x86_64
      ...
      ...
      ...
fi
```

The Slurm configuration should then be updated as in 7.4.2.

Any jobs submitted on the 'mistral' partition will now run under mistral.

7.5 PBS Professional

7.5.1 Hook script

Create a PBS hook script (python) that inserts the required environment variables and any default settings into the job's environment. For example create a script called `hook.py` that contains:

```
import pbs
pbsevent = pbs.event()
jobname = pbsevent.job.queue.name

if jobname == "demo";
      install_dir = "/home/users/ellexus/mistral_latest_x86_64/"
      config_dir = "/home/users/ellexus/pbsconfig/"

      pbsevent.env["MISTRAL_INSTALL_DIRECTORY"] = install_dir
      pbsevent.env["MISTRAL_LICENSE"] = install_dir

      pbsevent.env["MISTRAL_CONTRACT_MONITOR_GLOBAL"] = config_dir + \
                        "global.contract"
      pbsevent.env["MISTRAL_LOG_MONITOR_GLOBAL"]= config_dir + \
                         "global.log"
      pbsevent.env["MISTRAL_PLUGIN_CONFIG"] = config_dir + \
                         "output_plugin.conf"
      pbsevent.env["LD_PRELOAD"] = install_dir + \
                         "dryrun/lib64/libdryrun.so"
```

This script should be saved in an area accessible to all execution nodes.

Now the hook needs to be setup. Create a hook named "job_starter" (can use any name) and import it:

```
$ qmgr -c "create hook job_starter event=execjob_launch"
$ qmgr -c "import hook job_starter application/x-python default /path/to/hook.py"
```

Now all jobs submitted with `qsub` use Mistral.

Note: Everytime the hook script is modified, it needs to be "imported" again using the `qmgr -c "import hook ..."` command above.