docs.azul

Unified Garbage Collection Logging Recommendations

Starting with Java 9, the unified logging infrastructure in Java Virtual Machine (JVM) provides access to log messages from different subsystems (compiler, gc, classload, etc.). See JEP 158: Unified JVM Logging for more information.

The unified logging system for all JVM components includes reimplemented GC logging which is described in JEP 271: Unified GC Logging. Azul Platform Prime 20.09.0.0 introduces support for unified GC logging in Azul Platform Prime 8, 11, and 13. Both sets of GC logging options - old and unified - are supported.

For non-detailed GC logging, use unified `-Xlog:gc` instead of the old `-XX:+PrintGC`. For detailed information, use unified `-Xlog:gc*` instead of the old `-XX:+PrintGCDetails`. A unified log message has a tag and a level set associated with it. To quickly see examples of `-Xlog` usage syntax, available tags, levels, decorators, and sample command lines with explanations, use:

```
-Xlog:help
```

Understanding `-Xlog:gc` Format

`-Xlog:gc` is a command-line option to perform unified GC logging using the Azul Platform Prime unified logging framework.

Syntax of the -Xlog option

```
-Xlog:gc:<output>:<decorators>:<output_options>
```

Where:

- `gc` is a tag to log GC related information.
- `<output>` specifies the target location for GC log messages.
- `<decorators>` are used to define information that log messages include.
- `<output_options>` are varied output options for logging additional information.

Unified logging is performed at six different levels that correspond to the amount of details included. You can choose messages to log based on the preferred level. The levels are listed in increasing order of verbosity:

- `off`
- `error`
- `warning`
- `info`
- `debug`
- `trace`

Enabling Unified GC Logging

`-XX:[+/-]UseGCUnifiedLogging` toggles unified GC logging. By default, unified GC logging is enabled in Azul Platform Prime.

To collect a GC log with basic information into a file, use: `-Xlog:gc:/path/to/file.log`

To enable more detailed logging, use `-Xlog:gc*`, such as: `-Xlog:gc*:/path/to/file.log`

The following example logs messages tagged with the `gc` and `safepoint` tags using `info` level to `gc.log`. By default, a GC log is written to `stdout`.

```
-Xlog:gc,safepoint:gc.log
```

To get more detailed information, use:

```
-Xlog:gc*,safepoint:gc.log
```

A sample unified GC log entry

```
[2.308s][info][gc] [Safepoint max times: safepoint 21.697 ms, time to safepoint 14.273 ms]
```

Where:

- `[2.308s]` is the time since the start of the Azul Platform Prime
- `[info]` is a default logging level
- `[gc]` is the name of the GC-specific tag

- `[Safepoint max times: safepoint 21.697 ms, time to safepoint 14.273 ms]` is a log message

Azul Platform Prime supports the following three output types:

- stdout
- stderr
- text file

To log GC messages to a file rather than to a console, use:

```
-Xlog:gc:file=<filename>
```

Where `file=` is optional and can be omitted.

Configuring Rotation of Large Log Files

Log rotation is enabled by default since JDK 11. In addition to the current log file, 5 rotated logs will be created. All files are limited to 20 MBytes size each.

Log file rotation based on file size and number of files is configured using the following command:

Change the number of rotated files and increase each file size:

```
-Xlog:gc,safepoint:gc.log::filecount=1,filesize=100M
```

Where:

- `gc,safepoint` tags to specify which information to log
- `::` empty decorators section, using the default uptime,level,tags
- `gc.log` the output file
- `filecount=1` specifies the number of files in the rotating set (current log + 1 archive file)
- `filesize=100M` size of each file in the log file rotation set

Disable log rotation and use an unlimited filesize:

```
-Xlog:gc,safepoint:gc.log::filecount=0
```

Note: `::` (double colon) is set here for the log line decorators to use the defaults.

Example with non-default log line decorators, adding the local time and date to each log line and changed amount and size of rotated files:

```
-Xlog:gc,safepoint:gc.log:time,uptime,level,tags:filecount=1,filesize=100M
```

On Prime, adding the local time and date is not needed as Prime writes time/date info into each file header and each log line is prepended by the seconds since JVM start. The external Azul Platform Prime GCLogAnalyzer tool automatically derives local time and date from that information.

Concatenating Tags

To include messages tagged with different tags in your GC log output, you can concatenate the tags with `+` as shown below.

A sample -Xlog:gc command for concatenating tags

```
-Xlog:gc+phases:gc.log
```

The resulting GC log contains all messages that feature the gc and phases tags:

```
[2.299s][info][gc] [SPS 0.000 0.000 0.001 0.178 0.005 0.000 0.008 0.087 0.142 0.420 NewGC_pause2
[2.299s][info][gc, phases] [New GC cycle 3-2: concurrent ref processing]
```

Additionally, you can specify a logging level by adding `=<level>` after the set of tags as follows:

```
-Xlog:gc+task+stats=debug:gc.log
```

This includes all messages with the gc, task, and stats tags at the debug logging level.

Configuring Multiple Logs

To simultaneously log messages from different subsystems to separate files, specify particular tags (or a combination of tags) and file names in the command line, for example:

```
-Xlog:gc:gc.log::filesize=1M -Xlog:gc+details:gcdetails.log::filesize=5M
```

This creates a `gc.log` file with GC related messages and a `gcdetails.log` file with detailed GC related messages.

A sample log entry with a gc tag

```
[5.010s][info][gc] [VM has exited gracefully]
```

A sample log entry with gc and details tags

```
[5.009s][info][gc,details] [Unmapping heap memory, 0 millis]
```

Inspecting GC Performance

With the safepoint tag, you can compare the performance of GC implementations over different JVMs as it produces similar log lines on all JVMs for pauses affecting application performance. The highest performance regarding latency is achieved when pauses are as short as possible.

A sample output on Azul Platform Prime

```
[105.482s][info][safepoint] Total time for which application threads were stopped: 0.0003760 seco
```

The snippet above shows `-Xlog:gc,safepoint:gc.log` output on Azul Platform Prime with a pause of 0.376 ms at 105 seconds after application start.

A sample output on OpenJDK

```
[55.350s][info][safepoint] Safepoint "CollectForAllocation", Time since last: 553327366 ns, Reach
```

The log entry above shows `-Xlog:gc,safepoint:gc.log` output on OpenJDK with a pause of 94.848 ms at 55 seconds after application start.

To extract the application stopped time from the gc.log written by OpenJDK or Azul Platform Prime, you can use the following shell command:

```
grep safepoint.*Total gc.log | sed -E 's/.*\[([0-9\.]+)s\].*Total[a-z ]*: ([0-9\.]+ [a-z]+).*/\1
```

Example output, with the 1st column showing the JVM uptime in seconds, the 2nd column the application stopped time:

```
76.810 0.0001696 seconds
78.695 0.0001152 seconds
82.780 0.0002665 seconds
84.587 0.0002310 seconds
```

This list of timestamps and stopped times can be loaded into a spreadsheet tool to create a graph.

Alternatively you can feed it through a statistical analysis, as most often the percentiles are more interesting regarding application performance SLAs. Here an example to collect the P50, P99, P99.9 and maximum and skipping the first 60 seconds of warmup time:

```
grep safepoint.*Total gc.log \
| sed -E 's/.*\[([0-9\.]+)s\].*Total[a-z ]*: ([0-9\.]+ [a-z]+).*/\1 \2/' \
| awk '$1 > 60 { print $1,$2,$3 }' \
| datamash -t' ' perc:50 2 perc:99 2 perc:99.9 2 max 2
```

Output example:

```
0.0002 0.0034 0.0035 0.0160
```

Interpretation: 50% of all GC pauses are below 0.2ms, 90% below 3.4ms, 99% below 3.5ms and all are below 16ms.

For Azul Platform Prime GC logs, you can also use Azul Platform Prime GCLogAnalyzer which displays performance-related metrics as graphs.

Mapping Old and Unified GC Logging Options

The table below contains a list of old GC logging command-line options and their unified GC logging equivalents.

Old GC Logging Option	Unified GC Logging Configuration
`PrintGC`	`-Xlog:gc`
`PrintGCDetails`	`-Xlog:gc*`
`PrintGCTimeStamps`	`-Xlog:gc`
`PrintGCDateStamps`	`-Xlog:gc:time,uptime,level,tags`
`PrintGCApplicationConcurrentTime`	`-Xlog:safepoint`
`PrintGCApplicationStoppedTime`	`-Xlog:safepoint`