java-gc

How to choose a java garbage collector

Default collector

Introduced with java5, jvm sets parameters (default heap size, jit, gc, etc) depending of the host computer caracteristics.
Caracteristics are OS, number of procs, memory:

- java 5 ergonomics
- java 5 performance white paper
- java 6 ergonomic (read java 5 ergonomics first)

Better than try to guess the class of computer, use the command:

```
$ java -XX:+PrintCommandLineFlags -version
```

- Results on windows 64bits (4Go) with jvm 64 bits:

  ```
  -XX:MaxHeapSize=1046199296 -XX:ParallelGCThreads=2 -XX:+PrintCommandLineFlags
  -XX:+UseLargePagesIndividualAllocation -XX:+UseParallelGC
  java version "1.6.0_16"
  Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
  Java HotSpot(TM) 64-Bit Server VM (build 14.2-b01, mixed mode)
  ```

  JVM is started in server mode and the default GC is parallelGC.

- Results on windows 64bits (4Go) with jvm 32 bits:

  ```
  -XX:+PrintCommandLineFlags -XX:+UseLargePagesIndividualAllocation
  java version "1.6.0_16"
  Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
  Java HotSpot(TM) Client VM (build 14.2-b01, mixed mode)
  ```

  JVM is started in client mode and the default GC is serialGC (by convention). The console doesn't log that the JVM is a 32 bits.

Garbage collectors

In short:

- serial GC is use with client jvms on not powerfull machine.
- parallel GC is en enhancement of serial GC that allow minor GC to be executed in parallel (by default major collection - aka full - is done in a stop-the-world way)
- concurrent mark and sweep allows a lot of gc process to be executed concurently with the application (some short work is done in stop-the-world way)

The 2 firsts GC are throughput collectors, CMS try to enhance response time with low pause.

Force a collector algorithm

Use one of the following parameter:

```
-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:+UseConcMarkSweepGC
```

Remark: the '+' is to activate. To deactivate, use '-': `-XX:-UseParallelGC`

Determine collector with gc logs

To display GC logs, add the following parameters on command line:

```
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps (or -XX:+PrintGCTimeStamps
that gives relative timestamp from application start)
```

To analyze gc logs, use the fork of GCViewer: https://github.com/chewiebug/gcviewer/wiki

Samples bellow use `-XX:+PrintGCTimeStamps` but it's better to use human-readable date with `-XX:+PrintGCDateStamps`.

- serial GC logs:

  ```
  7810.729: [GC 7810.729: [DefNew: 52144K->5506K(52480K), 0.0323370 secs]
  389229K->344757K(518528K), 0.0324120 secs] [Times: user=0.03 sys=0.00,
  real=0.03 secs]
  7816.865: [Full GC 7816.865: [Tenured : 339250K->99984K(466048K), 0.5083560
  secs] 361888K->99984K(518528K), [Perm : 20480K->20480K(20480K)], 0.5084270
  secs] [Times: user=0.50 sys=0.01, real=0.51 secs]
  27562.518: [GC 27562.519: [DefNew: 52735K->5823K(52736K), 0.0320420
  secs]27562.551: [Tenured: 466481K->116360K(466560K), 0.4671200 secs]
  516964K->116360K(519296K), [Perm : 20970K->20970K(20992K)], 0.4992810 secs]
  [Times: user=0.50 sys=0.01, real=0.50 secs]
  ```

  The last line is a minor GC that trigger a full GC (see archive)

- parallel GC logs:

  ```
  0.672: [GC [PSYoungGen: 1344K->480K(1664K)] 4678K->4314K(5760K), 0.0023071
  secs] [Times: user=0.02 sys=0.00, real=0.00 secs]
  0.674: [Full GC [PSYoungGen: 480K->22K(1664K)] [PSOldGen: 3834K->4095K(7360K)]
  4314K->4118K(9024K) [PSPermGen: 7270K->7270K(14592K)], 0.0293704 secs] [Times:
  user=0.00 sys=0.00, real=0.03 secs]
  ```

- CMS GC logs:

```
0.376: [GC 0.376: [ParNew: 13184K->790K(14784K), 0.0039676 secs]
13184K->790K(63936K), 0.0040756 secs] [Times: user=0.00 sys=0.00, real=0.00
0.705: [GC 0.705: [ParNew: 13974K->1600K(14784K), 0.0071664 secs]
13974K->2251K(63936K), 0.0072038 secs] [Times: user=0.03 sys=0.00, real=0.02
secs]
1.193: [GC 1.193: [ParNew: 14712K->870K(14784K), 0.0024345 secs]
22681K->8839K(63936K), 0.0024684 secs] [Times: user=0.01 sys=0.00, real=0.00
secs]
1.494: [Full GC 1.494: [CMS: 7969K->8698K(49152K), 0.0795908 secs]
13818K->8698K(63936K), [CMS Perm : 12286K->12273K(12288K)], 0.0797423 secs]
[Times: user=0.06 sys=0.01, real=0.08 secs]
1.972: [GC 1.972: [ParNew: 13184K->649K(14784K), 0.0020326 secs]
21882K->9347K(63936K), 0.0020638 secs] [Times: user=0.01 sys=0.00, real=0.01
secs]
```

Tuning jvm

- read link at the begining of this page
- This pdf is a good introduction to gc algos: http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf
- a tuning method from Atlassian: https://confluence.atlassian.com/display/ATLAS/Garbage+Collection+%28GC%29+Tuning+Guide
- read this blog: http://randomlyrr.blogspot.fr/2012/03/java-tuning-in-nutshell-part-1.html
- summary of GC options: http://www.tagtraum.com/gcviewer-vmflags.html
- another lits of options: http://blog.ragozin.info/2011/09/hotspot-jvm-garbage-collection-options.html
- reflection: excessive full GC : http://anshuiitk.blogspot.fr/2010/11/excessive-full-garbage-collection.html
- CMS
  - how it works: http://webcache.googleusercontent.com/search?q=cache:yy1uhbD68BwJ:blog.griddynamics.com/2011/06/understanding-gc-pauses-in-jvm-hotspots_02.html+&cd=4&hl=fr&ct=clnk&gl=fr&client=firefox-a
  - explanations: https://blogs.oracle.com/jonthecollector/entry/did_you_know
  - logs: https://blogs.oracle.com/poonam/entry/understanding_cms_gc_logs

Uses following parameters (just for tuning, disable it in production env):

```
-XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCApplicationStoppedTime
```

These options will instruct the Java process to print the time an application is actually running, as well as the time the application was stopped due to GC events. Here are a few sample entries that were produced:

```
$ egrep ‘(Application|Total time)’ gc.log |more
Application time: 3.3319318 seconds
Total time for which application threads were stopped: 0.7876304 seconds
Application time: 2.1039898 seconds
Total time for which application threads were stopped: 0.4100732 seconds
```