



De: Andreas Loew

from my point of view (10 years of experience with tuning Hotspot JVM in Sun PS) the CMS algorithm definitely is recommended.
Regarding OutOfMemoryError, you must not see this error with any GC algorithm, but an OOM as such does not mean you that have to change the GC algorithm, but rather that you either have a memory leak or are trying to store too much data into the JVM which exceeds the configured heap size (-Xmx).

I strongly recommend Charlie Hunt's new book for Hotspot GC tuning
http://www.amazon.com/Java-Performance-Addison-Wesley-Charlie-Hunt/dp/0137142528

as well as my « educated best guess » sample JVM settings for a Coherence cache node as a starting point:

`-server`

`-d64` (only for a 64-bit JVM with a max heap size of 32 GB)

`-verbose:gc` (and optionally `-Xloggc:`<your GC log target file>)

`-Xms`<heapsize>`m`

`-Xmx`<heapsize>`m`

`-XX:PermSize=`<permsize>`m` (optimally determine the Perm Gen size really needed by looking at jvisualvm with VisualGC plugin when all classes have been loaded)

`-XX:MaxPermSize=`<permsize>`m`

`-XX:NewSize=`<between heapsize/4 and heapsize/2>`m`

`-XX:MaxNewSize=`<between heapsize/4 and heapsize/2>`m`

`-XX:SurvivorRatio=2` (increase in even numbers up to 8 in case you note from looking at jvisualvm with VisualGC plugin that Survivor spaces are never filled up more than 60%)

`-XX:+HandlePromotionFailure`

`-XX:TargetSurvivorRatio=80` (somewhere between 60 to 80, decrease if you note direct Survivor space overflow)

`-XX:MaxTenuringThreshold=8`

`-XX:SoftRefLRUPolicyMSPerMB=5000` (« a soft reference will survive (after the last strong reference to the object has been collected) for 5000ms times the number of megabytes of free space in the heap »)

`-XX:+AggressiveOpts`

`-XX:+DoEscapeAnalysis`

`-XX:+UseCompressedOops` (only for a 64-bit JVM)

`-XX:+DisableExplicitGC`

`-XX:+UseParNewGC`

`-XX:ParallelGCThreads=<n` – number of cores you want to assign on average to this JVM on this machine> (default value: (ncores <= 8) ? ncores : 3 + ((ncores * 5) / 8))

`-XX:ParallelCMSThreads=<m` – number of cores you want to use for concurrent GC while the app continues to run – typically choose m between n/4 and n/3> (default value: (ParallelGCThreads + 3) / 4)

`-XX:+UseConcMarkSweepGC`

`-XX:CMSWaitDuration=<d` – where d needs to be larger than the maximum time between two minor Young Gen collection cycles> (from 6u27 onwards and in JDK7, you can set d = 0 instead to mean the same

`-XX:+CMSScavengeBeforeRemark`

`-XX:+CMSClassUnloadingEnabled`

`-XX:+ParallelRefProcEnabled`

`-XX:+CMSParallelRemarkEnabled`

`-XX:CMSInitiatingOccupancyFraction=75` (somewhere between 55 and 75, start with 75 and decrease in case CMS cannot keep up with the app)

`-XX:+UseCMSInitiatingOccupancyOnly`

`-XX:-TraceClassUnloading`

`-XX:+PrintTenuringDistribution`

`-XX:+PrintGCDateStamps`

`-XX:+HeapDumpOnOutOfMemoryError`

And be sure to use the most recent HP JDK 6.0.13 « (includes Oracle update 6u29) » which is available to be able to make use of all the new features (CompressedOops, EscapeAnalysis)…

## Author

- 

  Patrick

  GPM Factory