Garbage Collection

Table of contents

1 Introduction

For high transactional workloads, a significant component of The Grinder's response time can include the performance of the Java Garbage Collector (GC), which is a necessary component of the Java Virtual Machine (JVM) that The Grinder workers run on.

This page documents the improvements obtained by tuning garbage collection for a particular test configuration.

Gary Mulder

2 Testing

Comparison tests were performed for an identical complex test suite (5 million requests over 4 hours) with The Grinder deployed firstly on a single two quad core dual socket (i.e. 8 core) server with 12GB of RAM, and secondly on four dual core PCs with 4GB of RAM each (i.e. the same number of CPUs, but twice the sockets and so twice the memory bandwidth). All test variables were attempted to be controlled for, and the only significant change was The Grinder hardware used.

On the latter 4 PC configuration response times reported were 25% lower on average, and more significantly standard deviations of response times were 25% lower as well. The key changes between test scenarios were the change in JVM heap sizes from 1*8GB to 4*3GB, and the fact that four GCs were running simultaneously (i.e. one GC per JVM per PC). GC is very sensitive to memory bandwidth, so with four sockets (4 x 2 core) rather than two sockets (2 x 4 core) it is likely memory bandwidth for The Grinder was about doubled, which in turn reduced GC pause durations. Furthermore, with four GCs running simultaneously the times when The Grinder is subject to GC has been smoothed, which was directly reflected in the reduced response time standard deviations.

3 Conclusions

Conclusions are as follows. Your mileage may vary:

1. Use the Sun Hotspot JVM with CMS GC (not the Java 7 G1 GC which was observed to behave badly in some tests) with settings similar to the following (for 4GB dedicated PCs):

   ```
   grinder.jvm.arguments = -Xms3g -Xmx3g -XX:NewSize=2g -XX:MaxNewSize=2g
    -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseConcMarkSweepGC
    -XX:+UseParNewGC -XX:+ExplicitGCInvokesConcurrent -XX:+CMSConcurrentMTEnabled
    -XX:+AlwaysPreTouch
   ```

   `NewSize` is set proportionally large (67% of the heap size) as Jython seems to create a lot of short lived objects. CMS is used as it attempts to minimise GC pause times at the cost of transactional throughput. `Pretouch` is used to ensure the JVM is less likely to be paused waiting for memory pages from the Linux kernel.
2. Scale your Grinder clients horizontally (i.e. lots of cheap PCs) rather than vertically (i.e. big expensive multi-socket servers).
3. Keep a very close eye on the GC times reported by each Grinder's GC log. If The Grinder starts timing a request, pauses for GC, and then ends timing a request, some unknown amount of GC time will be added to the response time reported. GC times of 200ms are not uncommon, and GC pauses of 5 seconds can be produced by poorly

tuned GCs. Under Linux, to redirect GC logs from stdout invoke the Java worker as follows:

```
java net.grinder.Grinder $GRINDERPROPERTIES >> worker_out.log 2>&1
```

The `2>&1` also redirects (http://tldp.org/LDP/abs/html/io-redirection.html) any errors to `worker_out.log`.
To directly specify a GC log add the following JVM argument

```
-Xloggc:/tmp/gc_log
```

Make sure the JVM can write to the log file specified.