

[Pages / ... / Advanced Topics \(UV\)](#)

# Special considerations when running Java programs

Created by paul fretter (NBI), last modified on May 13, 2016

When running Java programs on a large system, such as the UV it is important to constrain the behaviour of the JVM Garbage Collection (GC). Please ensure that you set the following environment variable in your job script, before executing your Java program, which will tell the JVM to use only a single thread for garbage collection (this typically is satisfactory for most cases).

```
JAVA_OPTS="-XX:+UseSerialGC"
```

Failure to do so will result in the JVM spawning a very large number of processes for garbage collection, resulting in slow performance of your code and potentially a significant impact on other users of the system.

You can also manually set the parallel GC thread count using the following option:

```
JAVA_OPTS="-XX:ParallelGCThreads=2" # use 2 threads for GC
```

Example:

```
[fretter@UV2K3 test]$ qsub -I # let's be nice and do this
qsub: waiting for job 32865.UV00000010-P003 to start
qsub: job 32865.UV00000010-P003 ready

[fretter@UV2K3 test]$ cd test
[fretter@UV2K3 test]$ source jdk-1.7.0_25 # source the JDK wrapper
[fretter@UV2K3 test]$ javac HelloWorld.java # compile my java program
[fretter@UV2K3 test]$ java -XX:ParallelGCThreads=2 HelloWorld # execute HelloWorld with

[fretter@UV2K3 test]$ ps -ef | grep fretter | grep [j]ava # find my java process id
fretter 282390 256941 0 1970 pts/16 00:00:00 java -XX:ParallelGCThreads=2 HelloWorld

[fretter@UV2K3 test]$ jstack 282390 # get a thread dump; note
{... stuff deleted ...}
"VM Thread" prio=10 tid=0x00007fffff008d800 nid=0x44f1a runnable
"GC task thread#0 (ParallelGC)" prio=10 tid=0x00007fffff0016000 nid=0x44f18 runnable
"GC task thread#1 (ParallelGC)" prio=10 tid=0x00007fffff0018000 nid=0x44f19 runnable
"VM Periodic Task Thread" prio=10 tid=0x00007fffff00f8000 nid=0x44f21 waiting on condition
JNI global references: 110
```

Note: if using the -XX options on the java command line they must be specified BEFORE the name of your java program. Placing them after will not produce an error, but also they will not work!

## Background

When the JVM is launched it will enumerate all the CPU cores available on the system by reading /proc/cpuinfo. This is whether or not it actually has the ability to use them all, and will then spawn a number (N) of garbage collectors according to the following formula:

$$N = ( \text{ncpus} \leq 8 ) ? \text{ncpus} : 3 + ( ( \text{ncpus} * 5 ) / 8 )$$


In the case of the UV systems installed here, /proc/cpuinfo will report up to 1024 cores. Thus a very large number of GC threads will be spawned, which will contend with each other and will cause a general 'slow down' of the system, which will affect other users.

While the UV systems have the largest CPU count, this idiosyncrasy of Java GC is not limit to large systems. If not throttled this can also impact the performance of jobs when running on the SLURM cluster. This is because the SLURM scheduler will bind jobs to specific CPUs. In a scenario where we submit a single threaded job via SLURM and request a single CPU, if the job were to be scheduled onto a node with 32 CPUs it would result in the java application running all threads on the same CPU - this would be ~34 threads (23 GC threads + ~10 JVM worker threads + 1 main application). This results in all the threads competing for the single resource.

The following table shows the number of garbage collection threads to CPUs if not limit is imposed ...

Host CPUs	GC Threads
1	1
2	2
4	4
8	8
16	13
32	23
64	43
128	83
256	163
512	323
1024	643

If you require a different approach to garbage collection (object / heap management) please get in touch with the CiS Service Desk, or contact [your Institute's Scientific Computing support group](#).

 Like Be the first to like this

No labels

