Wednesday, June 9. 2010

About Java, parallel garbage collection and processor sets

Garbage Collection is still one of interesting topics in the Java world. There was and is a lot of development in this realm , lead from simple mechanisms over Parallel Garbage collection over Concurrent Mark Sweep and leads to G1. I don't want to talk about the simple mechanisms, i don't want to talk about the G1, i want to talk about the both in the middle, and i don't want to talk about the garbage collection itself. I'm a operating system guy, thus i want to look at them at the "damned, even more processes with fscking long process names" perspective.

On interesting question with those parallel and concurrent garbage collection mechanisms is: How many threads are running in parallel to do this job. There are an simple formula to do this computation. I've looked them up in the source code of JDK6u21.I found several different versions in blogs, thus i wanted some more authoritative.

Some mathLet's look at first at the number of threads for the ParallelGC. When you have less than or exactly 8 CPUS the formula is simple.

When you have more than 8 CPUs, it's getting a little more complex. (By doing so, they want to prevent you from ending up with 512 GC threads on a fully blown M9000)

The number of threads for the ConcurrentMarkSweep process is dependent on the number of the threads for the parallel GC.

The problemThose math is not a problem in itself. The problem arises when you have more than one running java process on your system. Because each of them does the same math and it thinks it's the only one on the system. One example. Let's assume you have an 8 core system, let's say you run 6 JDKs in it. Then you will have 48 ParallelGC threads and 12 ParallelCMS Threads alone. And that's a little to much.

The solutionsThere are two solutions to prevent the system to have too much GC threads. The first one is to directly intervene into this calculation and simply setting both values by hand. You can do this via -XX:ParallelGCThreads=n and -XX:ParallelCMSThreads=n. But i'm not a fan of such hard-coded things ... out of simple reasons ... people tend to forget them when they migrate them to different locations.

Another way is to leave the default mechanisms in action and using the single variable in the equation as an control instrument: The number of processors usable by Java. In Solaris you could use processor sets for this task. To get back to our 6 java processes on 8 procs. If you divide them into 6 processor sets with one proc each and leave the other 2 procs for housekeeping networking, interrupt and so on. each JDK will just start 1 ParallelGCThread and 1 CMS threads. Thus 6 ParallelGC Threads instead of 48 and 6 instead of 12 CMS threads. Okay, you could argue about the sense of ParallelGC/CMS here), but you should get the point.

Issues may look in the mirror more distant than they are.That isn't only a issues of the big boys with their 512 core systems. Think of the point that even a quad-socket Nehalem system with HT and hex-cores is 48-core system (the same for the 12-core quad socket Opteron-Systems) from the perspective of the os and thus from the perspective of Java.

Posted by Joerg Moellenkamp in English, Oracle, Solaris at 07:06

Good point if you run several Java processes.

But I feel that processor sets are too restrictive.
Is it possible to allocate a more flexible set of processors, e.g. 1-3 procs instead of fixed 1?
I guess I need some other resource control like projects then.

Will the java process detect these resource restriction from the current project?

Anonymous on Jun 9 2010, 10:42

It seems very strange formulas:
For instance, if we have 16 CPU's, then ParallelGCThreads=3 and ParallelCMSThreads=1

It must be an error in sign (instead - we should read +), so we'll have ParallelGCThreads=11 and ParallelCMSThreads=3
Anonymous on Jun 9 2010, 13:36

You are obviously right ... will correct that as soon i'm in reach of Internet for my notebook ...
Anonymous on Jun 9 2010, 13:40

Interesting. Can you point me some directions to docs on how to manage processor sets?
Anonymous on Jun 12 2010, 10:01

How exactly is nCPU determined? Especially with the CMT servers.. what will be used on e.g. a 5440? 2 (physical sockets) or 16 (physical cores) or 128 (virtual threads)?
Anonymous on Jun 15 2010, 10:08

When there is no processor set it will use the output of sysconf(_SC_NPROCESSORS_ONLN) systemcall. For a 5440 the answer should be 128.
Anonymous on Jun 15 2010, 13:51

BTW: in more recent JDK like 1.6.0_35 the option is called -XX:ConcGCThreads (but -XX:ParallelCMSThreads= is still accepted).
Anonymous on Oct 2 2012, 20:50