

Re: Game Company– Java Server Thread Priority

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2004-01/3461.html>

From: Chris Uppal (chris.uppal_at_metagnostic.REMOVE-THIS.org)

Date: 01/25/04

Date: Sun, 25 Jan 2004 10:31:54 -0000

[Over-enthusiastic cross-posting removed]

BlackHawke wrote:

- > *When we first set this up, the Game Server Program took exactly 50% of the*
- > *CPU cycles when idle, and up to 80% or 90% when working (processing a*
- > *turn, for example) and could process a turn of 10,000 ships in 15 seconds.*

50% CPU time on *IDLE* ?? What on earth was it doing ? Was it intended to do that ? If you don't already know the answers to these questions (and I *guess* that you don't or you'd have included them in you detailed description) then you *need* to find the answers...

- > *We revamped some of the major systems (the code and the computer itself),*
- > *did a fresh install of Windows 2003 on a different system (still dual*
- > *Xeon, but faster MB and CPU), migrated the Game Server Program there,*
- > *changed the threadpool priorities in the code, etc.*
- >
- > *We've now noticed that on idle (IE not processing turns), the Game Server*
- > *Program takes 0% of the CPU load, and when processing a turn goes up to*
- > *3-7%. It never exceeds 7%, and takes FOREVER now when processing a turn.*

If it's only using 0% CPU on idle now then that's what I'd expect, but it's not consistent with the first case. You should be able to find out the difference without considering what it's doing during its "real" workload. That should make the investigation easier. Also there is no way that changing, even, the entire OS is going to make you app go from 50% to 0% CPU to do *nothing*, so you can forget — for now — about possible OS links.

However, if the thing is running slowly, but is not consuming CPU then it *must* be waiting for something. I can think of two possibilities:

- network waits
- internal blocking

An example of network waits (this is a real example from my own past). An application was able to serve network requests at X/sec when the test workload was delivered from the same box. As soon as I generalised the test to run the

client on a colleague's box the workrate plummeted to around 1/20th, when I'd expected it to go *up* (since the server box wasn't running the client too). That turned out to be because of my logging code which logged the hostname of the client rather than the IP address -- reverse look up "localhost" was local and fast, but for some reason reverse lookup of my colleague's box always involved a network access (DHCP or something) which slowed it down massively. Of course, almost any kind of network access could show the same kind of problem, database access for instance.

Internal blocking could be almost anything. Perhaps you have over-"synchronised" something so that all processing is now happening on just one thread. Perhaps you are waiting for some conditions somewhere that never becomes true (or where the code for signalling that it has become true is broken), and so your app is halted until something times out, or otherwise wakes up the sleeping threads.

I would advocate sitting down and *thinking* about what's going on -- messing around trying to fix the symptoms (e.g. playing with task priorities) is just a waste of time. (I don't blame you though, it's hard to think clearly in what sounds like a panic situation). However, *while* you are thinking, it may be worthwhile running some profiling on the application, that should be able to find operations that are taking a long time without doing any real work.

HTH

-- chris