What Could Cause Software Hardware?

Asked 5 years, 8 months ago Modified 2 years, 9 mo

3

I have a Java OSGi (Apache Felix) applica
~975 packets/second (1038 octets in len
going across a boundary device, and trar
multiple threads involved and it's written
while to process one payload, it buffers i

When looking at the packet latency through this integration test scenario, two different desktop grade machines are significantly faster than the fairly high end servers we expect to deploy with.

- Server Latency 5 seconds. HW: Dual Xeon E5-2667v4@3.2GHz, 128G RAM, 16 physical, 32 logical cores, RAID 1 SAS SSDs.
- Desktop A < 1 second. HW Xeon E5-1620v4@3.5Ghz, 64G RAM, 4 physical, 8 logical cores, 500G SSD
- Desktop B < 1 second. HW i7-3770@3.4Ghz, 16G RAM, 4 physical, 8 logical cores, 1TB 7200RPM drive.

I only mention the hard drive for completeness as this application doesn't write to disk. On paper the server should perform at least as fast as the two desktops.

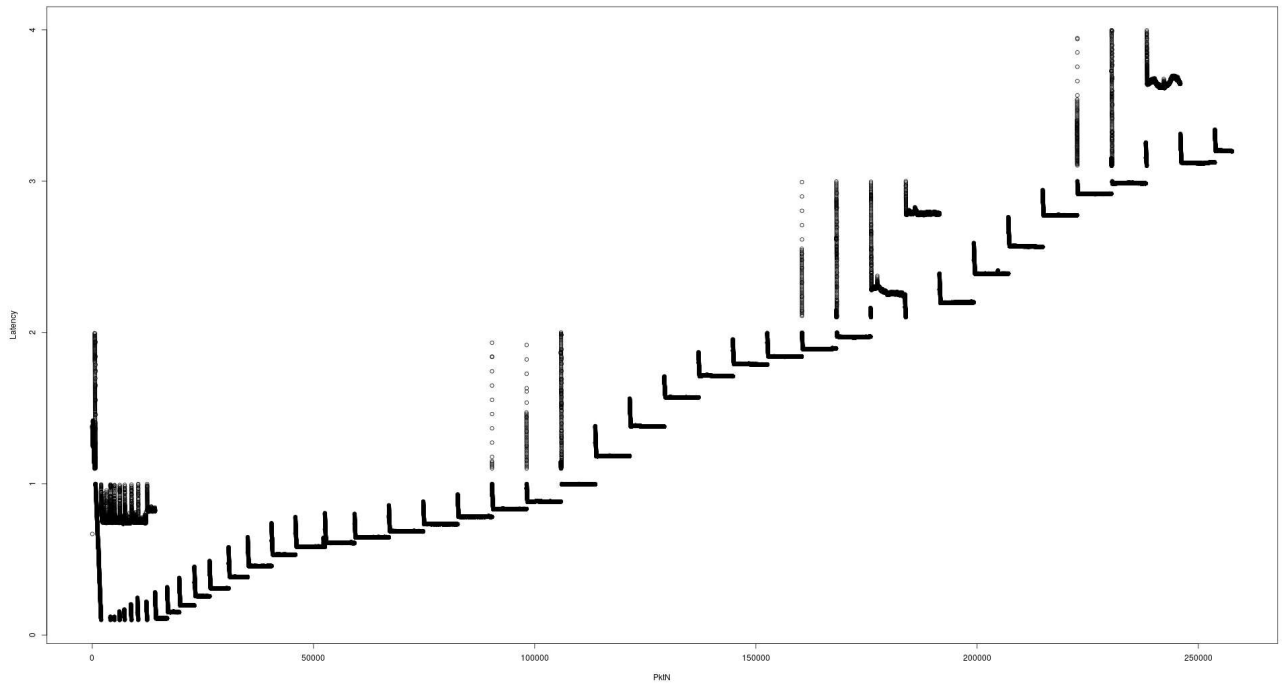
Things I've eliminated:

- Network cards. I've tested with both the physical NIC and the dummy device just in case there's significant differences between the NICs.
- Number of logical cores. I've tried disabling 16 and 24 of the servers logical cores in an effort to rule out variables.
- Java version. All three have been tried with both OpenJDK and Oracle's Java with identical versions (Java `1.8.0`) yielding the same results.
- Java flags are identical and all relate to felix (install directory, configuration properties, and jar to execute).
- SELinux. I've tried it in all three modes (disabled, enforcing, permissive). I didn't expect a difference here, but I'm grasping for anything at this point.
- Kernel Versions. I've tried the test against `3.10.0`, `4.13.0`, and `4.15.0` with similar results.

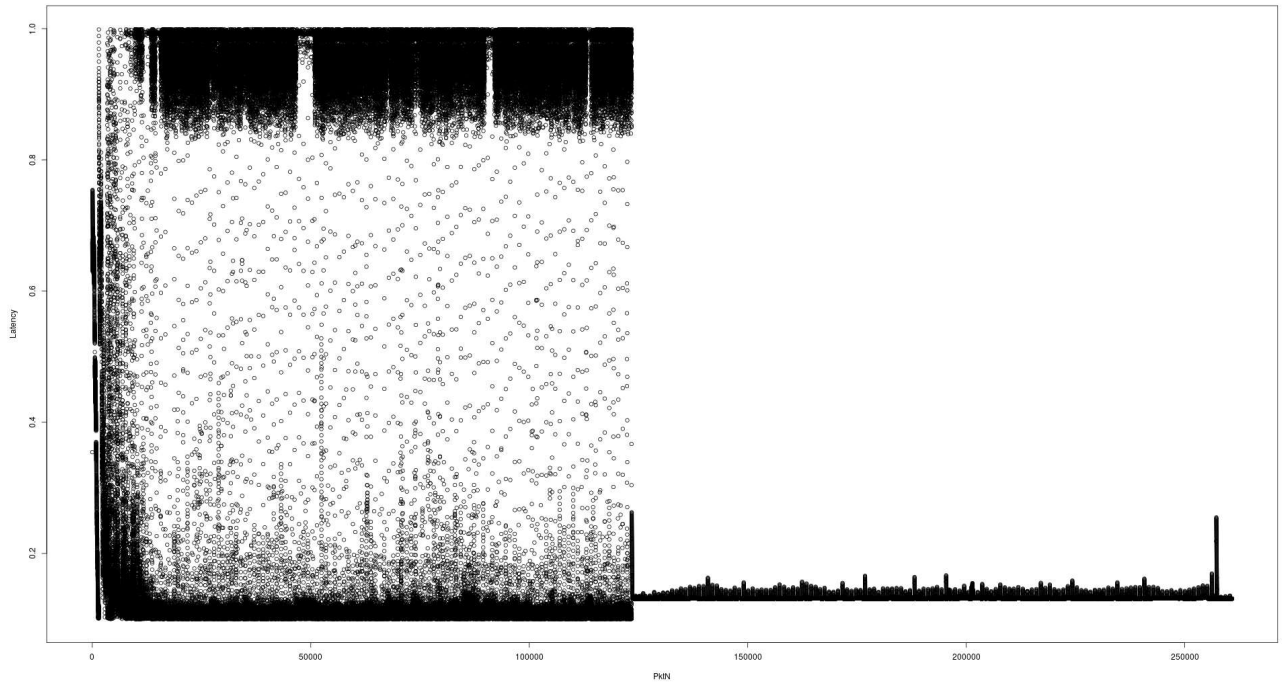
ark.intel.com processor comparison

Here's two sample graphs to illustrate the issue. This test sends 260,960 UDP packets across 4 minutes 10 seconds to multicast address A, and after it's been processed through the application, the packets are sent to multicast address B. `tcpdump` records the timestamps of both and subtraction yields the latency. All three applications (Sender, Application, `tcpdump` are on the same machine).

First the server hardware against the dummy interface



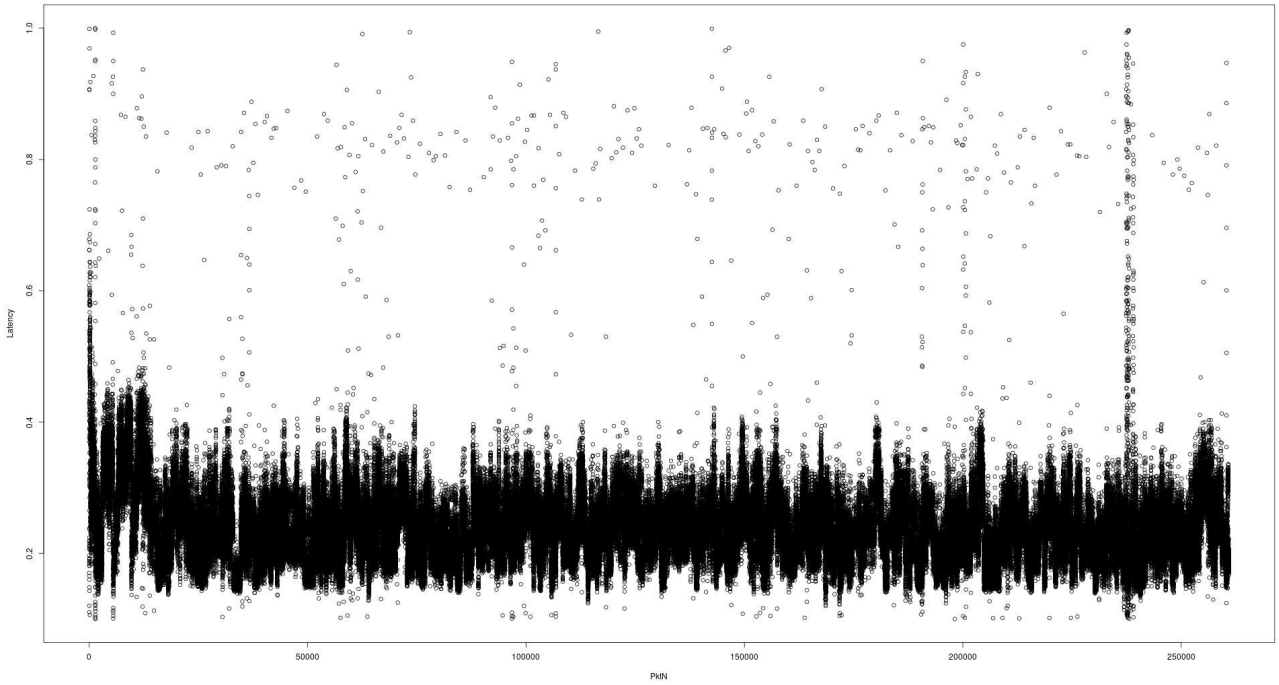
i7 Desktop hardware against the dummy interface



Note the Y axis scale difference. Server is 0-4 seconds, i7 Desktop is 0-1 seconds. The X axis which appears difficult to read is Packet Number.

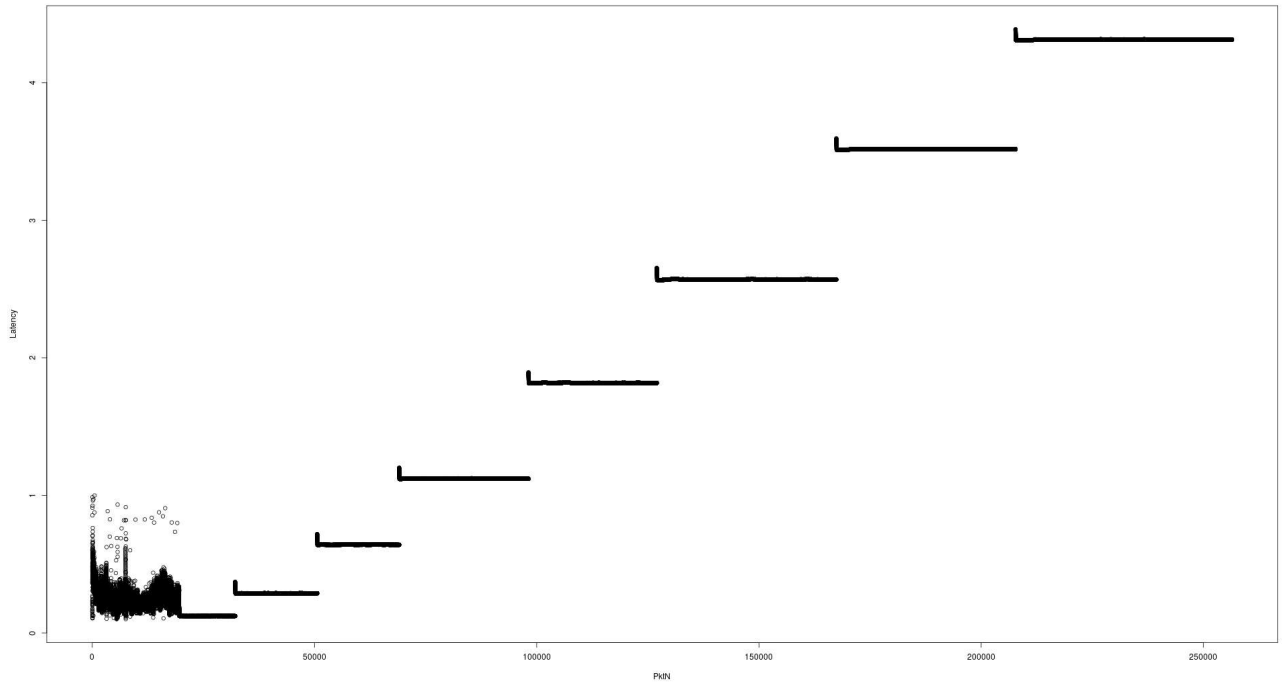
Next attempt

I was running a local integration verison of the application. I then eliminated almost 100% of the work begin done by the application and saw growing latencies on the server hardware. I then tried `-Xmx100G -Xms100G` essentially to keep the garbage collector from running EVER and saw the following results (< 1 second consistent latency).

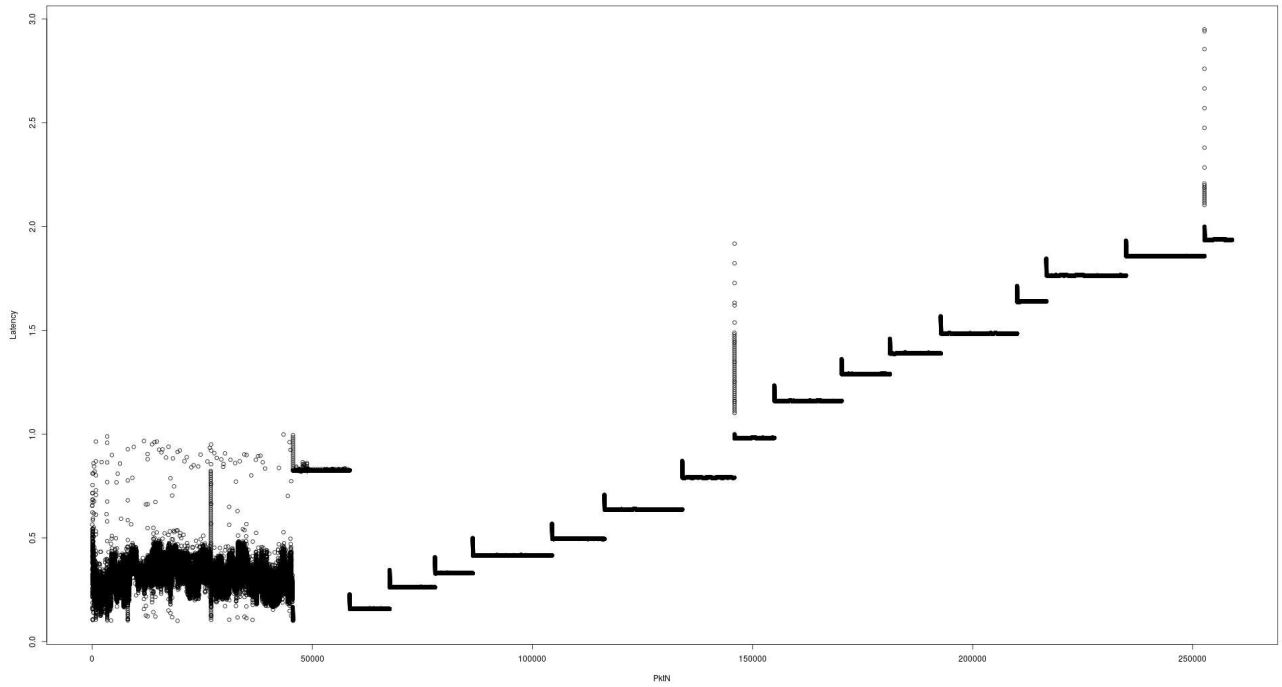


Which led me to Java 8's Available Garbage Collectors.

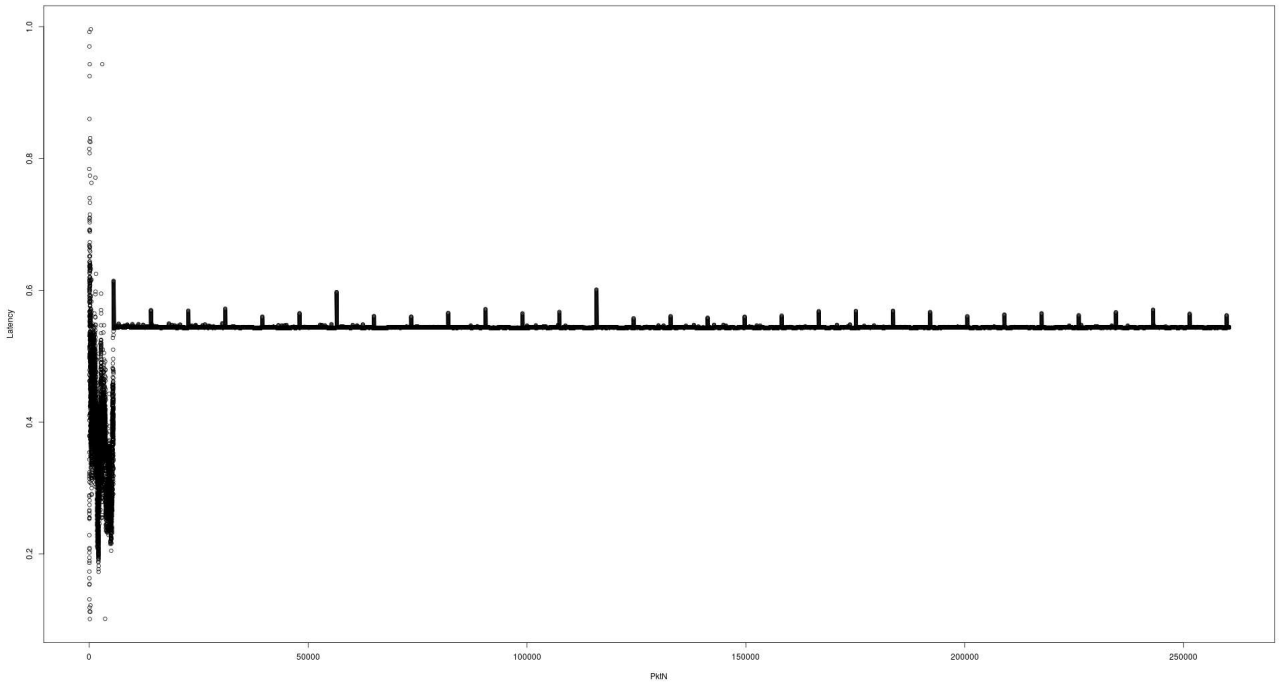
Default Garbage Collector select on the server hardware was New: ParallelScavenge, Old: ParallelOld. Here's the resulting latency graph without the XML conversion, as simple a test as I could make it to duplicate the issue.



Explicitly selecting the Garbage First Garbage Collector `-XX:+UseG1GC` selected New: G1New, Old: G1Old and it's resulting latency graph wasn't great:



Explicitly selecting the Concurrent Mark Sweep Garbage Collector `-XX:+UseConcMarkSweepGC` selected New: ParNew, Old: ConcurrentMarkSweep and it's resulting latency graph looked excellent:



It appeared like the problem was solved. Once I added all the components back into place, I'm still getting unacceptable latencies. I'm still running tests to see if I can isolate the issue.

Strace Results

Trying `strace -c -o /path/to/file -f` yielded the following top system calls

First the i7's desktop `strace` report (truncated at the top 10 items)

% time	seconds	usecs/call	calls	errors	syscall
93.71	1418.604132	959	1479659	134352	futex
1.74	26.294223	730395	36		poll
1.74	26.288786	314	83645	4	read
1.41	21.373672	73	293618		epoll_pwait
1.19	17.952475	120	149854	2	recvfrom
0.10	1.448453	2	909731		getrusage
0.06	0.896903	3	281407		sendto
0.03	0.394695	2	198041		write
0.01	0.182809	10	18246		mmap
0.01	0.120735	6	20582		sched_yield

Now for the server's `strace` report:

% time	seconds	usecs/call	calls	errors	syscall
97.46	2119.311196	2642	802183	131276	futex
1.28	27.734136	6933534	4		poll
0.59	12.840448	49	263597		epoll_wait
0.41	8.885742	113	78387	2	recvfrom

```
0.07    1.575401           6    263671           sendto
0.07    1.515999           6    262256           epoll_ctl
0.04    0.902788          54     16800           sched_yield
0.03    0.743231          10     75455           write
0.02    0.490052           6     84509         7 read
0.01    0.170152           4     42732           lseek
```

I'm unclear what I should conclude from this. The desktop is many times faster in both the `futex` and the `poll` system call. I still don't understand why the application is so much more latent on the faster hardware.

Profiling

I've profiled the software on both pieces of hardware showing similar locations for hotspots which seems to rule that out.

java osgi performance-testing multicast latency

Share Improve this question Follow edited Jun 16, 2021 at 9:48

asked Jul 31, 2018 at 13:42

Rob Paisley
447 1 3 13

I'm assuming environment flags are the same on both. A shot in the dark: have you tried looking into how JIT works on either (i.e. whether it doesn't on the server for whatever reason)? Have you tried using Flight Recorder or something similar to see where the latency could be coming from? – M. Prokhorov Jul 31, 2018 at 14:12

I ran with only felix related java options, so the flags should be the same. I'll look into both JIT / Flight Recorder now to see if there's something striking. – Rob Paisley Jul 31, 2018 at 14:26

Is the memory allocated to the JVM crossing the threshold where it no longer uses compressed pointers? You can actually take a significant performance hit at that point, since less content fits into cache. (Yes, you're specifying that the JVM flags are the same, but you aren't specifying what those flags *are*, so we don't know which configuration is explicit and which is inferred from hardware). – Charles Duffy Jul 31, 2018 at 14:40

...really, though, this is the point where I'd be pulling out (commercial) JVM profiling tools (and maybe also sysdig, to see if there's a difference in OS-level syscall responsive times). – Charles Duffy Jul 31, 2018 at 14:43

Java flags are defaulted other than to tell felix where to get configuration/properties. How do I tell if the memory allocated to JVM has crossed this threshold? Further, the server version has 2.5x the cache of desktop A which makes it less likely, but I'd like to rule EVERYTHING out at this point. I'll look into sysdig. – Rob Paisley Jul 31, 2018 at 14:46

1 Answer

Sorted by: Highest score (default)

1

I confirmed I was using the `performance` CPU governor with RedHat: CPUfreq Coverners

I ran across a VMWare ESXi report of problematic BIOS settings Virtual Machine Application runs slower than expected on EXSi

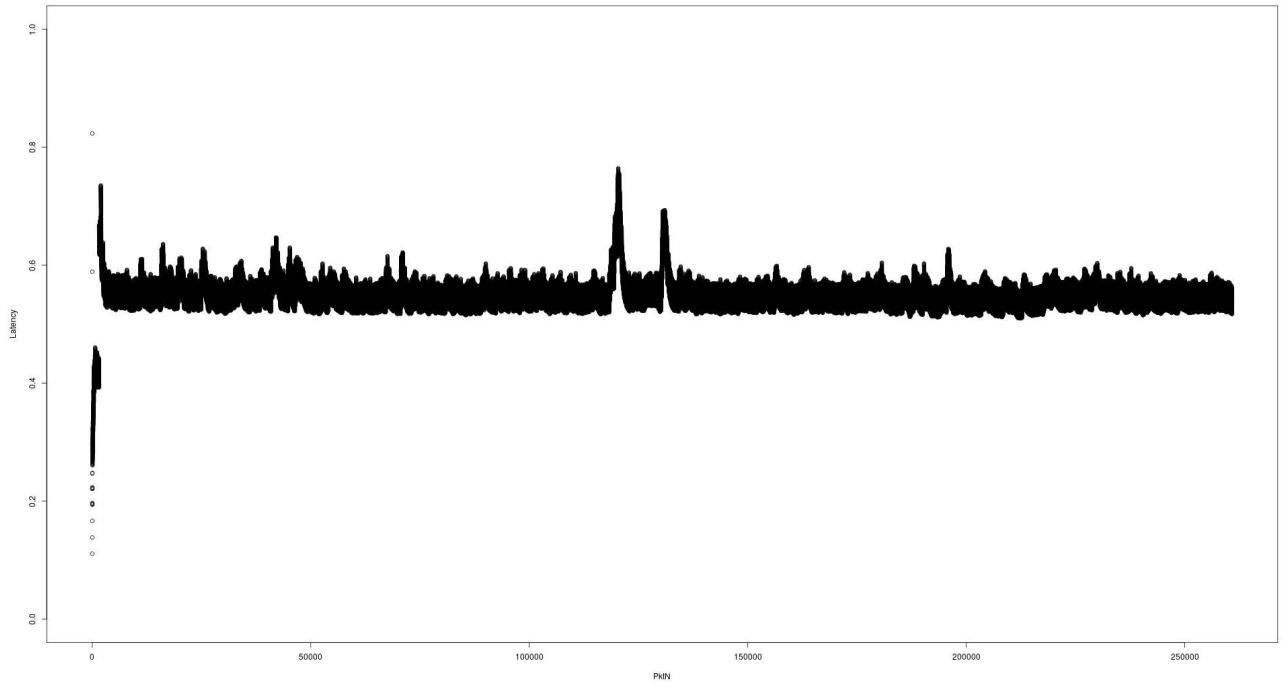
Which pointed directly to my answer. The default on this Dell R630 was "Performance Per Watt (DAPC)" (DAPC: Dell Active Power Controller). Switching to "Performance" fixed this issue entirely. The machine felt much snappier at the console, and latencies were much lower than the desktop was able to achieve which was what I expected given the CPU differences.

Steps to change the BIOS on a Dell R630 (and likely others) on startup:

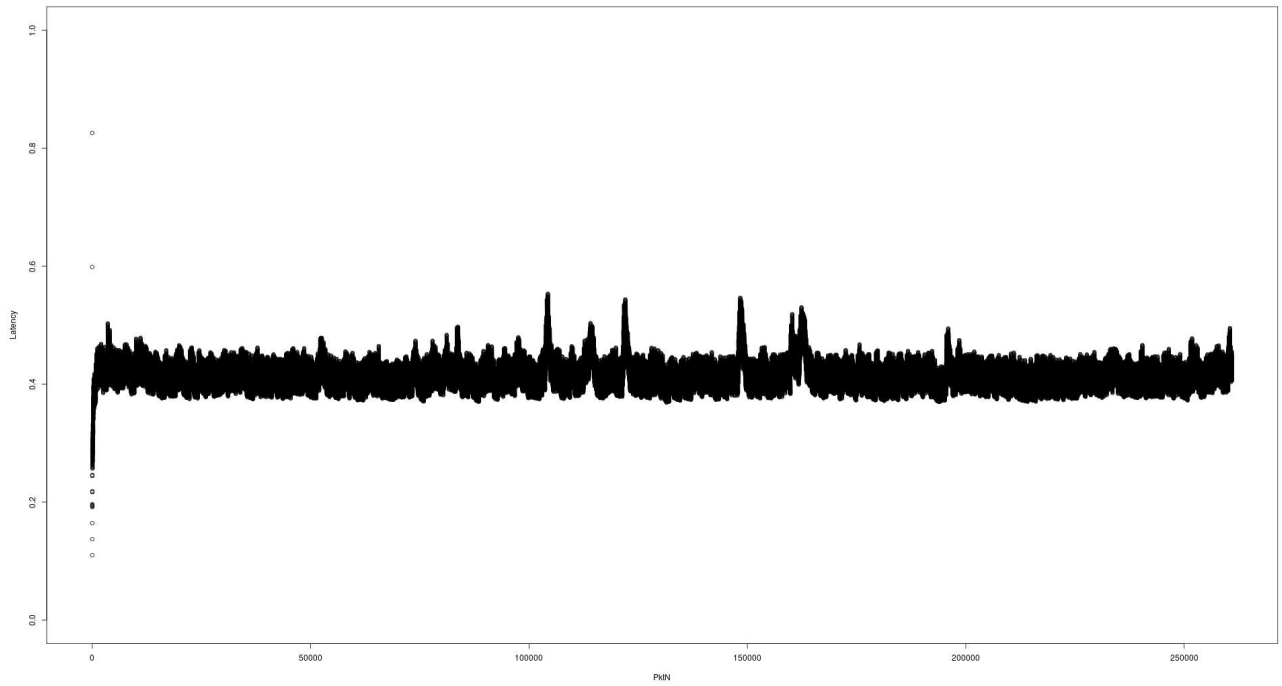
1. F2 to enter System Setup
2. Select "System BIOS"
3. Select "System Profile Settings"
4. Ensure first entry is set to "Performance" default is "Performance Per Watt"
5. Select "Back"
6. Select "Finish"
7. Select "Yes" to save changes with system reset
8. Select "OK" to the settings were saved successfully

Here's the resulting latency graph(s), they're using the same 1 second scale.

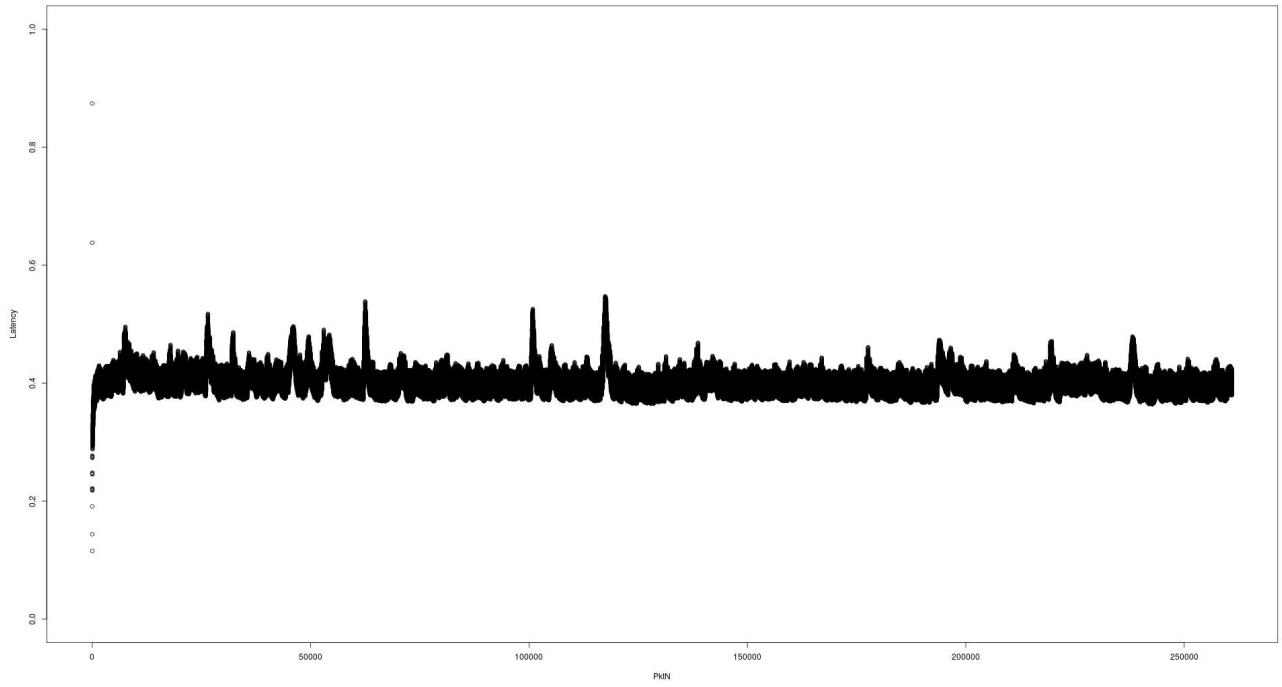
Default GC on the server(s):



Concurrent Mark Sweep GC on the server(s):



First Generation GC on the server(s):

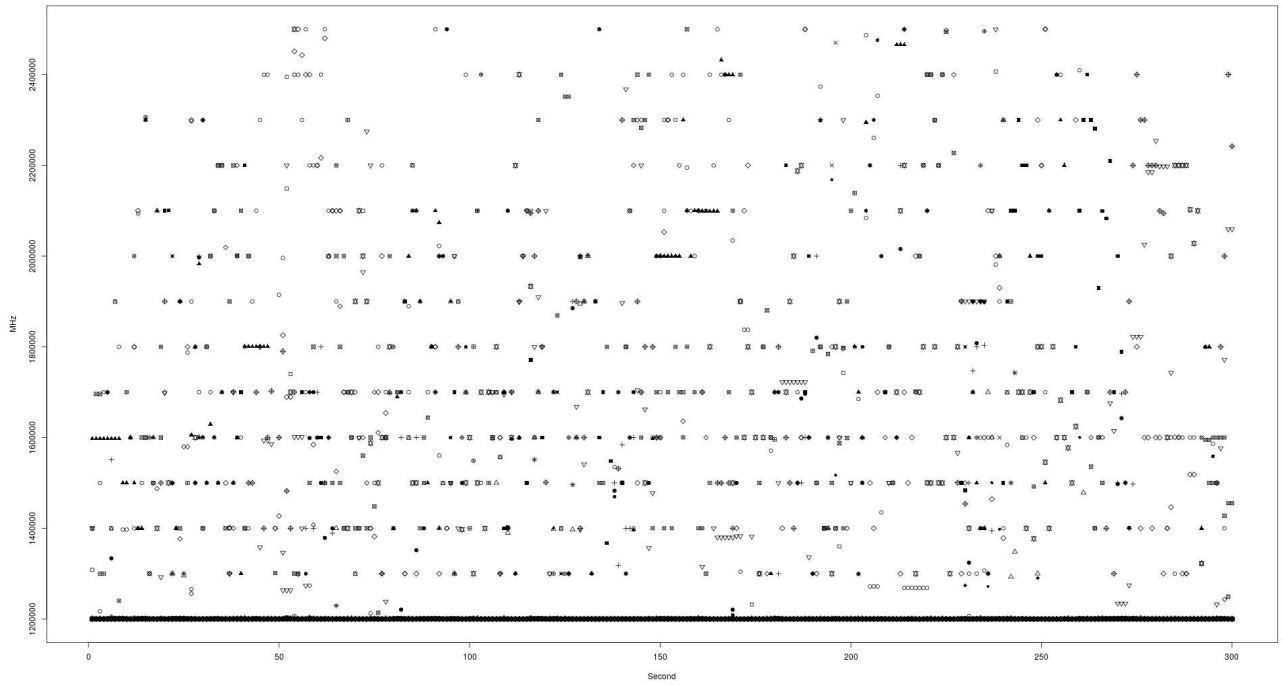


Not much difference between the G1GC and the CMSGC, but both are clearly better latency than the default (which is expected).

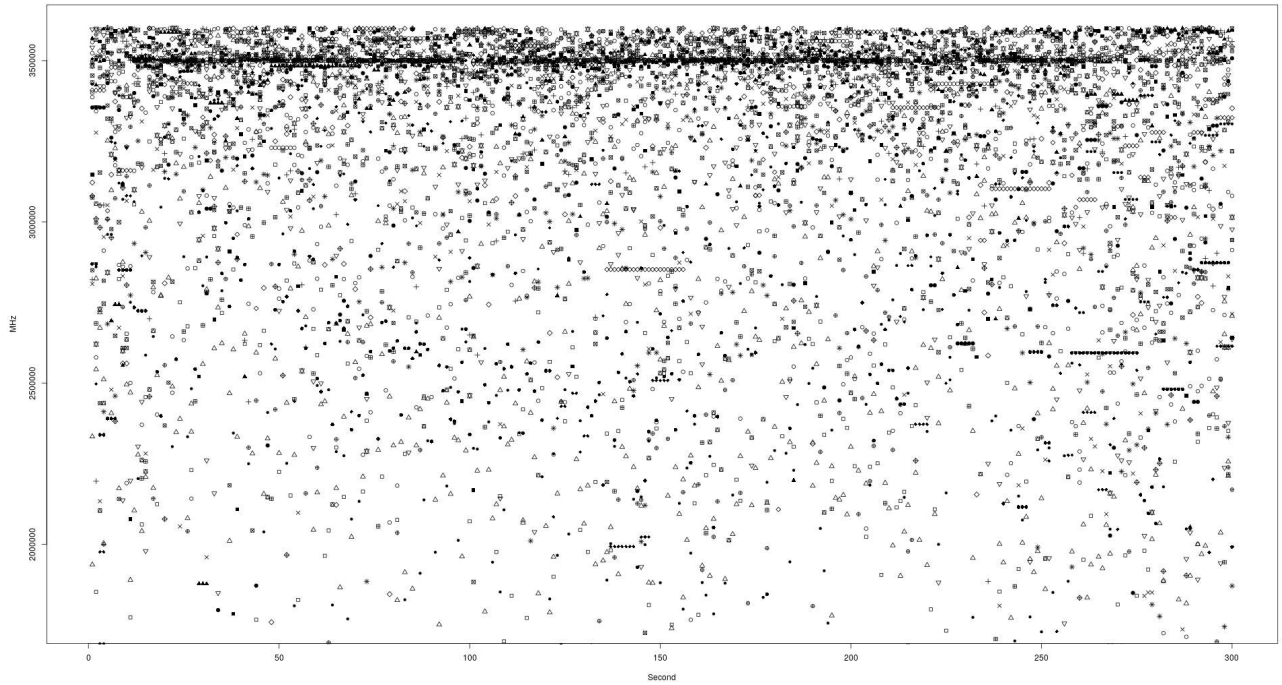
Graphs of Logical Core Clock Speed

Symbols are difficult to see, but there's 32 different points on these two graphs. Overall you can quickly tell which one was performance, and which one was performance-per-watt-dapc.

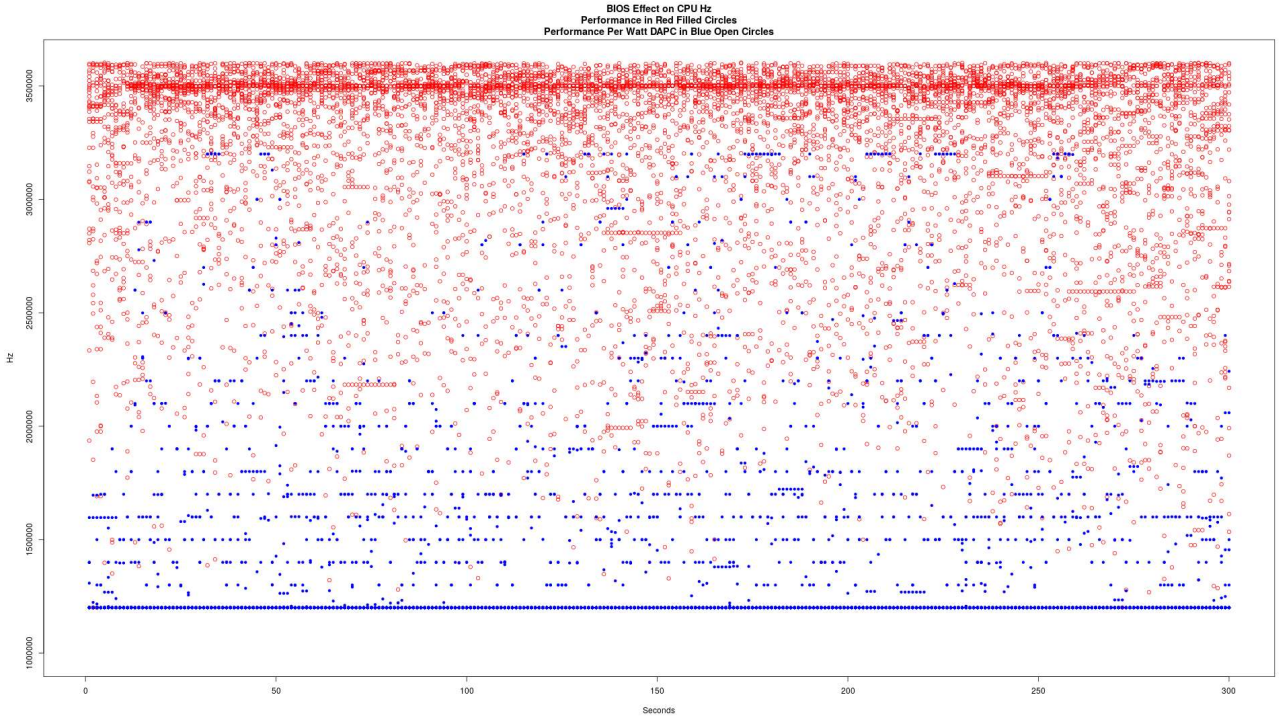
Performance Per Watt (DAPC):



Performance



Plotted Together. Performance in Red Bullets, Performance Per Watt in Blue Open Circles



This was captured during 300 seconds of data flow with the BIOS set accordingly. Here's how I captured the data in case anyone wants to know:

```
for i in `seq 300`; do
  paste /sys/devices/system/cpu/cpu[0-9]*/cpufreq/cpuinfo_cur_freq
  sleep 1
done > performance.log
```

Share Improve this answer Follow edited Aug 27, 2018 at 15:07 answered Jul 31, 2018 at 19:23

Rob Paisley
447 1 3 13

Great work, thanks for sharing this (I'm not certain SO is the ideal forum, but it's some solid sleuthing regardless). As an aside, I'd suggest `for ((i=0; i<300; i++)); do paste /sys/devices/system/cpu/cpu[0-9]*/cpufreq/cpuinfo_cur_freq; sleep 1; done >performance.log` -- that way you're opening the output file just once, vs 300 times. – Charles Duffy Aug 13, 2018 at 12:11

You're right the redirect should have happened at the done point in the loop, good catch. – Rob Paisley Aug 13, 2018 at 13:16