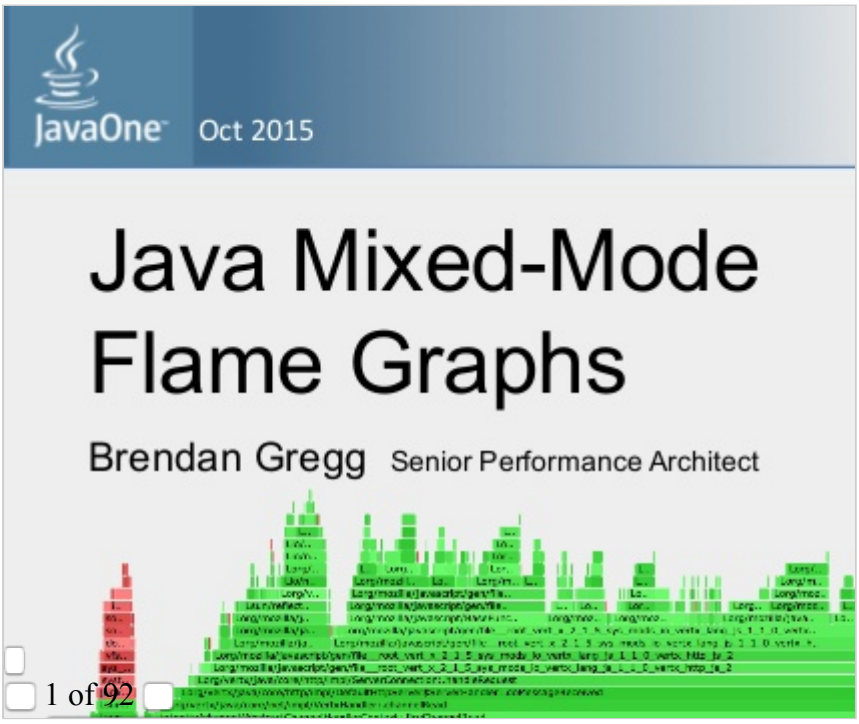
Java Mixed-Mode Flame Graphs at Netflix, JavaOne 2015

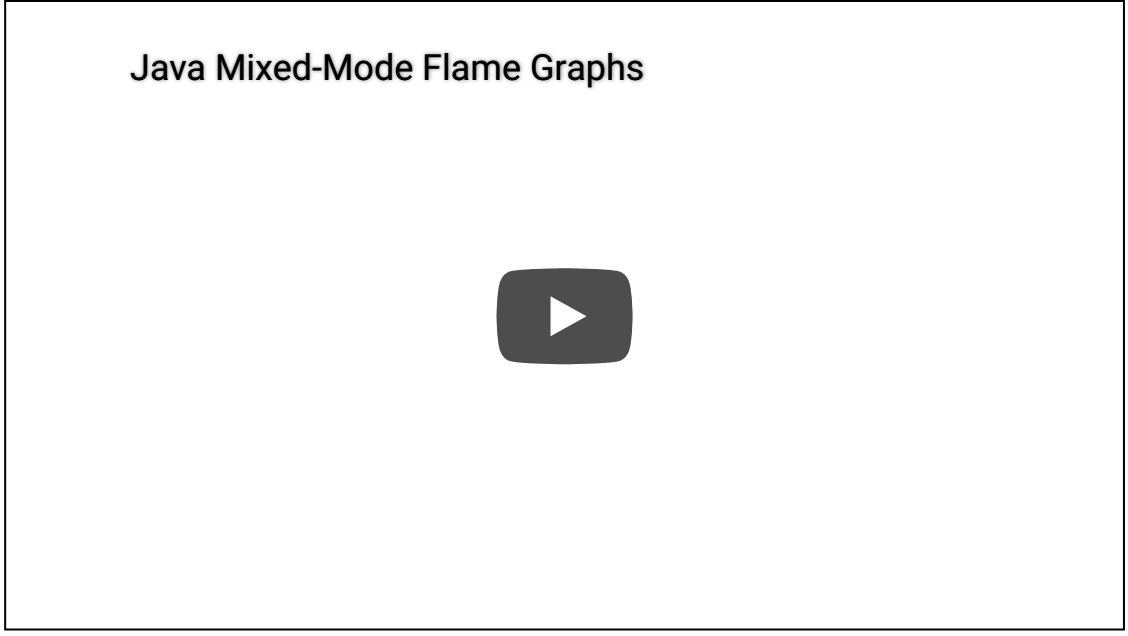
06 Nov 2015

At JavaOne this year I gave a talk on Java mixed-mode flame graphs, which we are rolling out at Netflix for CPU analysis and more. These make use of a new feature in JDK8u60: -XX:+PreserveFramePointer, which allows system profilers, including Linux perf_events, to capture stack traces.

The slides are on slideshare:



The talk was not recorded, however, low quality audio and video was captured, which is on youtube (the screen capture missed the demos, so they were taken from a handheld video):

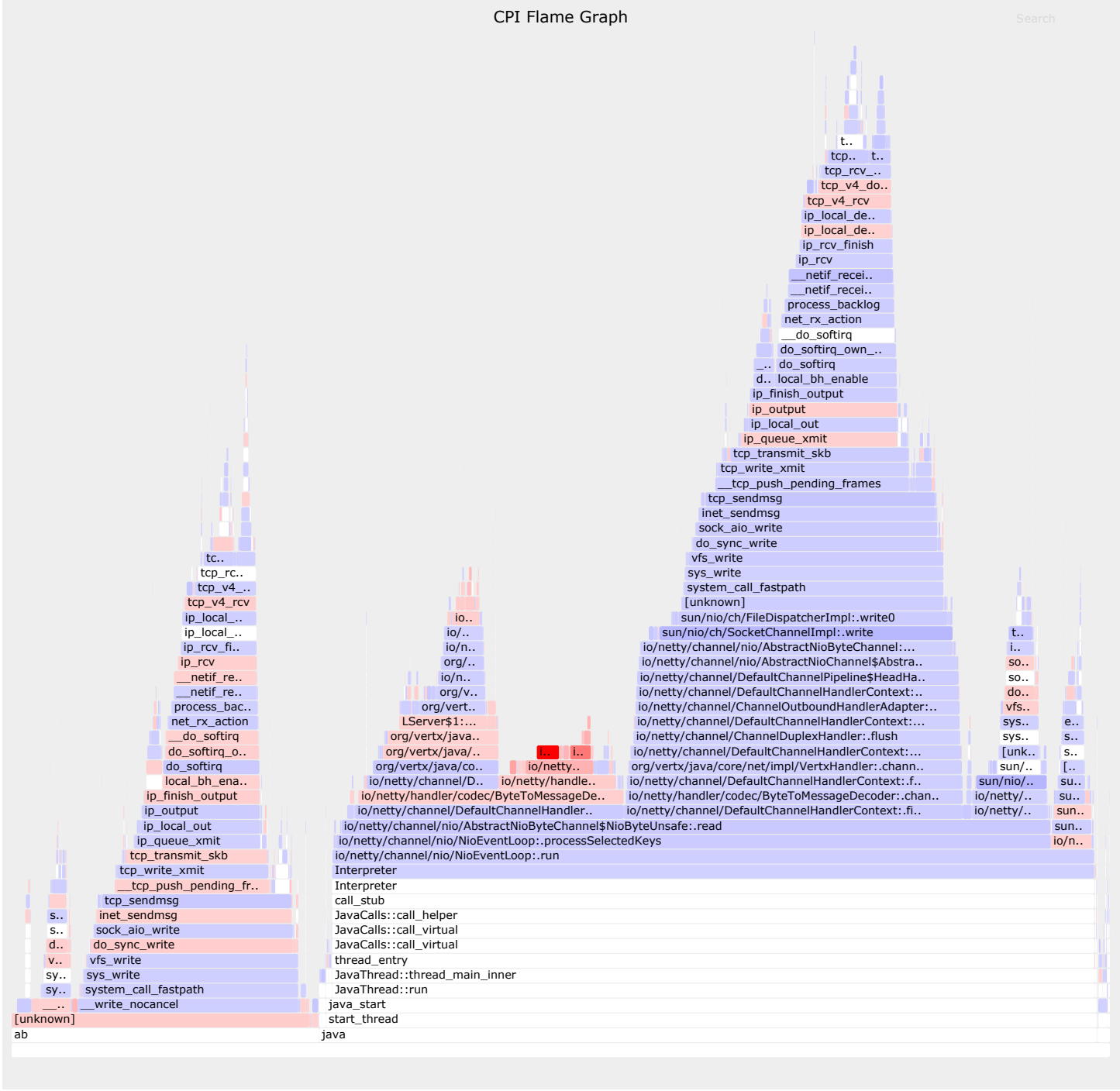


These flame graphs can be generated entirely with open source software, and I included the instructions in the talk. As it requires a number of steps, you'll likely want to automate it so that it becomes push-button. We've been doing that for our open source Vector instance analysis tool.

In this talk I introduced some new types of Java flame graphs, that visualize stack traces leading to:

- **Page faults**: To show what Java or JVM code triggered main memory (resident memory) to grow. This explains growth in the "RES" column you see in top(1).
- **Context switches**: To show why Java is blocking: what code paths lead to Java leaving the CPU. This can identify locks, I/O, sleeps, and other events. (Caveat: if it includes some random paths that shouldn't block, the reason may be kernel involuntary context switches due to CPU demand: however, you should already have identified those through basic CPU monitoring before using a context switch flame graph!).
- **Disk I/O requests**: To show Java and system code paths that lead to issuing a disk I/O request.
- **TCP events**: To show Java code paths that lead to initializing TCP sessions with connect() or accept(), or sending packets (receiving TCP packets happens asynchronously, so Java stack traces aren't present; they can be identified by moving up to the socket layer and syscalls if necessary).
- **CPU cache misses**: To show Java and JVM paths that lead to last level cache (LLC) misses, visualizing physical memory access by Java. Similar flame graphs can also be generated for memory stall cycles, helping the developers locate where to improve memory access (techniques for this were discussed in other sessions at JavaOne both this and last year).
- **CPI flame graph**: To show a CPU flame graph decorated with cycles-per-instruction as a color dimension. You might be more familiar with instructions-per-cycle (IPC), which is just the same metric inverted.

I was most excited to get the CPI flame graph working, which is the first time they've worked on Linux. An example flame graph is below (click to zoom, or view the SVG):



Red paths are instruction heavy (low CPI), blue paths are stall cycle heavy (likely memory stalls; high CPI). If you zoom in on the Java code, you can find the function AbstractByteBuf:.forEachByteAsc0 is instruction heavy (shown as red), and many other paths, especially involving reading and writing memory, are more stall cycle heavy (shown as blue). I'd previously blogged about CPI flame graphs on FreeBSD, which we've used for analysis on the Netflix Open Connect Appliances (our CDN).

For more background on these new types of Java profiling, see my Java in Flames post on the Netflix Tech Blog. For flame graphs in general, see my Flame Graphs page and the CPU Flame Graphs examples.

Thanks to the conference organizers for having me, and those that attended my talk. There was a lot of good technical content at JavaOne, and I'm still catching sessions I missed from the live stream recordings. Hopefully they will be eventually organized like the JavaOne 2014 sessions, making them easy to browse and watch.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).