Interview with Dick Sites

RIK FARROW

Richard L. Sites is a semi-retired computer architect and software engineer. He received his PhD from Stanford University several decades ago. He was co-architect of the DEC Alpha computers and then worked on performance analysis of software at Adobe and Google. His main interest now is to build better tools for careful, nondistorting observation of complex live real-time software, from datacenters to embedded processors in vehicles and elsewhere. dick.sites@gmail.com

Rik is the editor of *;login:*. rik@usenix.org

Part of editing *;login:* means reading the near-final page proofs. The authors have had their chance to correct mistakes that appeared after the production pipeline, and so I have a chance to read each article one last time prior to publication. While reading Dick Sites's article about his kernel tracing tool [1] and his bio, I decided I had some more questions about his article.

I also got to ask Dick about things he's done in his long career. In a three-hour interview at the Computer History Museum [3], Dick says that the summary of places he's worked spans seven pages. He started college (MIT) early and immediately started working as a programmer for IBM. I wasn't so much interested in Dick's early years, although they are fascinating, as I was in other more recent topics, things we covered by phone.

Rik Farrow: As I read your article [1] again, I wondered how you came up with these examples. Were they the results of prior work, or perhaps a lot of experimentation?

Dick Sites: I have been working on and teaching about KUtrace for several years now, and looking at the output from literally hundreds of traces.

As noted in the references to my article [1], Lars Nyland (Nvidia) did the initial scheduler comparison in the class I was teaching at the University of North Carolina in the fall of 2019. I redid it with a simpler program for this article.

Too-early `mwait` shows up in almost all Linux traces on Intel x86, which uses Intel-specific idle loop code, versus the less-aggressive generic code used for AMD chips. The idle loop is a kernel-mode process that does nothing but tries to do it slowly and with little power consumption.

I had seen unusually slow IPC (instructions per cycle) now and then over the last couple of years. I added IPC tracking to KUtrace in late 2017, but I only added the frequency tracing in 2020, which immediately revealed portions of code executing 5× too slowly. That explained the 5× drops in instructions per (constant) cycle, which really means instructions per 1/3.9 nsec on a 3.9 GHz chip.

The original 1972 Cray-1 cycle counter incremented once per CPU cycle and could be read in one cycle. I carried this idea into the first DEC Alpha chip in 1992, and it appeared across the industry by 1994. The 2001 introduction of Intel SpeedStep meant that the CPU clock frequency varied, creating problems for code that used the cycle counter to track elapsed time. Thus the so-called "constant TSC" was introduced in 2005 with a very simple implementation. A CPU clock is created by multiplying up some base clock frequency of say 100 MHz. Multiplying by 39 gives a 3.9 GHz clock; multiplying by eight gives an 800 MHz clock. SpeedStep and follow-ons just vary the multiplier. To produce a constant TSC on a chip advertised as 3.9 GHz, the chip always increments the cycle counter by 39 at a 100 MHz rate, independent of the actual CPU clock multiplier. The same chip advertised as 3.6 GHz would always increment by 36.

Page faults occur all over the place, usually in bursts, as shown in the Cost of Malloc section [1]. Even a trace on a vehicle board showed page fault bursts that were a complete surprise since no paging is done.

I am working on a paper to submit that focuses on explaining the 30× range of response times from 200 absolutely identical in-memory key-value lookup RPCs on a client-server pair of x86 desktops. Some of the underlying reasons for variation are the same as here, but the target audience is different—application programmers in response-time-constrained client-server environments.

RF: These days, eBPF, or just BPF, seems to be the favorite tool for profiling kernel events. I suspect that you wouldn't still be working on KUtrace unless each tool fulfilled different roles. BPF queries kernel structures, from what I understand, while KUtrace seems more focused on capturing timings of kernel events or system calls.

DS: It is all about speed. eBPF takes a bytecode program and interprets it to decide what to do and what to trace. Newer versions have a just-in-time compiler, but that is off by default in Linux. The JIT has been a source of security exposures.

eBPF is useful for tracking less common events or less common packets. The fact that the "F" means "filter" is the clue—it is not designed to track all packets or, in its extended form (the "e"), to track all of anything else. eBPF is not designed to track *all* system calls, interrupts, faults and context switches at full speed in a real-time environment. KUtrace is designed to do that and essentially nothing else, taking about 40 CPU cycles per transition.

The other clue is in your use of the word "profile"—a set of counts of how often something happened, with no timeline relating them. Profiles are useless for understanding variance between execution times of nominally similar tasks, because profiles simply average together all instances. That is what drove me to design KUtrace.

RF: You seem to be focused on Intel architectures? Have you looked at other CPU architectures?

DS: During March 2020 I ported KUtrace to the Raspberry Pi-4B and now have some interesting traces from the low end of the computing spectrum. I will be revising my book proposal, introduction, and some content to change the emphasis from just datacenter software to the entire span of datacenter to embedded computing.

RF: The article [1] you wrote for the Summer 2020 issue and your *ACM Queue* article [2] both feature some amazing graphs. Does KUtrace include tools to help produce such useful visualizations from the output of KUtrace?

DS: Yes, all the diagrams are produced by the KUtrace post-processing programs, posted on GitHub. The `rawtoevent` program turns raw binary trace files into text, `eventtospan` turns transitions into timespans expressed as a long JSON file, and `makeself` packages that and a JavaScript template (4200 non-comment lines) into an HTML/SVG file. The article diagrams are high-resolution screenshots or SVG. I have spent more development time on the diagrams than on the raw tracing.

References

[1] R. L. Sites, "Anomalies in Linux Processor Use," *;login:*, vol. 45, no. 2 (Summer 2020): https://www.usenix.org/system/files/login/articles/login_summer20_05_sites.pdf.

[2] R. L. Sites, "Benchmarking 'Hello World'," *ACM Queue*, vol. 16, no. 5 (November 2018): https://queue.acm.org/detail.cfm?id=3291278.

[3] "Oral History of Dick Sites": https://www.youtube.com/watch?v=A47a6Nqa2aM.