OPINION

Musings

RIK FARROW

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

In my June 2011 Musings [1], I used the metaphor of an assembly line's parts supply for the hierarchy of storage used with modern processors: cache, memory, disk, and network. I've always been amazed by both assembly lines, and how it is possible that a CPU can get so much work done when it is so much faster than the devices that supply it.

Modern assembly lines are less than a century old, and through the mechanism of YouTube we can easily watch examples of assembly lines at work [2, 3]. In the first example, you can watch a Chevrolet assembly line from 1936, and in the newer one, a BMW line from 2010. One big difference between the two lines is that in the BMW example, the only time you see a person appears to be accidental, just someone walking past the line. In the older line, most people are simply positioning parts, or performing a small set of tasks like several welds or tightening bolts. It is pretty easy to see why owners of a modern line might want to replace people doing boring, repetitive work—even if well-paying—with mindless machines.

Not an Assembly Line

My assembly line metaphor really falls short in a particular way that would have annoyed Alan Turing. A Turing machine mandates having the ability to test results and then branch, something assembly lines do not do. Much work has been done by Intel and other CPU vendors on branch prediction, because CPUs do include miniature assembly lines, called pipelines, which work best when kept filled with both instructions and data. A missed branch invalidates all the work already done by the pipeline, another cause of delay in our speedy processors. These missed branches also mean changes stored in the fastest (L1) cache, meaning more delays waiting for the slower caches and much slower memory.

The inflexibility of real assembly lines is actually a problem for manufacturers. Setting up an assembly line takes time and money, so manufacturers want to continue to use each line for as long as possible. Another problem, similar to the CPU's test and branch, occurs when one stage of the assembly line breaks down: the entire line stops. I got to visit a truck frame assembly line once, while I was visiting a factory that was investigating ways of manufacturing frames without using the traditional assembly line. That company wanted a system that was both more flexible and capable of keeping production going even if one stage breaks down or runs out of supplies.

Locality of instructions is just as important as careful arrangement of data. You are probably aware of several techniques that help with the locality of instructions. Loop unrolling involves doing more work before the (potentially) terminating test instruction. Function inlining replaces a function call with the set of instructions that act on the calling arguments. Both make the code larger, but both reduce the amount of jumping to other locations in memory.

Another technique, which evolved during the late '80s, I believe, was the use of re-entrant libraries. Instead of statically linking libc into every binary, libc gets shared among all programs that use it, through being dynamically linked. This allows just one copy of libc to be present in memory, more likely just the parts of it currently in use, instead of those same parts being loaded into many locations in memory.

Kernel samepage merging [4] is a more recent development that also helps to avoid having multiple copies of the same instructions in memory. Originally developed to reduce the memory footprint of running multiple VMs, where you'd expect there to be lots of duplicate pages if you are running the same OS in many VMs, KSM is now recommended for use even on systems where you are not running VMs.

FlexSC [5] represents another advance in dealing with memory issues caused by system calls. In almost all systems since the dawn of multiprocessing, a special instruction triggers an exception that is handled by the kernel's system call interface. While the kernel executes, it uses its own instructions and data, necessitating the replacement of cache lines used by the currently waiting program. Besides the replacement of user mode instructions and data in various cache levels, other data also gets flushed, such as entries in the data translation look-aside buffer (dTLB), used to convert virtual to physical addresses. In the FlexSC paper, these side effects of a system call exception can decrease the number of instructions completed per CPU clock cycle (IPC) by 20% to 60%.

To fit this into the assembly line metaphor, an exception-based system call is like taking a portion of the parts supply for an assembly line and using it to support a completely different assembly line. This analogy is a forced one, as system calls are actually working on behalf of the program being executed. But it is as if a second assembly line gets called into play, one that shares the same supply stream, and that interference results in less work being completed overall.

The Lineup

We start this issue with an article about a tool for determining the correct balance of memory, disk, and SSD for a server application. Madhyastha et al. explain how their tool, scc, takes into account SLAs and the need for storing and access data, and is able to both reduce cost while suggesting appropriate changes in the proportions of storage devices used. Their implementation of scc is available for download (http://www.cs.ucr.edu/~harsha/scc/).

David Slik describes Cloud Data Management Interface (CDMI), an open protocol for storage data transfer and management for cloud and object storage systems. David first explains the goals of the standard and then demonstrates how its RESTful interface can be used (with curl). CDMI is designed so that storage providers can supply just those portions of the interface that are applicable to the services they provide, and the standard can be extended whenever support for a new interface becomes sufficiently common.

Yanpei Chen and his co-authors revisit work they did for a workshop paper on TCP incast. TCP incast occurs when many servers attempt to reply with data simultaneously, resulting in much lower data transfer. In their article, they explain incast, supply equations for modeling incast, demonstrate the fit of their equations to experimental data, and show how a simple solution can reduce the effects of incast, with several examples of popular distributed systems, including Hadoop.

Robert Escriva and his co-authors, having taken a hard look at current NoSQL solutions, decided that another approach is warranted. They have created HyperDex, a system that provides consistency and reliability guarantees while outperforming popular systems such as Cassandra on benchmarks. In their article, they explain how they use a multi-dimensional space for indexing and node assignment, and how HyperDex manages to be both fast and consistent. Along the way, they highlight issues involving NoSQL solutions.

In my interview with Nathan Milford of Outbrain, I get him to discuss his use of Cassandra. Nathan is both an architect and a sysadmin for his company, and I can tell that he feels comfortable and secure in his decision to use Cassandra, along with several other tools for distributed computing.

Doug Hughes wanted to write about a series of incidents that befell his organization, including the near loss of almost a petabyte of data. Doug describes the diagnostics for several hardware and networking-related problems in terms that will be familiar to most system administrators, and ends each story with some lessons learned. Along the way he describes some useful hardware features.

David Blank-Edelman has decided to discuss the weather in his Perl column. Well, perhaps it would be more accurate to say that he explains how to fetch weather information for particular locations using three different APIs using two different Perl modules for parsing the information. This is not just for weather junkies, but for anyone with the need to pull information out of XML or JSON-encoded data.

David Beazley takes us beyond the basics of Python's lists, sets, and dictionaries, using libraries that will be included with any Python install after versions 2.7 and 3.3. David presents some useful techniques with collections. The Counter and defaultdict objects are dictionaries but with special features, and David provides examples of how to use them, including in analyzing Web logs.

Dave Josephsen begins by being mystified by a coworker who feels that "brothifying" his food will make it more absorbable, but then goes on to tie this concept into making it easier to scale Nagios to more hosts. The Check_MK tool makes collecting multiple checks from a host appear as a single check to Nagios, while simplifying the configuration on the host.

Robert Ferrell was intrigued with the multiple dimensions used in HyperDex and decides to invent his own hyper-dimensional quantum computer. Then he worries about keeping track of data in the cloud, and visualizes techniques for monitoring data as it replicates.

While Elizabeth Zwicky takes a well-deserved break, six other book reviewers tackle six different books. Mark Lamourine discusses *Jenkins: The Definitive Guide,* covering a large book about an ever larger topic, an automated build system. Brandon Ching covers *Webbots, Spiders, and Screen Scrapers,* a second edition about collecting, storing, and processing data collected from the Web, whether from a single page or a wide sweep. Jeff Berg really liked *The Tangled Web,* a book

for anyone who needs to secure Web applications. Evan Teran takes a look at *A Bug Hunter's Diary,* a book targeted at those interested in learning how to find and exploit code vulnerabilities present in various popular software programs. Peter Salus considers *A Culture of Innovation,* a collection of narratives by 19 individuals who have worked at BBN over the years. And, finally, I review *D is for Digital,* certainly the easiest read of this lot, but also a useful book to give to any educated person who wants to know more about computers and networking.

This issue concludes with summaries from the USENIX FAST conference. I've always enjoyed FAST, possibly because it combines hardware and software, academic and commercial research, into a single conference. The scc tool (Madhyastha et al.) was presented as a FAST paper, and there are many other excellent papers summarized in this issue.

Even though the authors of FlexSC have demonstrated that the effects of system calls go well beyond the side-effects of a software interrupt—saving register and other process state, performing the system call (potentially blocking), then restoring process state and continuing to execute in user mode—not much has changed since then. CiteSeerX shows only three citations, and FlexSC has certainly not become a part of the Linux kernel. Yet Apache httpd runs twice as fast with FlexSC, and there are few proposed system-level changes that have such strong, positive effects. I am left wondering whether there are other, better methods for avoiding cache pollution caused by system calls, or are there perhaps architecture advances on the hardware side that will lead to a more efficient system call interface?

Resources

[1] Musings, *;login:*, June 2011, vol. 36, no. 3.

[2] Chevrolet Assembly line in 1936: http://youtu.be/HPpTK2ezxL0.

[3] BMW in 2010: http://youtu.be/KEQdn57Kz1Q.

[4] Kernel samepage merging: http://www.linux-kvm.org/page/KSM.

[5] Livio Soares and Michael Stumm, "FlexSC: Flexible System Call Scheduling with Exception-LessSystem Calls": https://www.usenix.org/conference/osdi10/flexsc-flexible-system-call-scheduling-exception-less-system-calls.