Musings

RIK FARROW

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

After writing this column for more than 15 years, I'm a bit stuck with what I should talk about. But I do have a couple of things on my mind: Krste Asanović's FAST '14 keynote [1] and something Brendan Gregg wrote in his *Systems Performance* book [2].

Several years ago, I compared computer systems architecture to an assembly line in a factory [3], where not having a part ready (some data) held up the entire assembly line. Brendan expressed this differently, in Table 2.2 of his book, where he compares the speed of a 3.3 GHz processor to other system components by using human timescales. If a single CPU cycle is represented by one second, instead of .3 nanoseconds, then fetching data from Level 1 cache takes three seconds, from Level 3 cache takes 43 seconds, and having to go to DRAM takes six minutes. Having to wait for a disk read takes months, and a fetch from a remote datacenter can take years. It's amazing that anything gets done—but then you remember the scaling of 3.3 billion to one. Events occurring in less than a few tens of milliseconds apart appear simultaneous to us humans.

Krste Asanović spoke about the ASPIRE Lab, where they are examining the shift from the performance increases we've seen in silicon to a post-scaling world, so we need to consider the entire hardware and software stack. You can read the summary of his talk in this issue of *;login:* or go online and watch the video of his presentation.

A lot of what Asanović talked about seemed familiar to me, because I had heard some of these ideas from the UCB Par Lab, in a paper in 2009 [4]. Some things were new, such as using photonic switching and message passing (read David Blank-Edelman's column) instead of the typical CPU bus interconnects. Asanović also suggested that data be encrypted until it reached the core where it would be processed, an idea I had after hearing that Par Lab paper, and one that I'm glad someone else will actually do something about. Still other things remained the same, such as having many homogeneous cores for the bulk of data processing, with a handful of specialized cores for things like vector processing. Asanović, in the question and answer that followed his speech, said that only .01% of computing requires specialized software and hardware.

Having properly anticipated the need for encryption of data while at rest and even while being transferred by the photonic message passing system, I thought I'd take a shot at imagining the rest of Warehouse Scale Computing (WSC), but without borrowing from the ASPIRE FireBox design too much.

The Need for Speed

First off, you need to keep those swift cores happy, which means that data must always be ready nearby. That's a tall order, and one that hardware designers have been aware of for many years. Photonic switching at one terabit per second certainly sounds nice, and it's hard to imagine something that beats a design that already seems like science fiction based on the name. For my design, I will simply specify a message-passing network that connects all cores and their local caches to the wider world beyond. Like the FireBox design, there will be no Level 1 cache coherency or shared L1 caches. If a module running on one core wants to share

data with another, it will have to be done via message passing, not by tweaking shared bits and expecting the other core to notice.

Whatever interconnect you choose, having sufficient bisection bandwidth will be key to the performance of the entire system. Can't have those cores waiting many CPU cycles for the data they need to read or write. The photonic switches have lots of bandwidth, and if they can actually switch without resorting to converting light back to electrons then to light again, they can work with a minimal of latency.

In my design, I imagine having special encryption hardware that's part of every core. That way, checking the HMAC and the decryption of messages (as well as the working in the opposite direction) can take advantage of hardware built for this very specific task. It should be possible to design very secure systems using this approach, because all communications between processes and the outside world come with verification of their source—a process that knows the correct encryption key. Like the FireBox, this system will be a service-oriented architecture, with each core providing a minimal service, again minimizing but not eliminating the probability that there will be security-affecting bugs in the code.

Some cores will be connected to the outside world, managing communications and storage. This is not that different from current approaches, where network cards for VM hosts already do a lot of coprocessing and maintain multiple queues. But something similar will need to be done for storage, as it will remain glacially slow, from the CPU's perspective, even with advances such as non-volatile memory (NVM) being available in copious amounts with speeds as fast or faster than current DRAM.

Cores will be RISC, because they are more efficient than CISC designs. (Note to self: sell Intel, buy ARM Holdings.) With Intel server CPUs like eight-core Xeon ES having 2.7 billion transistors, that's a lot of heat, much of which is used to translate CISC instructions into internal, RISC-like, microcode instructions. The AMD A1100 that was announced in January 2014 will have eight 64-bit ARM (Cortex-A57) cores and built-in SIMD, which supports AES encryption and is rated at 25 watts TDP (thermal design power), compared to 80 for the 3.2 GHz Xeon. (Hmm, buy AMD?)

Unlike the FireBox's fit-in-a-standard-rack design, my imaginary system will look like something designed by Seymour Cray, but without a couch. Cray's best-known designs were circular, because Cray was concerned about having components separated by too much distance. After all, light can only travel 29.979 cm in a nanosecond, and with CPU clock cycles measured in nanoseconds in Cray's day, distance mattered. Actually, distance matters even more today.

My design has the outward appearance of a cube. Inside, the cores will be arranged in a sphere, with I/O and support filling in the corners. Also, unlike one of Cray's designs, where you could see the refrigerant flowing around the parts, my cube will float on a fountain of water. The water will both cool and suspend the cube, while the I/O and power connections will prevent it from floating off the column of water.

At the end of Asanović's talk, I asked him why they would be using Linux. Asanović replied that Linux provides programmers with a familiar interface. That's certainly true, and I agree. But I also think that a minimal Linux shell, like that provided by a picoprocess [5], will satisfy most programs compiled to work with Linux, while being easy to support with a very simple message passing system under the hood (so to speak).

Except for some dramatic flairs, I must confess that my design is not that different from the FireBox.

Reality

One problem with my floating cube design is how to deal with broken hardware. Sometimes cores or supporting subsystems fail, and having to toss an entire cube because you can't replace failed parts isn't going to work. There's a very good reason why supercomputers today appear as long rows of rackmount servers [6]. One can hope that the reason the FireBox will fit in racks is that it contains modules that can be easily serviced and replaced.

Using water for cooling has been done before [7], but it does make maintenance more difficult than just using air. Still, even low-power RISC cores dissipate "waste" heat, and having 1000 of them translates into 3.3 kilowatts of heat—a space heater that you really don't want in your machine room. Still, that beats the 12.5 kilowatts produced by the Xeons.

Even the photonic switching network could prove problematic. In the noughts, a company named SiCortex [8] built supercomputers that used RISC cores and featured a high-speed, message-passing interconnect using a diameter-6 Kautz graph, and they failed after seven years and only selling 75 supercomputers. Perhaps the market just didn't think that having an interconnect designed to speed intercore and I/O communications was important enough.

The Lineup

We start this issue with an article from the people who built ~okeanos, a public cloud for Greek researchers. They have written for *;login:* before [9], and when I discovered that they had used Ceph as the back-end store, I asked if they would write about their experiences with Ceph. The authors describe what they needed from a back-end storage system, what they tried, and how Ceph has worked so far.

EDITORIAL

Musings

I met Tyler Harter during a poster session at FAST '14. Tyler had presented what I thought was a great paper based on traces collected from Facebook Messages. What he and his co-authors had discovered showed just how strongly layering affects write performance, resulting in a huge amount of write amplification. In this article, Harter et al. explain how they uncovered the write amplification and what can be done about it, as well as exploring whether the use of SSDs would improve the performance of this HBase over HDFS application.

I knew that Mark Lamourine had been working on Red Hat's implementation of OpenStack and thought that I might be able to convince him to explain OpenStack. OpenStack started as a NASA and Rackspace project for creating clouds and has become a relatively mature open source project. OpenStack has lots of moving parts, which makes it appear complicated, but I do know that people are using it already in production. The ~okeanos project is OpenStack compatible, and if you read both articles, you can learn more about the types of storage required for a cloud.

Tim Hockin shares an epic about debugging. What initially appeared to be a simple problem took Hockin down many false paths before he finally, after going all the way down the stack to the kernel, found the culprit—a tiny but critical change in source code.

David Lang continues to share his expertise in enterprise logging. In this issue, Lang explains how to detect and fix performance problems when using rsyslog, a system he has used and helps to maintain.

Andy Seely introduces us to some rock stars. You know, those people you worked with at the startup that didn't make it? The ones willing to work long hours for a reward that remained elusive? Seely's story is actually about how a small management change improved the lives of the people he worked with.

David Blank-Edelman delves into the world of message queues via 0MQ. I became interested in message queues when I learned, from Mark Lamourine, that they were being used in OpenStack. David shows us how simple it is to use 0MQ, as well as demonstrating just how powerful 0MQ is, using some Perl examples, in the first of a two-part series.

Dave Beazley explores a feature found in the newest version of Python, asynchronous I/O. You might think that async-io has been around for a while in Python, and you'd be right. But this is a new implementation, which considerably simplifies how event loops are used. Oh, and there's a backport of the new module to Python 2.7.

Dave Josephsen has us considering monitoring design patterns. I found his column very timely, as I know that sysadmins are questioning the design patterns they have been using for many years to collect status and information.

Dan Geer and Jay Jacobs discuss where we are today in collecting security metrics. At first, we just needed to start collecting usable data. Today, what's needed is the ability to better perform meta-analysis of publicly available data.

Robert Ferrell also explores clouds and muses about the future of advertising. Like Robert, I just can't wait until my heads-up display is showing me advertising when what I really want are the directions to where I needed to be five minutes ago.

James Mickens had written a number of columns that were only published online, and we decided to print his first one [10]. James has been experiencing deep, existential angst about issues surrounding the unreliability of untrusted computer systems. In particular, papers about Byzantine Fault Tolerance. Don't worry, as James' column will not put you to sleep.

Elizabeth Zwicky has written three book reviews. She begins with a thorough and readable tutorial on R, covers an excellent book on threat modeling, and finishes with a book on storage for photographers.

This will be Elizabeth's final column. Elizabeth has been the book reviews columnist for *;login:* since October 2005, and I, like many, have thoroughly enjoyed her erudite and droll reviews. Her work will be missed, although I hope she still finds the time to pen the occasional review.

If you want to contribute reviews to *;login:* of relevant books that you have read, please email me.

We also have summaries of FAST '14 and the Linux FAST Summit. I took notes there and converted them into a dialog that covers a lot of what happened during the summit. The summary also provides insights into how the Linux kernel changes over time.

Although the key ideas of the FireBox design appear sound to me, people have learned how to work with off-the-shelf rackmounted servers for massive data processing tasks, and the biggest change so far has been to move toward using SDN for networking. Perhaps there will be a move toward customized cores as well. When designing WSC, the ability to keep data close to where it is processed has been the key so far, whether we are discussing MapReduce or memcached.

Resources

[1] "FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers": https://www.usenix.org/conference/fast14/technical-sessions/presentation/keynote.

[2] Brendan Gregg, *Systems Performance for Enterprise and Cloud* (Prentice Hall, 2013), ISBN 978-0-13-339009-4.

[3] Rik Farrow, "Musings," *;login:,* vol. 36, no. 3, June 2011: https://www.usenix.org/login/june-2011/musings.

[4] Rose Liu, Kevin Klues, Sarah Bird, Steven Hofmeyr, Krste Asanović, and John Kubiatowicz, "Space-Time Partitioning in a Manycore Client OS," Hot Topics in Parallelism, 2009: https://www.usenix.org/legacy/event/hotpar09/tech/full_papers/liu/liu_html/.

[5] Jon Howell, Bryan Parno, and John R. Douceur, "How to Run POSIX Apps in a Minimal Picoprocess": http://research.microsoft.com/apps/pubs/default.aspx?id=190822.

[6] Bluefire at UCAR, in the *Proceedings of USENIX Annual Technical Conference (ATC '13)*, June 2013: http://www.ucar.edu/news/releases/2008/images/bluefire_backhalf_large.jpg.

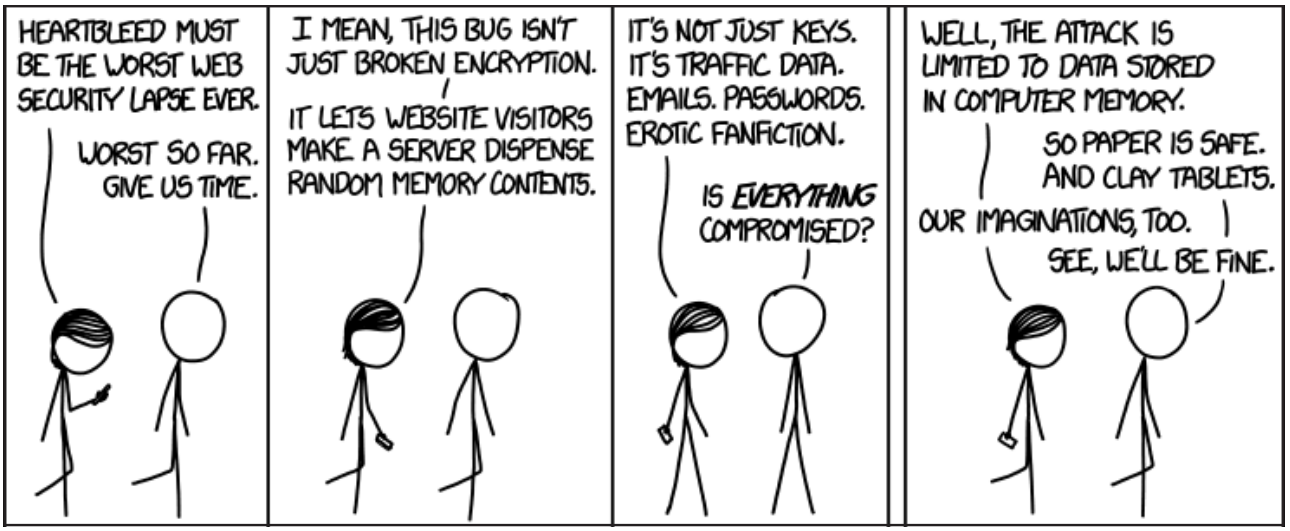
[7] Hot water-cooled supercomputer: http://www.zurich.ibm.com/news/12/superMUC.html.

[8] SiCortex supercomputer: http://en.wikipedia.org/wiki/SiCortex.

[9] Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris, "Synnefo: A Complete Cloud Stack over Ganeti," *;login:,* vol. 38, no. 5, October 2013: https://www.usenix.org/login/october-2013/koukis.

[10] James Mickens, "The Saddest Moment," ;login: *logout,* May, 2013: https://www.usenix.org/publications/login-logout/may-2013/saddest-moment

xkcd



xkcd.com