EDITORIAL

Musings

RIK FARROW

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

The trouble with the future is that it never arrives. For me, there is only right now, with an imagined future some constant distance away in the, uh, future. But things do change.

I love attending FAST, because some of my own roots are on the vendor side, and FAST provides one of the best blends of academic and vendor research of any USENIX conference. FAST '13 was no exception, and when Rick Wheeler suggested I attend a particular BoF (one I might have skipped), I discovered something interesting.

A Game Changer?

Andy Rudoff, now with Intel, once a Sun engineer, was already well into his presentation when I wandered in. Andy was enthusiastic, that was certain. But would Non-Volatile Memory (NVM) really be the game changer that he was hinting at?

Not that Andy was really hinting at anything. His presentation, and his article in this issue, was about adding two new interfaces for NVM. We already have two interfaces for NVM, in the now familiar form of SSDs: a block and a file interface. What Andy was explaining relies on there being a form of NVM that is addressable at the byte level, instead of the block level, as with current Flash devices.

This suggestion got my attention. During HotOS 2011, Katelin Bailey presented a position paper about systems built with NVM [1]. Imagine a system with gigabytes of byte-addressable *persistent* memory. If you hibernate such a system, it can sleep using no power, but wake up immediately with all its memory intact. You don't need virtual memory, because you don't need to use disk for backing limited DRAM.

But there are also problems with this picture. For example, now we can safely assume that rebooting a system clears memory (well, almost [2]), but if DRAM is replaced with NVM, that assumption is no longer true; whatever got us into trouble before is still there.

Andy suggests two new interfaces to byte-addressable NVM: PM (persistent memory) Volume Mode and PM File Mode. Although these might sound similar to the current ways we have for accessing Flash, they are different in that they assume the CPU can perform load/store instructions at the byte level, which is very different from the block-oriented Flash we've been learning about and using for years.

In an interview [3], Intel's Chief Technology Officer, Justin Rattner, said that new NVM will have low latency, in the tens of nanoseconds. Just compare that to typical disk latency, which is measured in milliseconds. Well, let's see:

$$\text{nanosecond} = 10^{-9} \text{ vs. millisecond} = 10^{-3}$$

That would be quite a difference. I tried to pry more from Andy about just how much NVM we might expect to have, but he couldn't tell me—or if he could, I couldn't tell you. You know, the usual NDA conundrum. But my suspicion is that NVM could be a real game changer, if the various technologies for creating this memory actually prove capable of being used in cheap, reliable, fast, and dense devices.

Shingles, Not on a Roof

Not that I am expecting terabytes (or tebibytes [4]) of affordable PM anytime soon. Hard disk drives (HDDs) have managed to grow in capacity at a rate of about 40% a year. But this growth is going to hit a wall soon, as the ability to write sectors to ever narrower tracks has become a serious problem.

HDD heads can read narrower tracks, but the magnetic field used to record sectors disturbs nearby sectors, unless enough space is left between the tracks. And it is this space that the drive vendors are planning to get rid of. If you look at Figure 2 in Tim Feldman's article on page 22, you can at least get a feeling for how much more space: perhaps a doubling of space per drive. But this doubling (my guesstimate) comes with a price: sectors will be overlapping. The overlapping sectors can be read easily enough, but randomly writing a sector also means overwriting overlapping sectors, and the loss of the data stored there.

Of course, losing data is not acceptable, so the plan for shingled disks, where "shingling" refers to an overlap like we see in shingled roofs, is to have firmware on these HDDs manage reading and writing. If a random write must overwrite some sectors, these need to be read first, so they can be rewritten. As sectors get moved around, HDDs will need maps, like those used in the Flash Translation Layers (FTLs) in SSDs. And this will mean that shingled disks will behave more like SSDs in that their performance will become more erratic, depending on the internal state: Are there sectors that can be written *now* without having to relocate data?

I asked Ted Ts'o, in an interview in this issue, what he, wearing his file system developer and Google storage guy hat, thinks about Shingled Magnetic Recording (SMR) disks. Ted wasn't very positive, likely because current SMR drives are slower than non-SMR drives. He can foresee using these drives for archiving, as they work well for sequential writes and later reads, but poorly for random workloads.

The disk vendors want to get around this problem by exposing more of the internals of SMR HDDs. Instead of having drive firmware do all the work of relocating sectors and garbage collection, their idea is to allow file system designers more control over sector layout, even to the size of the shingled regions of tracks, which are called bands.

Like the NVM interface, the SMR HDD interface requires some changes to make this work, and both NVM and HDD vendors are looking for input, as they work toward creating new standards. Although it is more difficult for me to see SMR HDDs as being as much as a game changer as replacing DRAM with persistent memory, I wonder whether Ted just might be wrong about SMR. Google File System (GFS) uses 64 megabyte files on top of ext4 with journaling disabled because it is fast, and they get their reliability through redundant copies of data. But these files are (I believe) write-once; if SMR drives could provide better performance, if 64 MB bands were used instead of using ext4, I think that Google just might be interested.

Many years ago, a friend who worked on IBM mainframes in a bank let me look at some of their documentation. I discovered that this mainframe allowed programmers to format sectors to whatever size worked best with the length of the database records their application used. Although being able to chose a band size is not the same level of control once allowed IBM systems programmers, there is likely a real place for this.

The Lineup

We start out this issue with an article by Geoff Kuenning. After FAST '13, Geoff and I had lunch, and I shared some thoughts I'd had about the keynote presentation by Kai Li, the founder of Data Domain. Among other things, Li spoke about the importance of being able to turn out production quality code, and I asked Geoff what he thought about a very common occurrence during CS research paper presentations. Often graduate students will have completed an interesting project, but be unwilling to share the code, for various reasons, which I found myself questioning. Geoff agreed with me, as he believes that students need to be taught to produce readable and maintainable code, whatever the reason for writing it.

Geoff provides suggestions for both teaching better coding practices, as well as writing better code. I did press Geoff for an example of good code writing, but he was unable to find some open source code he wanted to hold up as an exemplar. I found that sad, as well as telling.

FAST began with a trio of papers about Linux file systems, including the Best Paper winner. Lu et al. spent more than a year analyzing all of the file system patches for six file systems for the entire period of the 2.6 kernel. I found their insights fascinating, although perhaps some should simply be expected. For example, the most common reason for a patch was data corruption or system crash, leaving me thinking, "Of course, how could you miss problems like that!" But there are much more subtle issues—for example, the large number of bugs found during error-handling routines. By their very nature, these code paths are not executed as often as non-error code paths, and forgetting to release resources properly while handling errors turns out to be a big problem, and one that anyone writing code might encounter.

I had met Ted Ts'o at LISA '12 (he couldn't attend FAST '13), and we started an email discussion that turned into an interview. I had questions dating back to 1991 that I thought Ted could answer, as well as questions relevant to the current state of file systems in the Linux kernel.

I've already mentioned that Tim Feldman, with Garth Gibson, has written a lengthy article about SMR. We spent a lot of

time on this article, as the problem is difficult and as yet really unsolved. There is one current solution, ShingledFS [5], but SMR is really a new opportunity for file system designers.

Next up, Guido Trotter and Tom Limoncelli describe Ganeti. Ganeti is a set of Python scripts and programs for managing VMs. Ganeti currently works with both KVM and Xen, and based on its popularity at LISA '12, I really pushed these nice guys to write an article about it. Although there are other methods for firing up, or migrating, VMs, both open source and proprietary, I sensed that Ganeti was striking a chord with the people who have tried it and that it was worth learning more about.

Garth Gibson had been talking about providing real clusters for use by CS researchers for (it seemed) years. Now, Garth and others have access to several clusters of computers that had been part of supercomputers used by the US Department of Energy. The point of this program is to provide actual hardware for testing distributed programming instead of simulations within clouds, as clouds cannot provide consistency of performance that can be matched from one run to the next. PRObE, however, allows researchers to work on large, real clusters of up to 1,000 systems, complete with the appropriate supercomputer-style network interconnect and storage.

I've already written about Andy Rudoff, and am happy he wrote a clear article about potential interfaces for byte-addressable NVM. While I can imagine other game changers to the current von Neumann system architecture, NVM is much closer to reality than anything I have imagined. I also welcome you to imagine just what you might do if you could save to memory, instead of a file.

David Blank-Edelman decided to be a bit more austere in his approach to Perl. Well, if not austere, then Constant. David discusses several approaches to having actual constants in Perl. C programmers should be well aware of the benefit of being able to define constants, and although constants are not part of Perl's core system, they can certainly be accommodated.

David Beazley wrote his column on his return from PyCon 2013. Dave covers IPython and Notebook, two hot topics at the conference. IPython is a new Python shell that combines features of common *nix shells with the Python prompt. I haven't gone that far into the Python world that I want to first `cd` then execute some Python code on the directory's contents, but Dave shows how this can be done, with a little help from some additional modules. Notebook goes much further, being more like a researcher's notebook, à la Mathematica, with statistics, charts, graphs, and both documentation and access to the shell.

Dave Josephsen discusses sampling from the perspective of monitoring. Although it's usually Elizabeth Zwicky showing both knowledge and interest in statistics, Dave explains how an understanding of sampling is important in monitoring, especially when you have more data to watch than you can or should be collecting.

Dan Geer has moved his long running column, "For Good Measure," to *;login:,* starting with this issue. He and his co-author, Dan Conway, take a measured look at how to calculate risk. They propose using an options trading model as providing a way to quantify risk.

Robert Ferrell begins with a riff about clustering software, visits high availability, then heads off wondering how best to explain UNIX file permissions to people for whom a console window is an alien notion.

Elizabeth Zwicky took this issue off, but we have book reviews from Mark Lamourine and Trey Darley. Mark begins with a book about Steampunk and the Maker culture, which really helped me put both into perspective. Then he takes a look at *Testable Javascript,* which appears to be valuable to any code writer, although you do need familiarity with JS to get the most out of it. Finally, Mark looks at a book on EPUB 3, one of the several formats used today for electronic publishing.

Trey Darley begins with a book about the culture of cryptography and the law. Not surprisingly, it turns out that the law treats concepts like non-repudiation differently than cryptographers think about it, or design for it. The focus of this book is on digital signing, and Trey has good things to say about the book. Trey briefly covers an equally lengthy book about testing physical security in the form of lockpicking.

This issue includes the FAST '13 reports, with much more on the keynote by Kai Li and great summaries of paper and poster presentations. Ao Ma presented the first paper at FAST, which motivated Kirk McKusick to add some of the features Ao described to FreeBSD's FFS that same day. Kirk wrote about this experience in the April 2013 issue of *;login:*.

Recently, I watched a feature on the *PBS News Hour* about infants and children using touch-screen devices. Hanna Rosin had written an article about children, including her own, using touch-screen-based computer devices for *The Atlantic* magazine [6]. While the interviewer appeared worried about children spending too much time playing with computers, Hanna brushed this off by saying parents need to control access when necessary.

I recalled an infant boy, perhaps two years old, playing with a computer mouse in 1989 or so. The boy, who otherwise appeared precocious, didn't get the connection with moving the mouse and the cursor moving on the screen of the Sun workstation. But there is no disconnect when a child interacts with a touch-screen device, like a tablet or smartphone. The child gets "natural" results with gestures that touch or brush against the screen.

Perhaps we are living in the future. It just seems like now.

References

[1] Bailey et al., "Operating System Implications of Fast, Cheap, Non-Volatile Memory": http://www.usenix.org/events/hotos11/tech/final_files/Bailey.pdf.

[2] J. Alex Halderman et al., "Lest We Remember: Cold Boot Attacks on Encryption Keys": http://static.usenix.org/event/sec08/tech/full_papers/halderman/halderman_html/.

[3] Jack Clark, "Intel: Non-Volatile Memory Shift Means Chips Need an Overhaul," Sept 13, 2012: http://www.zdnet.com/intel-non-volatile-memory-shift-means-chips-need-an-overhaul-7000004221/.

[4] http://en.wikipedia.org/wiki/Tebibyte.

[5] Anand Suresh, Garth Gibson, Greg Ganger, "Shingled Magnetic Recording for Big Data Applications," Anand Suresh. Garth Gibson. Greg Ganger. CMU-PDL-12-105, May 2012: www.pdl.cmu.edu/PDL-FTP/FS/CMU-PDL-12-105.pdf.

[6] Rosin, "The Touch-Screen Generation," *The Atlantic,* March 20, 2013: http://www.theatlantic.com/magazine/archive/2013/04/the-touch-screen-generation/309250/.