RIK FARROW

musings

rik@usenix.org

WE ARE ALL ACCUSTOMED TO OUR software having bugs. We would be surprised if our software actually worked perfectly, with no glitches or gaping security holes. Surprised? More like flummoxed. Oddly, we seem to have a much stronger belief about our hardware being perfect, that processors can perform 128-bit floating-point multiplies accurately, and that disks actually store what we ask them to. Well, guess again.

During FAST '08, I was treated to what has become a yearly spectacle: researchers digging into massive disk-error databases to pick apart what goes wrong with disks while in production. At FAST '07, the big news was the failure curve for hard drives not being bathtub-shaped. In 2008, researchers looked for, and found, other problems with using disks that are certainly surprising.

As one researcher pointed out, the typical hard drive includes 300,000 lines of code in its firmware: room enough for errors, eh? And we thought we were using hardware, but like many hardware devices, even hard drives are software-controlled. As Goodson and his researchers write in the lead article in this issue, silent write errors are actually a big problem. Their findings certainly have me longing for new filesystem designs, such as ZFS and the under-development BTRFS, that include checksums with the data they store. Even enterprise-level drives have a problem with silent write errors, which is something you might not expect, as these drives write checksums and error-correcting code into each sector. But when the wrong data gets written, or data gets written to the wrong physical block, error-correction code just isn't going to help you.

Buggy Hardware

There was once a time when I relied on hardware failure. I've built a lot of my desktop systems over the years and would use them until they were obsolete (for three to five years). Over time, I would use my own personal method for organizing directories, resulting in a pretty incredible mess. I could still find things, of course, and rarely lost things. But my file hierarchies began to look more and more like the amazing Winchester Mystery House in San Jose.

Then a miracle would happen. The hard drive would cease working, and only the directories I had deemed important enough to back up could be restored. I would start out with a brand new, empty filesystem, on a hard disk that was generally twice as big as the previous one. Backup media evolved from floppy disks (really) to quarter-inch tape cartridges (a whopping 45 MBs), then to CDs (700 MB!), and finally to the plug-in USB drive.

And now I was in trouble. The USB drive was newer than my desktop drive, and I wound up with stuff I had deleted being still available on the backup drive. I now find myself in need of some serious filesystem organizing.

And I, like many in this modern age, find myself with an even worse problem: How do we safely archive the records of our increasingly digital lives? Digital photos, digital recordings, and digital videos that record life's big events all wind up on hard drives, perhaps backed up to another hard drive and the read-only storage of plastic disks (CDs and DVDs). I've already spoken of the problems with hard drives, both because of sudden, unexpected deaths, but also the now uncovered issues of silent write errors. But what of other uncontrollable issues, such as the feature creep of image standards? Will the JPEG digital images of a wedding or child's first birthday still be usable in 100 years?

One of the keynote speakers at FAST, Cathy Marshall, has researched how real people are currently handling archiving their digital media, and she outlined the shaky plans her research subjects had. But how good are our own plans for archiving? Will you remember to dig out the CDs to which you copied your digital photos, read them in, and upgrade the digital format to whatever is current before the conversion software can no longer recognize the format you had been using? And how long will CDs reliably store data? Nobody knows. They haven't been around long enough yet, but we certainly know that CDs exposed to sunlight will start failing pretty quickly. Heat will also destroy data disks, and perhaps age will as well. We have all seen photos of at least some (if not all) of our grandparents and even some of their parents as well. But will our media be readable in 50 years?

Tomes

Perhaps we should consider using Pergamum Tomes, simple embedded systems complete with a hard drive, designed for creating distributed archives (see the article by Storer and his colleagues). Although I like the idea of having lots of small systems, using power over Ethernet and only spinning up the hard drives when needed, I don't consider my own home proof against the fires we sometimes see in the Southwest. In fact, every spring brings with it incredible dryness, and the beautiful scenery takes on a terrifying alter ego that may destroy entire neighborhoods as well as forests. And there go my archives.

But there are still more hardware issues. Jiang et al. explain that more problems were found with supporting hardware than with hard drives in their research. Their article and FAST '08 paper go a long way toward explaining why disk manufacturers often find no problems with drives returned because they appeared DOA.

Sluggish Memory

I've been harping on CPU performance issues in my columns, and with solicited articles, for many years now. What has likely become evident to anyone with a technical background is that making CPUs run faster has little to

do with increasing clock speeds. The performance bottlenecks today have to do with memory latency issues. If a cache miss results in having to grab a line from DRAM, the processor can wait many thousands of cycles for the data needed to arrive. And if the very next operation results in another cache miss because of poor data layout, many more thousands of CPU cycles may be wasted.

CPU vendors have gotten creative about trying to hide these issues. Having multithreaded cores works to mitigate the problems, by allowing one thread to go idle while another thread proceeds once data has been copied from DRAM to cache. Sun's Niagara architectures does this very well, and Intel's Hyperthreading provides at least two threads to help with this problem.

Many-cored systems are the other important advance in processor technology, but this advance comes with a large dose of programming pain. McCool's article in the April 2008 issue focused on increased parallelism in CPU design, carefully describing the many different types of parallelism found in current architectures. In this issue, Andrew Brownsword looks at how parallelism is the only hope we have for increasing actual CPU performance. Andrew explains why this is so, invoking Amdahl's Law, and then provides examples of just why parallel programming is so difficult to do. We have neither the hardware support, such as transactional memory, nor the software approach, in new languages, compilers, and debuggers, for writing good parallel software.

I was very excited to learn that Dave Patterson will be giving the keynote at the 2008 USENIX Annual Technical Conference in Boston this summer. Patterson, best known for the development of both RISC and RAID, will be telling attendees that they are going to have to work with many-core systems and their increased parallelism, whether they like it or not. And I agree. Today, smart phones, such as the iPhone, consist of many processing cores, and this is going to be the future of desktops and servers. We already have desktops that include multiple GPUs used for graphics coprocessing (and for some scientific work as well). Many-core systems are the future of computing, and we need software that will support this future. In fact, we needed this software years ago. As Andrew points out in his article, testing CPU utilization at Electronic Arts, where he works, shows that CPUs generally run at 4% of capacity while executing an application, and occasionally they reach 60% utilization but only with painstakingly handcrafted code. So much for your 3.2-GHz Xenon CPUs and the registered DRAMs that run hot to the touch, as the CPUs are idling the majority of the time, waiting for data to make its way from memory.

I have often written about the problems with the current batch of hardware and software (see, for example, the August 2007 "Musings"). Behind the scenes, I have done more than just whine. I have encouraged researchers and practitioners to start thinking outside the box of the past. The most tangible effect, although certainly not one that I can take credit for, is the first USENIX Workshop on Hot Topics in Parallelism (HotPar '09: http://www.usenix.org/events/hotpar09/).

Research into parallelism has become a very hot topic, quite publicly pushed by Intel and Microsoft. I yearn for the day when our cell phones, desktops, and servers cease to be architected like micro-mainframes, with designs more suited to time-sharing, but instead begin to follow the secure and truly parallel models of computing that we should be using.

In This Issue

I've actually managed to introduce all the articles but two, at this point. Right before the FAST '08 conference, Margo Seltzer and I interviewed Dave Hitz, a founder of NetApp, and Brian Pawlowsky, an NFS developer there, about the future of SAN and NAS. The interview was recorded and transcribed, and you can find that transcription at www.usenix.org/publications/login/2008-06/netappinterview.pdf. I've taken the 42 pages of transcription and edited it down to something that fits on five pages. Of course, I don't cover all the points in that interview, but that's just not the point.

We also have an article written by Dan Tsafir, Dilma Da Silva, and David Wagner. Dan and his co-authors produced an article examining problems with setting user and group IDs correctly in widely utilized code. They set about explaining how the various methods for changing effective user and group IDs work, and they provide a portable technique for doing this properly.

In the columns, we start out with David Blank-Edelman, who demonstrates many of the ways that Perl can be used to store data structures. Peter Galvin then provides us with a splendid update on ZFS, including current and future features, as well as the strengths and weaknesses of the current implementation.

Dave Josephsen takes a different look at storage, from the perspective of auditing disk reads and writes. Dave finds most current techniques in Linux lacking, and he demonstrates a clever use of SystemTap to capture the user and group IDs during reads and writes of a particular disk partition.

Robert Ferrell provides us with his own personal take on storage issues. We have Nick Stoughton's comments on his herculean tasks working with competing standards committees, trying to manipulate the C++ standards body into not diverging too far from POSIX standards. We end, as usual, with a collection of book reviews.

Did I write "end"? Actually, we have summaries from both FAST '08 and the Linux Storage and Filesystem workshop. While the FAST summaries provide you with the scoop on what happened during the conference, the LSF summaries provide a different sort of insight, as in what to expect from the Linux community in regards to file systems in the next year. I found myself particularly interested in the issues surrounding solid state disks, as current high-end models can complete I/O operations faster than can be handled. It's obvious that support for SSDs is years out, so if you have plans to solve issues with memory-starved applications, SSDs are not a short-term solution.