;login:
JUNE 2013 VOL. 38, NO. 3

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

File Systems

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

FOR A COMPLETE LIST OF USENIX AND USENIX CO-SPONSORED EVENTS, SEE WWW.USENIX.ORG/CONFERENCES

2013 USENIX Federated Conferences Week

June 24–28, 2013, San Jose, CA, USA
www.usenix.org/conference/fcw13

USENIX ATC '13: 2013 USENIX Annual Technical Conference
June 26–28, 2013
www.usenix.org/conference/atc13

ICAC '13: 10th International Conference on Autonomic Computing
June 26–28, 2013
www.usenix.org/conference/icac13

HotPar '13: 5th USENIX Workshop on Hot Topics in Parallelism
June 24–25, 2013
www.usenix.org/conference/hotpar13

UCMS '13: 2013 USENIX Configuration Management Summit
June 24, 2013
www.usenix.org/conference/ucms13

8th International Workshop on Feedback Computing
June 25, 2013
www.usenix.org/conference/feedbackcomputing13

ESOS '13: 2013 Workshop on Embedded Self-Organizing Systems
June 25, 2013
www.usenix.org/conference/esos13

HotCloud '13: 5th USENIX Workshop on Hot Topics in Cloud Computing
June 25–26, 2013
www.usenix.org/conference/hotcloud13

WiAC '13: 2013 USENIX Women in Advanced Computing Summit
June 26–27, 2013
www.usenix.org/conference/wiac13

HotStorage '13: 5th USENIX Workshop on Hot Topics in Storage and File Systems
June 27–28, 2013
www.usenix.org/conference/hotstorage13

HotSWUp '13: 5th Workshop on Hot Topics in Software Upgrades
June 28, 2013
www.usenix.org/conference/hotswup13

USENIX Security '13: 22nd USENIX Security Symposium

August 14–16, 2013, Washington, D.C., USA
www.usenix.org/conference/usenixsecurity13

Workshops Co-located with USENIX Security '13

EVT/WOTE '13: 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections
August 12–13, 2013
www.usenix.org/conference/evtwote13

USENIX Journal of Election Technology and Systems (JETS)
Published in conjunction with EVT/WOTE
www.usenix.org/jets
Submissions for Volume 1, Issue 2, due: August 6, 2013

CSET '13: 6th Workshop on Cyber Security Experimentation and Test
August 12, 2013
www.usenix.org/conference/cset13

HealthTech '13: 2013 USENIX Workshop on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies
August 12, 2013
www.usenix.org/conference/healthtech13

LEET '13: 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats
August 12, 2013
www.usenix.org/conference/leet13

FOCI '13: 3rd USENIX Workshop on Free and Open Communications on the Internet
August 13, 2013
www.usenix.org/conference/foci13

HotSec '13: 2013 USENIX Summit on Hot Topics in Security
August 13, 2013
www.usenix.org/conference/hotsec13

WOOT '13: 7th USENIX Workshop on Offensive Technologies
August 13, 2013
www.usenix.org/conference/woot13

LISA '13: 27th Large Installation System Administration Conference

November 3–8, 2013, Washington, D.C., USA
www.usenix.org/conference/lisa13

SESA '13: 2013 USENIX Summit for Educators in System Administration

CO-LOCATED WITH LISA '13
November 5, 2013, Washington, D.C., USA
www.usenix.org/conference/sesa13
Submissions due: July 30, 2013

FAST '14: 12th USENIX Conference on File and Storage Technologies

February 17–20, 2014, Santa Clara, CA, USA
www.usenix.org/conference/fast14
Submissions due: September 26, 2013

;login:
JUNE 2013 VOL. 38, NO. 3

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Rikki Endsley
rikki@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Michele Nelson

TYPESETTER
Star Type
startype@comcast.net

USENIX ASSOCIATION
2560 Ninth Street, Suite 215,
Berkeley, California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

www.usenix.org

;login: is the official magazine of the USENIX Association. *;login:* (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for non-members are $90 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

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/.

OPINION

On Teaching Style and Maintainability

GEOFF KUENNING

Geoff Kuenning spent 15 years working as a programmer before changing directions and joining academia. Today he teaches computer science at Harvey Mudd College in Claremont, California, where he has developed a reputation for insisting that students must write readable software, not just working code. When not teaching or improving his own programs, he can often be found trying—and usually failing—to conquer nearby Mount Baldy on a bicycle. geoff@cs.hmc.edu.

Computer science has existed as a separate discipline for more than 50 years, and in that time we have learned a lot about what is important to the field and how to teach it to new entrants. We have long agreed that every self-respecting computer scientist should have a solid grounding in fundamental areas such as algorithms, discrete mathematics, programming languages, data structures, operating systems, software engineering, etc. But in this article, I will argue that there is a major missing component: style and readability. I'll try to convince you that style matters, and I will provide suggestions for how we might encourage better style from both new and experienced software developers.

The list of what we teach incoming students is long, and there are many critical concepts that they need to absorb if they are to be effective in our field. Real programmers use data structures every week, and if they don't have a strong grounding in algorithms, they'll make a major blunder every month. But the essence of software engineering is in the code, and too often we find ourselves wading through the software equivalent of this famous gem:

"In the Nuts (unground), (other than ground nuts) Order, the expression nuts shall have reference to such nuts, other than ground nuts, as would but for this amending Order not qualify as nuts (unground) (other than ground nuts) by reason of their being nuts (unground)."

(If you know what that sentence means, please write me. I've been trying to figure it out for years.)

The issue of comprehensibility is a huge hole in our current education program. Although the 2013 draft ACM curriculum mentions "documentation and style" as a required component of any CS education, the phrase is buried on page 147 as almost an afterthought, given no more attention than power sets and HTTP. (Is HTTP really so fundamental that it even deserves mention?) I claim that this neglect of style is akin to teaching English by concentrating on the common plot devices used in Hollywood thrillers—useful to those working in that specific area, but not to students who need to learn the fundamentals before attempting advanced work.

Think about it for a minute. How much of your programming time is spent writing new code, from scratch? Be honest. Is it ten percent? Five? Less? And how much time is spent working on existing code—trying to understand it, debugging it, adding shiny new features? (Most of us love adding features, because that's one of the rare times we get to write substantial new stuff.)

The reality is that we read code every day: sometimes our own, sometimes written by somebody else, and frequently a blend of the two. Reading code dominates our lives, and it only makes sense that we should try to help ourselves out by making our code easy to read. Even so, too many of us forget that fact and fall into the lazy trap of writing something that we understand at the moment but that won't make sense when we return to it in a year or two.

For example, I found the following snippet (slightly shortened for this article) in a program I use on a daily basis:

```
if (fw < 1)
    fw = 1;
if (fh < 1)
    fh = 1;
if (x + ww - fw > sw)
    x -= ww - fw;
else
    x -= fw;
if (x < 0)
    x = 0;
if (y + wh - fh > sh)
    y -= wh - fh;
else
    y -= fh;
if (y < 0)
    y = 0;
```

Wow. To be fair, this is windowing code, so we can assume the meanings of the suffixes "w" and "h". And the programmers at least had the sense to indent properly (and in the original they used curly braces consistently). But was it really necessary to limit the variable names to single characters, so that the reader must guess their purpose? Why not use max for all the limit-setting? Why are x and y limited to 0, but fw and fh to 1? And perhaps it would be helpful to add a comment explaining why, if x + ww - fw exceeds sw, we *subtract* that quantity (effectively adding fw), but otherwise we ignore ww and subtract fw! There's nothing nearby that gives a hint as to what's going on here.

The Problem

The programmers in the above case were far from incompetent. And they were clearly *trying* to write maintainable code; there are signs of care throughout the program. But in the end they produced something almost incomprehensible. What went wrong?

I believe that the fundamental difficulty is that they weren't taught how to understand what a programmer unfamiliar with the code needs. *They* knew what the variables were for, so single-letter reminders were sufficient. *They* knew why they were adjusting x and y in such an odd fashion, and it never occurred to them that an explanation might be useful. So somebody else who is trying to fix a bug in this program is left to spend hours tracing calls and analyzing the logic, or to step through with a debugger, or (all too often) to guess "Maybe if I change the -= to a +=, things will start working, and it's quicker to recompile and test than to figure out what's going on." But of course that hasty approach often introduces subtle bugs elsewhere.

And why don't programmers understand the needs of readers? There can be many causes, including inexperience, poor skills at explaining things, and even arrogance ("If you don't understand my code, you must just be stupid"). Some of these causes are difficult to address (although the world would probably be a better place if we could ship all the arrogant programmers to a desert island to argue amongst themselves about who is the smartest). But we can begin by doing a better job of teaching style.

Unfortunately, there's a chicken-and-egg problem involved: Relatively few academics have the background necessary to understand how to write maintainable code. The typical career path for a university professor is to go directly from an undergraduate degree to graduate school, and from there straight into a tenure-track position. Undergraduate students usually work only on their own code, and normally only on small programs. Graduates may work a little bit on someone else's code, but eventually they have to develop their own as part of a dissertation, and although that code may be massive (especially in systems-related specialties), it doesn't have to work particularly well and rarely has a lifetime beyond the awarding of a PhD. Because grad students spend 99% of their time working on their own code, which they necessarily understand intimately, they can get away with leaving things uncommented, choosing cryptic variable names, creating disastrously tangled logic, and even worse coding practices.

The result is that many new professors have only a vague idea of what good, maintainable code should look like. Even if they are committed to the concept of good style (and many are), their inexperience makes them poor judges of quality. It is as if we asked a literature professor to teach novel-writing when they had written only one unpublished, un-critiqued book in their lives; no matter how good their intentions, we would get a few great teachers and a plethora of extremely bad ones.

In the end, students who graduate with a bachelor's degree have spotty educations. They may be fantastic at algorithm analysis, but many write code so bad that their new employers must spend months or even years retraining them before they can be trusted to work alone. And in many cases, their bad habits lead to flawed designs, bugs, and security holes in shipped software.

A Solution?

What can be done to improve the situation? Although it's a tough nut to crack, I believe there are several approaches we can take. Much of the solution falls in the laps of colleges and universities, which, after all, have a primary mission of teaching young people how to succeed in our field.

First, we should make maintainability and coding style an important part of the grade on tests and especially on homework. Grading style is labor-intensive, so it's easy to fall into the trap

of only grading functionality (often with automated tools). But as I tell my own students, a perfectly working spaghetti program is worthless because it can't be enhanced, whereas a slightly broken but wonderfully readable program is priceless because any half-decent programmer can fix the bugs and the result will be useful for years to come. So it's worth hiring extra TAs and training them to recognize good coding practices. (You *will* have to train them at first, because they've come up through the same style-doesn't-matter ranks.)

Second, find ways to encourage students to read code. One of the best ways to learn English writing is to read the great authors, and the same holds true for software. Professors should provide their students with large sample programs and require them to be read and understood. Reading good code has a double benefit: the code provides examples of how things should be done, and it develops a skill that is essential for anyone embarking on a career in computing. (Exceptionally demanding—or downright mean—professors might also assign students to work with some astoundingly bad code, which in my experience quickly convinces students that readability matters. The Daily WTF (http://thedailywtf.com/Series/CodeSOD.aspx) is a good source of brief examples of bad programming, although many of the articles are more concerned with weak logic than unreadability.)

Third, we need to recognize the importance of industrial experience for our faculty. When universities hire professors, they should give preference to people who have worked on real production code rather than to those who charged straight through to a PhD without ever taking their eye off the ball. It doesn't take much; even a year or two in the trenches will do wonders to open a young student's eyes. (And the wise researcher will choose students who have a bit of industrial background; not only will they eventually become better faculty candidates, their own research projects will go more smoothly.)

Fourth, encourage pair programming in school settings. Working with a partner is a great way to learn how to explain your code to others and how to write in a more readable fashion. Many colleges and universities have already introduced pair programming in CS courses, so this recommendation is easy to implement.

Fifth, when bringing new grad students onto a project, assign them to maintain and enhance existing code. For example, when I joined a research group as a new grad student, we were just starting a push to turn our researchware into a robust system that we could use internally without endless crashes. In addition to working on my own research, I spent most of a year fixing bugs, which gave me an education in the system that couldn't have been duplicated any other way. The end result was that we had working software and all of the students involved had a practical understanding of maintainable code. Additionally, the original author got useful feedback on the quality of what he or she had written.

Sixth, we should make it clear to our students that "functionality first" is not an acceptable design paradigm. As part of that, we should discourage participation in functionality-only programming competitions and work to develop maintainability-focused ones. (See below for how industry can help with this goal.)

Finally, I believe that all schools should require a software engineering course as part of the standard curriculum, and that the course should teach style and maintainability.

Industry's Contribution

Although our post-secondary educational system carries the primary burden of developing new computer scientists, industry can do some things to help change the current situation.

First, when interviewing job candidates, especially new graduates, find ways to discover whether they can write good code. Famous puzzles may tell you how someone approaches a tricky problem, but they will do little to reveal whether their solution will be something your company can live with for the next decade. How much of your code base was written by a whiz kid who left an unmaintainable mess behind? Wouldn't it have been better to hire someone who worked slightly slower, but produced solid, readable code with a simple API? If you test on style, you might just find that jewel of an employee. And I can promise you that if you regularly test new graduates on the quality of their code, word will get back to their younger peers, who will then develop a strong interest in learning how to pass those tests.

Second, encourage professors to get more industry experience, ideally experience working on existing code. One way to do this is to reach out to faculty—especially young faculty—to offer them sabbatical positions or consulting opportunities. Many professors enjoy coding, are unable to do it on a daily basis, and would welcome the chance to get their hands dirty from time to time. There is nothing like experience with existing code—especially poor code—to teach the importance of style.

Third, think about ways to promote style as a first-order factor. Academia and industry sponsor lots of exciting programs for young students, such as the Google Summer of Code, the ACM Programming Competition, and the Netflix Prize. Unfortunately, the usual emphasis is on "Does it work?" rather than "Can we make this work for a decade?" A contest that required maintainability as well as innovation would be harder to judge, but it would do wonders to make students think about the long-term characteristics of their work, especially if a monetary reward were involved.

Fourth, if you don't already do code reviews, institute them. Programmers hate code reviews because they're embarrass-

ing—which is precisely why they're needed. Even the best coders can benefit from an outside eye, and if the code is good, we can all learn from it. This is one of the reasons pair programming has become so popular; it offers an instant, built-in code review process. But even in a pair-programming shop, separate code reviews can further improve quality.

What Can You Do?

Not all of us are in a position to make the changes suggested above. But we can still change ourselves and try to produce better code. First, read a good book on style. I'm fond of Kernighan and Plauger's dated but still relevant classic, *The Elements of Programming Style,* but there are many alternatives.

Second, learn from the programs you work with. Has the author made your life easy or difficult? Can you figure out what a function does without digging deep into the call tree? Is the information you want buried in a maze of macros, function pointers and virtual function calls, global variables, and messy data structures? Or is everything laid out so elegantly that you wish you could take credit?

Third, when you return to one of your own programs from several years ago, do the same analysis, and be ruthless. Can you figure out what you did, and why you did it? Is there a simpler and clearer way to do things? Has your code grown and changed over time, so that some code paths are obsolete?

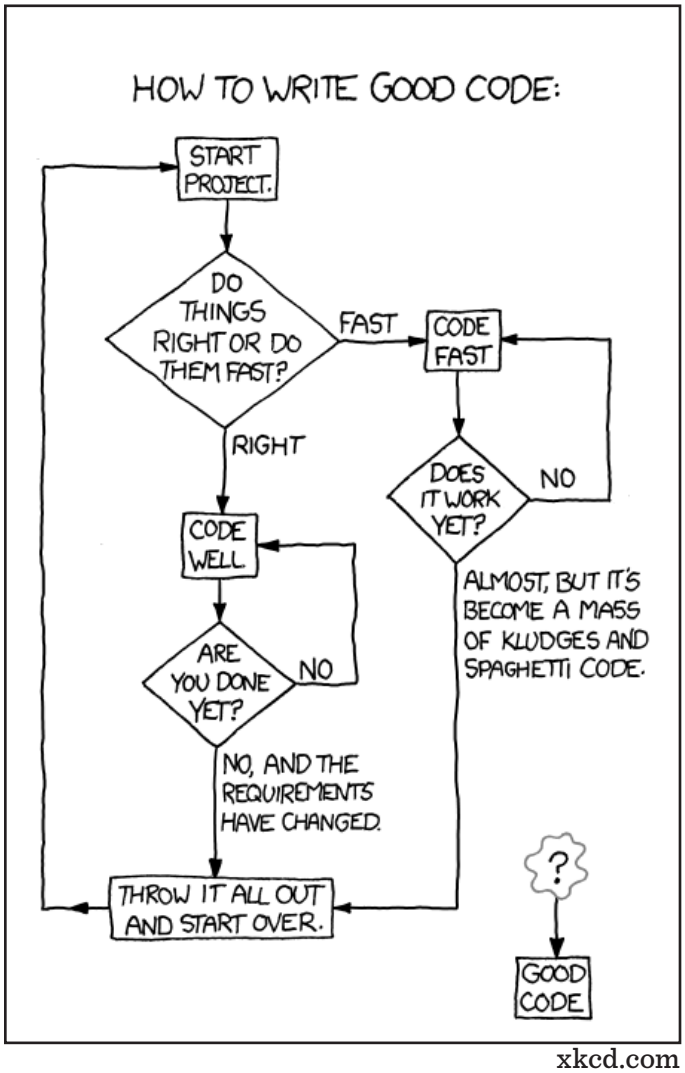
Fourth, show some of your code to a colleague and ask for honest feedback. Do you have enough comments? Are your variable names open to misinterpretation? Does it take ten minutes to figure out that clever for loop you were so proud of, the one with the null body and the tricky use of the side effects of ++? I got slapped down for that last one just a couple of weeks ago, and justifiably so. There's always room for learning.

Is It Hopeless?

As I said above, I don't think we are facing an easy task. When the ACM contest was first announced, I wrote a letter (I believe to the ACM Transactions on Computer Systems; unfortunately the ACM Digital Library doesn't seem to archive letters) suggesting that encouraging students to write hacked-up throwaway code was unwise, and perhaps the contest should instead reward what real programmers do. The suggestion was disdainfully dismissed, and 35 years later we are still lionizing undergraduates for solving toy puzzles with incomprehensible code that will be discarded the next day, never to be seen again. Is this really what we want to encourage? Are these the people you want to hire?

Nevertheless, I think progress can be made. Some of my suggestions above are easy to implement; none are impossible. We should start with baby steps, changing the world one discarded goto at a time. In fact, we have already started; the worst ideas of my youth are long gone, and no modern programmer would dare write unindented code (though, sadly, inconsistency is still rampant). So let us go forth from here and set an example by insisting that *our* students will learn to code well, our own code will be exemplary, and our new hires will earn their jobs by showing that what they write will outlast their careers.

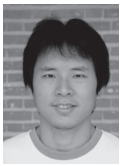
xkcd



xkcd.com

A Study of Linux File System Evolution

LANYUE LU, ANDREA C. ARPACI-DUSSEAU, REMZI H. ARPACI-DUSSEAU, AND SHAN LU



Lanyue Lu is a PhD student in Computer Sciences at the University of Wisconsin—Madison. His research interests include file systems, storage systems, and cloud computing. ll@cs.wisc.edu

Andrea Arpaci-Dusseau is a Professor and Associate Chair of Computer Sciences at the University of Wisconsin-Madison. Andrea co-leads a research group with her husband, Remzi Arpaci-Dusseau, and has advised 13 students through their PhD dissertations. She is currently a UW-Madison Vilas Associate and received the Carolyn Rosner "Excellent Educator" award; she has served on the NSF CISE Advisory Committee and as faculty Co-director of the Women in Science and Engineering (WISE) Residential Learning Community. dusseau@cs.wisc.edu

Remzi Arpaci-Dusseau is a Professor in the Computer Sciences Department at the University of Wisconsin—Madison. He received his BS in Computer Engineering summa cum laude from the University of Michigan, Ann Arbor, and MS and PhD in Computer Science from the University of California, Berkeley, under advisor David Patterson. Remzi co-leads a research group with his wife, Andrea Arpaci-Dusseau. Remzi also cares deeply about education and has won the SACM Student Choice Professor of the Year award four times and the Carolyn Rosner "Excellent Educator" award once for his efforts in teaching operating systems. remzi@cs.wisc.edu

Shan Lu is an Assistant Professor in the Computer Sciences Department at the University of Wisconsin—Madison. Her research interests include software reliability and computer systems. shanlu@cs.wisc.edu

We conducted a comprehensive study of Linux file system evolution by analyzing eight years of changes across 5,079 patches, deriving numerous new (and sometimes surprising) insights into the file-system development process. Our observations should be useful to file-system developers, systems researchers, and tool builders. Careful study of these results should bring about a new generation of more robust, reliable, and performant file systems.

A file system is not a static entity. Its code base constantly evolves through the addition of new features, repair of uncovered bugs, and improvement of performance and reliability. For young file systems, code sizes increase significantly over time. For example, ext4 nearly doubled its code size from Linux 2.6.19 (when it was introduced) to Linux 2.6.39. Even for ext3 (a stable file system), size increased more than 30% within eight years in Linux 2.6.

Patches describe how one version transforms to the next version and, thus, precisely represent the evolution of a file system code base. For open source file systems, every patch is available online, enabling us carefully to analyze in great detail how file systems change over time. A new type of "system archeology" is thus possible.

A comprehensive study of file system evolution can quantitatively answer many important questions. For example, where does the complexity of such systems lie? What types of bugs are dominant? Which performance techniques are utilized? Which reliability features exist? Is there any similarity across different file systems?

Such a study is valuable for different communities. For file system developers, they can learn from previous bug patterns to avoid repeating mistakes. They can improve existing designs by borrowing popular performance and reliability techniques. For system researchers, this study can help them identify real problems that plague existing systems, and match their research to reality. For tool builders, our study provides thousands of bug patterns, bug consequences, and performance and reliability techniques. These large-scale statistics can be leveraged to build various useful tools.

We studied six major file systems of Linux, including XFS, ext4, Btrfs, ext3, ReiserFS, and JFS. Readers may wonder why we only studied local file systems when distributed file systems are becoming increasingly important. We note that local file systems remain a critical component in modern storage, given that many recent distributed file systems, such as Google GFS and Hadoop DFS, all replicate data objects (and associated metadata) across local file systems. On smartphones and personal computers, most user data is also managed by a local file system; for example, Google Android phones use ext4 and Apple's iOS devices use HFSX.

Our study is based on manual patch inspection. We analyzed all patches of six file systems in Linux 2.6 multiple times. We have turned our manual analysis into an annotated data set, which enables us quantitatively to evaluate and study file systems in various aspects. We easily can analyze what types of patches exist, what the most common bug patterns are, how file systems reduce synchronization overhead, how file systems check for metadata corruption, and other interesting properties.

We make the following major observations:

- Bugs are prevalent in both young and mature file systems.
- Among these bugs, semantic bugs dominate.
- Over time, the number of bugs does not diminish, but rather remains a constant in a file system's lifetime.
- Data corruption and system crashes are the most common bug consequences.
- Metadata management has high bug density.
- Failure paths are particularly error-prone.
- The same performance techniques are used across file systems, whereas reliability techniques are included in a more ad hoc manner.

More results and analysis are discussed in our FAST '13 paper [3]. Another outcome of our work is an annotated data set of file-system patches, which we make publicly available for further study (at http://www.cs.wisc.edu/adsl/Traces/fs-patch).

Methodology

We chose a diverse set of file systems: XFS, ext4, Btrfs, ext3, ReiserFS, and JFS. These file systems are developed by different groups of people, use various techniques, and even represent a range of maturity. For each file system, we conducted a comprehensive study of its evolution by examining all patches from Linux 2.6.0 (Dec '03) to 2.6.39 (May '11). We manually analyzed each patch to understand its purpose and functionality, examining 5,079 patches in total.

Each patch contains a patch header, a description body, and source-code changes. The patch header is a high-level summary of the functionality of the patch (e.g., fixing a bug). The body contains more detail, such as steps to reproduce the bug, system configuration information, proposed solutions, and so forth. Given these details and our knowledge of file systems, we categorize each patch along a number of different axes, as described later.

Listing 1 shows a real ext3 patch. We can infer from the header that this patch fixes a null-pointer dereference bug. The body explains the cause of the null-pointer dereference and the location within the code. The patch also indicates that the bug was detected with Coverity [1].

```
[PATCH] fix possible NULL pointer in fs/ext3/super.c.

In fs/ext3/super.c::ext3_get_journal() at line 1675
`journal' can be NULL, but it is not handled right
(detect by Coverity's checker).

-   /fs/ext3/super.c
+++   /fs/ext3/super.c
@@  -1675,6 +1675,7 @@ journal_t *ext3_get_journal()

1   if (!journal){
2     printk(KERN_ERR "EXT3: Could not load ... ");
3     iput(journal_inode);
4 +   return NULL;
5   }
6   journal->j_private = sb;
```

Listing 1: An ext3 patch

This patch is classified as a bug (type=bug). The size is 1 (size=1), as one line of code is added. From the related source file (super.c), we infer the bug belongs to ext3's superblock management (data-structure=super). A null-pointer access is a memory bug (pattern=memory,nullptr) and can lead to a crash (consequence=crash).

Limitations: Our study is limited by the file systems we chose, which may not reflect the characteristics of other file systems. We only examined kernel patches included in Linux 2.6 mainline versions, thus omitting patches for ext3, JFS, ReiserFS, and XFS from Linux 2.4. As for bug representativeness, we only studied the bugs reported and fixed in patches, which is a biased subset; there may be (many) other bugs not yet reported.

Major Results

In this section, we present our major study results of bug and performance patches. Our results are illustrated around several key questions in the following sections.

What Do Patches Do?

We classified patches into five categories: bug fixes (*bug*), performance improvements (*performance*), reliability enhancements (*reliability*), new features (*feature*), and maintenance and refactoring (*maintenance*). Each patch usually belongs to a single category.

Figure 1(a) shows the number and relative percentages of patch types for each file system. Note that even though file systems exhibit significantly different levels of patch activity (shown by the total number of patches), the percentage breakdowns of patch types are relatively similar.

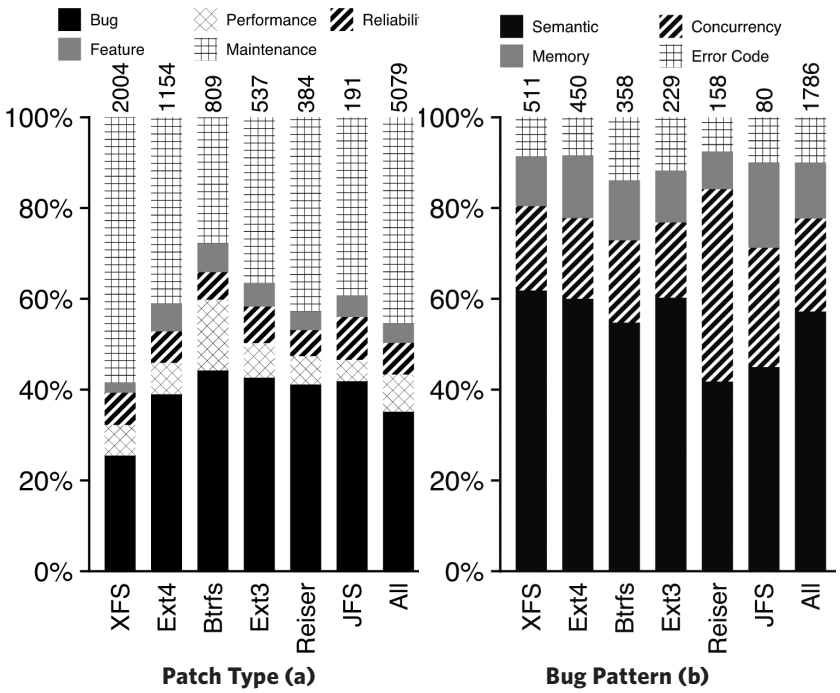


Figure 1: This figure shows the distribution of patch types and bug patterns. The total number of patches is on top of each bar.

Type	Sub-Type	Description
Semantic	State	Incorrectly update or check file-system state
	Logic	Wrong algorithm/assumption/implementation
	Config	Missed configuration
	I/O Timing	Wrong I/O requests order
	Generic	Generic semantic bugs: wrong type, typo
Concurrency	Atomicity	The atomic property for accesses is violated
	Order	The order of multiple accesses is violated
	Deadlock	Deadlock due to wrong locking order
	Miss unlock	Miss a paired unlock
	Double unlock	Unlock twice
	Wrong lock	Use the wrong lock
Memory	Resource leak	Fail to release memory resource
	Null pointer	Dereference null pointer
	Dangling Pt	Dereference freed memory
	Uninit read	Read uninitialized variables
	Double free	Free memory pointer twice
	Buf overflow	Overrun a buffer boundary
Error Code	Miss Error	Error code is not returned or checked
	Wrong Error	Return or check wrong error code

Table 1: Bug Pattern Classification. This table shows the classification and definition of file-system bugs.

Maintenance patches are the largest group across all file systems (except Btrfs, a recent and not-yet-stable file system). These patches include changes to improve readability, simplify structure, and utilize cleaner abstractions; in general, these patches represent the necessary costs of keeping a complex open-source system well-maintained. Because maintenance patches are relatively uninteresting, we do not examine them further.

Bug patches have a significant presence, comprising nearly 40% of patches across file systems. Not surprisingly, Btrfs has a larger percentage of bug patches than others; however, stable and mature file systems (such as ext3) also have a sizable percentage of bug patches, indicating that bug fixing is a constant in a file system's lifetime (see "Do Bugs Diminish Over Time?" below).

Both *performance* and *reliability* patches occur as well, although with less frequency than maintenance and bug patches. They reveal a variety of the same techniques used by different file systems. Finally, feature patches account for a small percentage of total patches; but, most of feature patches contain more lines of code than other patches.

What Do Bugs Look Like?

We partitioned file-system bugs into four categories based on their root causes. The four major categories are incorrect design or implementation (*semantic*), incorrect concurrent behaviors (*concurrency*), incorrect handling of memory objects (*memory*), and missing or wrong error code handling (*error code*). The detailed classification is shown in Table 1. Figure 1(b) shows the total number and percentage of each type of bug across file systems. There are about 1,800 bugs, providing a great opportunity to explore bug patterns at scale.

Semantic bugs dominate other types (except for ReiserFS). Most semantic bugs require file-system domain knowledge to understand, detect, and fix; generic bug-finding tools (e.g., Coverity [1]) may have a hard time finding these bugs. An example of a *logic* bug is shown in S1 of Table 2: `find_group_other()` tries to find a block group for inode allocation, but does not check all candidate groups; the result is a possible `ENOSPC` error even when the file system has free inodes.

Concurrency bugs account for about 20% of bugs on average across file systems (except for ReiserFS), providing a stark contrast to user-level software in

ext3/ialloc.c, 2.6.4 — **Semantic (S1)**

```
1     find_group_other(...){
2 -     group = parent_group + 1;
3 -     for (i = 2; i < ngroups; i++) {
4 +     group = parent_group;
5 +     for (i = 0; i < ngroups; i++) {
```

ext4/extents.c, 2.6.30 — **Concurrency (C1)**

```
1     ext4_ext_put_in_cache(...){
2 +     spin_lock(i_block_reservation_lock);
3       cex = &EXT4_I(inode)->i_cached_extent;
4...6   cex->ec_FOO = FOO; // elided for brevity
7 +     spin_unlock(i_block_reservation_lock);
```

ext3/super.c, 2.6.7 — **Memory (M1)**

```
1     ext3_get_journal(...){
2       if (!journal) {
3         ...  ...
4 +       return NULL;
5       }
6       journal->j_private = sb;
```

reiserfs/xattr_acl.c, 2.6.16 — **Error Code (E1)**

```
1     reiserfs_get_acl(...){
2       acl = posix_acl_from_disk(...);
3 -     *p_acl = posix_acl_dup(acl);
4 +     if (!IS_ERR(acl))
5 +       *p_acl = posix_acl_dup(acl);
```

ext4/resize.c, 2.6.25 — **Failure Path (F1)**

```
1     ext4_group_extend(...){
2       if (count != ext4_blocks_count(es)){
3         ext4_warning(...);
4 +       ext4_journal_stop(handle);
5         error = -EBUSY;
6         goto exit_put;
```

ext4/inode.c, 2.6.22 — **Failure Path (F2)**

```
1     ext4_read_inode(...){
2     ...  ...
3     if (inode_is_bad){
4 +     brelse(bh);
5       goto bad_inode;
6     }
```

ext4/extents.c, 2.6.31 — **Performance (P1)**

```
1     ext4_fiemap(...){
2 -     down_write(&EXT4_I(inode)->i_data_sem);
3 +     down_read(&EXT4_I(inode)->i_data_sem);
4       error = ext4_ext_walk_space(...);
5 -     up_write(&EXT4_I(inode)->i_data_sem);
6 +     up_read(&EXT4_I(inode)->i_data_sem);
```

btrfs/free-space-cache.c, 2.6.39 — **Performance (P2)**

```
1     btrfs_find_space_cluster(...){
2 +     if (bg->free_space < min_bytes){
3 +       spin_unlock(&bg->tree_lock);
4 +       return -ENOSPC;
5 +     }
6       /* start to search for blocks */
```

Table 2: Code Examples. This table shows the code examples of bug and performance patches.

which fewer than 3% of bugs are concurrency-related [2]. ReiserFS stands out along these measures because of its transition, in Linux 2.6.33, away from the Big Kernel Lock (BKL), which introduced a large number of concurrency bugs. An example of an atomicity violation bug in ext4 is shown in C1 of Table 2. For this bug, when two CPUs simultaneously allocate blocks, there is no protection for the `i_cached_extent` structure; this atomicity violation could thus cause the wrong location on disk to be read or written. A simple spin-lock resolves the bug.

There are also a fair number of memory-related bugs in all file systems; their percentages are lower than that reported in user-level software [2]. Many research and commercial tools have been developed to detect memory bugs [1, 5], and some of them are used to detect file-system bugs. An example of a null-pointer dereference bug is shown in M1 of Table 2; a return statement is missing, leading to a null-pointer dereference.

Error code bugs account for only 10% of total bugs. A missing error code example is shown in E1 of Table 2. The routine `posix_acl_from_disk()` could return an error code (line 2); however, without error checking, acl is accessed and thus the kernel crashes (line 3).

Do Bugs Diminish Over Time?

File systems mature from the initial development stage to the stable stage over time, by applying bug-fixing and performance and reliability patches. Various bug detection and testing tools are also proposed to improve file-system stability. A natural question arises: Do file-system bug patterns change over time and, if so, in what way?

Overall, our results (Figure 2) show that the number of bugs does not die down over time (even for stable file systems), but rather ebbs and flows. A great example is XFS, which under constant

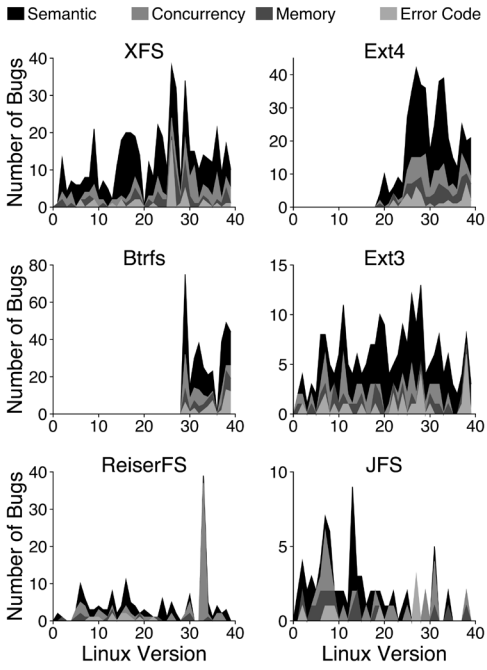


Figure 2: Bug Pattern Evolution. This figure shows the bug pattern evolution for each file system over all versions.

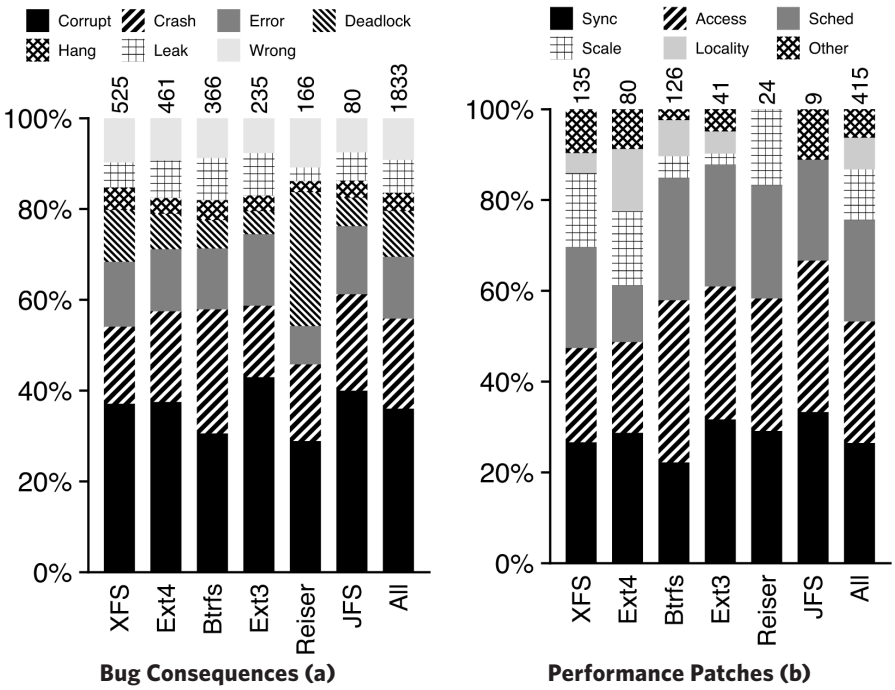


Figure 3: This figure displays the breakdown of bug consequences and performance patches. The total number of consequences and patches is shown on top of each bar. A single bug may cause multiple consequences; thus, the number of consequences instances is slightly higher than that of bugs in Figure 1(b).

development goes through various cycles of higher and lower numbers of bugs. A similar trend applies to ext3. For ext3, a new block reservation algorithm was added at Linux 2.6.10, and extended attributes in inodes were added at Linux 2.6.11. Therefore, a surge of bug patches are related to these new features. Similar things happened at 2.6.17, where a new multiple block allocation algorithm was introduced. Then, many related bug patches followed. At Linux 2.6.38, the spike is because ext3 fixed multiple error-handling bugs.

New file systems, such as ext4 and Btrfs, have a high number of bugs at the beginning of their lifetime. JFS and ReiserFS both have relatively small developer and user bases compared to the more active file systems XFS, ext4, and Btrfs. JFS does experience a decline in bug patches.

Within bugs, the relative percentage of semantic, concurrency, memory, and error code bugs varies over time but does not converge. Interesting exceptions occasionally arise (e.g., the BKL removal from ReiserFS led to a large increase in concurrency bugs in 2.6.33).

What Consequences Do Bugs Have?

As shown in Figure 1(b), there are a significant number of bugs in file systems. But how serious are these file-system bugs? We now categorize each bug by impact; such bug consequences include severe ones (data corruption, system crashes, unexpected errors, deadlocks, system hangs, and resource leaks) and other wrong behaviors.

Figure 3(a) shows the per-system breakdowns. If the patch mentions that the crash also causes corruption, then we classify this bug with multiple consequences. Data corruption is the most predominant consequence (40%), even for well-tested and mature file systems. Crashes account for the second largest percentage (20%); most crashes are caused by explicit calls to `BUG()` or `Assert()` as well as null-pointer dereferences. Unexpected errors and deadlocks occur quite frequently (just below 10% each on average), whereas other bug consequences arise less often. For example, exhibiting the wrong behavior without more serious consequences accounts for only 5-10% of consequences in file systems, whereas it is dominant in user applications [2].

Where Does Complexity Lie?

The code complexity of file systems is growing. The original FFS had only 1,200 lines of code; modern systems are notably larger, including ext4 (29K LOC), Btrfs (47K LOC), and XFS (64K LOC). Several fundamental questions are germane: How are the code and bugs distributed? Does each logical component have an equal degree of complexity?

File systems generally have similar logical components, such as inodes, superblocks, and journals. To enable fair comparison, we partition each file system into nine logical components: data block allocation (*balloc*), directory management (*dir*), extent mapping (*extent*), file read and write operations (*file*), inode metadata (*inode*), transactional support (*trans*), superblock metadata (*super*), generic tree procedures (e.g., insert an entry) (*tree*) and other supporting components (*other*).

Figure 4 shows the percentage of bugs versus the percentage of code for each of the logical components across all file systems and versions. Within a plot, if a point is above the y = x line, it means that a logical component (e.g., inodes) has more than its expected share of bugs, hinting at its complexity; a point below said line indicates a component (e.g., a tree) with relatively few bugs per line of code, thus hinting at its relative ease of implementation.

We make the following observations. First, for all file systems, the file, inode, and super components have a high bug density. The file component is high in bug density either due to bugs on

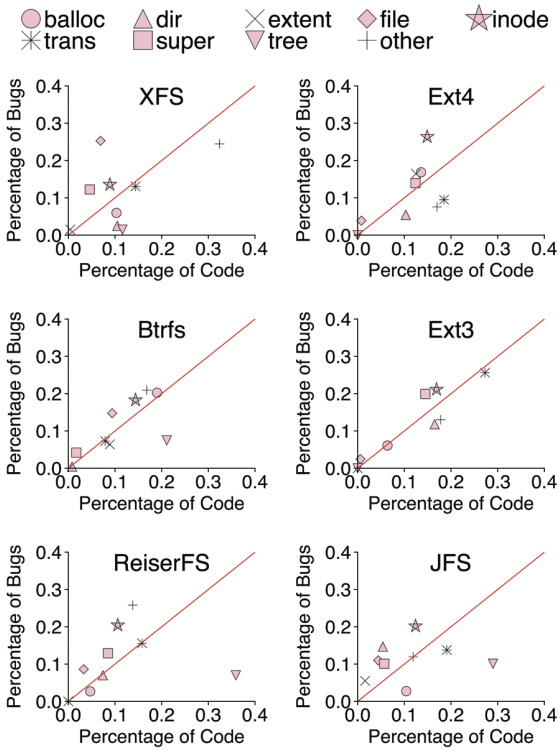


Figure 4: File System Code and Bug Correlation. This figure shows the correlation between code and bugs. The x-axis shows the average percentage of code of each component (over all versions); the y-axis shows the percentage of bugs of each component (over all versions).

the fsync path (ext3) or custom file I/O routines added for higher performance (XFS, ext4, ReiserFS, JFS), particularly so for XFS, which has a custom buffer cache and I/O manager for scalability. The inode and superblock are core metadata structures with rich and important information for files and file systems, which are widely accessed and updated; thus, it is perhaps unsurprising that a large number of bugs arise therein (e.g., forgetting to update a time field in an inode, or not properly using a superblock configuration flag).

Second, transactional code represents a substantial percentage of each code base (as shown by the relatively high x-axis values) and, for most file systems, has a proportional amount of bugs. This relationship holds for ext3 as well, even though ext3 uses a separate journaling module (JBD); ext4 (with JBD2, adding a transaction checksum to JBD) has a slightly lower percentage of bugs because it was built upon a more stable JBD from Linux 2.6.19. In summary, transactions continue to be a double-edged sword in file systems; whereas transactions improve data consistency in the presence of crashes, they often add many bugs due to their large code bases.

Third, the percentage of bugs in tree components of XFS, Btrfs, ReiserFS, and JFS is surprisingly small compared to code size.

XFS	Ext4	Btrfs	Ext3	ReiserFS	JFS
200 (39.1%)	149 (33.1%)	144 (40.2%)	88 (38.4%)	63 (39.9%)	28 (35%)

(a) By File System

Semantic	Concurrency	Memory	Error Code
283 (27.7%)	93 (25.4%)	117 (53.4%)	179 (100%)

(b) By Bug Pattern

Table 3: Failure Related Bugs. This table shows the number and percentage of the bugs related to failures in file systems.

One reason may be the care taken to implement such trees (e.g., the tree code is the only portion of ReiserFS filled with assertions). File systems should be encouraged to use appropriate data structures, even if they are complex, because they do not induce an inordinate amount of bugs.

Do Bugs Occur on Failure Paths?

Many bugs we found arose not in common-case code paths but rather in more unusual fault-handling cases. File systems need to handle a wide range of failures, including resource allocation, I/O operations, silent data corruption, and even incorrect system states. These failure paths have a unique code style. *Goto* statements are frequently used. Error codes are also propagated to indicate various failures detected. We now quantify bug occurrences on failure paths; Table 3 presents our accumulated results.

As we can see from the Table 3a, roughly a third of bugs are introduced on failure paths across all file systems. Even mature file systems such as ext3 and XFS make a significant number of mistakes on these rarer code paths.

When we break it down by bug type in Table 3b, we see that roughly a quarter of semantic bugs occur on failure paths. Once a failure happens (e.g., an I/O fails), the file system needs to free allocated disk resources and update related metadata properly; however, it is easy to forget these updates, or to perform them incorrectly. An example of a semantic bug on failure path is shown in F1 of Table 2. When ext4 detects multiple resizers run at the same time, it forgets to stop the journal to prevent potential data corruption.

A quarter of concurrency bugs arise on failure paths. Sometimes, file systems forget to unlock locks, resulting in deadlock. Moreover, when file systems output errors to users, they sometimes forget to unlock before calling blocking error-output functions (deadlock). These types of mistakes rarely arise in user-level code [4].

For memory bugs, most resource-leak bugs stem from forgetting to release allocated resources when I/O or other failures happen. There are also numerous null-pointer dereference bugs that incorrectly assume certain pointers are still valid after a

failure. An example of a memory bug on failure path is shown in F2 of Table 2. When ext4 detects a corrupted inode, it forgets to release the allocated buffer head. Finally (and obviously), all error code bugs occur on failure paths (by definition).

Testing failure-handling paths to find all types of bugs is difficult. Most previous work has focused on memory resource leaks and missing unlock and error codes; however, existing work can only detect a small portion of failure-handling errors, especially omitting a large amount of semantic bugs on failure paths. Our results provide strong motivation for improving the quality of failure-handling code in file systems.

What Performance Techniques Are Used?

Performance is critical for all file systems. Performance patches are proposed to improve existing designs or implementations. We partition these patches into six categories: inefficient usage of synchronization methods (*sync*), smarter access strategies (*access*), I/O scheduling improvement (*sched*), scale on-disk and in-memory data structures (*scale*), data block allocation optimization (*locality*), and other performance techniques (*other*). Figure 3(b) shows the breakdown.

Synchronization-based performance improvements account for more than a quarter of all performance patches across file systems. Typical solutions used include removing a pair of unnecessary locks, using finer-grained locking, and replacing write locks with read/write locks. A *sync* patch is shown in P1 of Table 2; `ext4_fiemap()` uses write instead of read semaphores, limiting concurrency.

Access patches use smarter strategies to optimize performance, including caching and work avoidance. For example, ext3 caches metadata stats in memory, avoiding I/O. Figure 3(b) shows access patches are popular. An example Btrfs access patch is shown in P2 of Table 2; before searching for free blocks, the patch first checks whether there is enough free space, avoiding unnecessary work.

Sched patches improve I/O scheduling for better performance, such as batching of writes, opportunistic readahead, and avoiding unnecessary synchrony in I/O. As can be seen, sched has a similar percentage compared to sync and access. Scale patches utilize scalable on-disk and in-memory data structures, such as hash tables, trees, and per block-group structures. XFS has a large number of scale patches, as scalability was always its priority.

Lessons Learned

Beyond the results, we also want to share several lessons we learned from this project. First, a large-scale study of file systems is feasible and valuable. Finishing this study took us one and half years. Even though the work is time-consuming, it is still manageable. A similar study may be interesting for other OS components, such as the virtual memory system.

Second, details matter. We found many interesting and important details, which may inspire new research opportunities. For example, once you know how file systems leak resources, you can build a specialized tool to detect leaks more efficiently. Once you know how file systems crash, you can improve current systems to tolerate crashes more effectively.

Third, research should match reality. New tools are highly desired for semantic bugs. More attention may be required to make failure paths correct.

Finally, history repeats itself. We observed that similar mistakes recur, both within a single file system and across different file systems. We also observed that similar (but old) performance and reliability techniques were utilized in new file systems. We should pay more attention to system history, learn from it, and use this knowledge to help build a correct, high-performance, and robust next-generation file system from the beginning.

Acknowledgments

This material is based upon work supported by the National Science Foundation under the following grants: CNS-1218405, CCF-0937959, CSR-1017518, CCF-1016924, as well as generous support from NetApp, EMC, and Google.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or other institutions.

References

[1] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," Communications of the ACM, February 2010.

[2] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu, Yuanyuan Zhou, and Chengxiang Zhai, "Have Things Changed Now?—An Empirical Study of Bug Characteristics in Modern Open Source Software," Workshop on Architectural and System Support for Improving Software Dependability (ASID '06), San Jose, California, October 2006.

[3] Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Shan Lu, "A Study of Linux File System Evolution," Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST '13), San Jose, California, February 2013.

[4] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou, "Learning from Mistakes—A Comprehensive Study on Real World Concurrency Bug Characteristics," Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII), Seattle, Washington, March 2008.

[5] Yoann Padioleau, Julia Lawall, René Rydhof Hansen, and Gilles Muller, "Documenting and Automating Collateral Evolutions in Linux Device Drivers," Proceedings of the EuroSys Conference (EuroSys '08), Glasgow, Scotland UK, March 2008.

Ted Ts'o on Linux File Systems

An Interview

RIK FARROW

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

Theodore Ts'o is the first North American Linux Kernel Developer, having started working with Linux in September 1991. He also served as the tech lead for the MIT Kerberos V5 development team, and was the architect at IBM in charge of bringing real-time Linux in support of real-time Java to the US Navy. He previously served as CTO for the Linux Foundation and is currently employed at Google. Theodore is a Debian Developer and is the maintainer of the ext4 file system in the Linux kernel. He is the maintainer and original author of the e2fsprogs userspace utilities for the ext2, ext3, and ext4 file systems.
tytso@mit.edu

I ran into Ted Ts'o during a tutorial luncheon at LISA '12, and that later sparked an email discussion. I started by asking Ted questions that had puzzled me about the early history of ext2 having to do with the performance of ext2 compared to the BSD Fast File System (FFS).

I had met Rob Kolstad, then president of BSDi, because of my interest in the AT&T lawsuit against the University of California and BSDi. BSDi was being sued for, among other things, having a phone number that could be spelled 800-ITS-UNIX. I thought that it was important for the future of open source operating systems that AT&T lose that lawsuit.

That said, when I compared the performance of early versions of Linux to the current version of BSDi, I found that they were closely matched, with one glaring exception. Unpacking tar archives using Linux (likely .9) was blazingly fast compared to BSDi. I asked Rob, and he explained that the issue had to do with synchronous writes, finally clearing up a mystery for me.

Now I had a chance to ask Ted about the story from the Linux side, as well as other questions about file systems.

Rik: After graduating from MIT, you stayed on, working in the Kerberos project. But you also wrote the e2fsprogs, which include e2fsck for ext2, at about the same time. Can you tell me how you got involved with writing key file-system utilities for the newly created Linux operating system?

Ted: I originally got started with file systems because I got tired of how long it took for ext2's fsck to run. Originally, e2fsck was an adaption of the fsck for MINIX, which Linus Torvalds had written. It was very inefficient, and read inodes in from disks multiple times.

So I got started with file systems by deciding to create a new fsck for ext2 from scratch. I did this by creating a library called libext2fs that allowed programs to easily manipulate the file system data structures. This library was originally designed for use by e2fsck, but I anticipated that it would also be useful for other applications, including dump/restore, ext2fuse, etc.

I also made a point of creating a very robust set of regression tests for fsck, consisting of small file system images with specific file system corruptions, which I could use to make sure e2fsck would be able to handle those file system corruptions correctly. As far as I know, it is one of the only file system checkers (even today) that has a regression test suite.

For the design of how to check the file system, and the order of the file system scans, I took most of my inspirations from the Bina and Emrath paper "A Faster fsck for BSD Unix" [3]. This paper described improvements to the BSD fsck for the Fast File System. As far as I know, its ideas were never adopted into BSD 4.x's fsck, probably because the changes it suggested were too disruptive. Since I was doing a reimplementation from scratch, though, it was a lot easier for me to use their ideas in e2fsprogs.

So that's how I got involved with file-system development. I started by creating the e2fsprogs utilities, and then I gradually switched the focus of my kernel development efforts from the tty layer and serial driver to the ext2 and later the ext4 file system code.

Rik: The difference between sync and async had mystified me since I first read the Lions code [1]. Later, I read the source code to BSD fsck, as well as the paper, and still didn't get it. Both express the idea that certain types of damage to the file system will not happen, and when others do, assumptions can be made about how this damage occurred. I later learned that they were talking about what would be called "ordered writes." How did Linux avoid these issues in ext2?

Ted: So the concern behind asynchronous FFS, and what BSD folks constantly would say was inherently dangerous with ext2fs, is what would happen with a power failure. Various BSD folks swore up and down that after a power failure or a kernel crash, ext2 users would surely lose data. To avoid that, with the synchronous FFS, metadata updates were written in a very careful order so that fsck could always figure out how to either roll back or complete a metadata operation.

I was never all that persuaded by this argument, since if you didn't use fsync(), the data blocks weren't getting forced to disk, so who cares if the metadata blocks were written in a very careful order? Furthermore, in practice, all modern disks (especially those engineered to get the best WinBench scores) work with writeback caching enabled, and in fact are optimized assuming that writeback caching is enabled, and this defeats the careful write ordering of FFS synchronous mode (or soft updates). You can disable writeback caching, of course, but on modern disks, this will very badly trash write performance.

For more information on soft updates and why we aren't all that impressed with it as a technology, please see Val Henson's LWN piece [2]. The one amplification I'll add to it is that soft updates are so complicated, I very much doubt there are many BSD developers beyond Greg Ganger and Kirk McKusick who understand it well enough to add new file system features. As a result, UFS didn't get extended attributes or ACL support until Kirk personally added it himself, and as far as I know, it still doesn't have online file-system resizing, while ext3/4 have had online resizing since 2004.

Rik: Ext2 had been in use for years when the decision was made to create ext3. Can you talk about the decision-making process, about what changes were needed to justify a newer version of the venerable ext2?

Ted: As hard drives started getting bigger (which back in 2000 to 2001 meant drives with 40 GB to 80 GB), the time to run fsck got larger and larger. As a result, there was a strong desire to have file systems that did not require running fsck after a power failure. So there wasn't an explicit decision-making process, as much as there was a cry from the user community demanding this feature.

Between 2010 and 2011, there were multiple efforts launched to add a journaling file system to Linux: ReiserFS, ext3, and JFS. ReiserFS actually got merged into the Linux kernel first, in version 2.4.1; however, it wasn't fully stable when it hit the mainline. Stephen Tweedie didn't submit ext3 for merging until 2.4.15, in November 2011, and at that time ext3 and ReiserFS were roughly at the same level of stability. JFS was merged into the mainline a bit later, in version 2.4.18, although like ext3, it was available in preview releases before it was deemed ready to be submitted to Linus.

For a while, ReiserFS was favored by SUSE (it had helped to fund the development of ReiserFS by Namesys) while ext3 was favored by Red Hat (since Stephen Tweedie, the primary author of the journaling feature, worked for Red Hat). JFS was donated by IBM, and at least initially it actually had somewhat better performance and scalability than the other two file systems; however, the only people who understood it were IBM employees, whereas ext3 had a much wider developer pool. As a result, it evolved faster than its siblings, and eventually became the de facto default file system for Linux.

Rik: You were involved in the decision-making process to go ahead with the design and implementation of Btrfs [4]. Can you tell us a bit about that process and your part in it?

*Ted:*At the time, Sun's ZFS was receiving a lot of attention due to a large marketing push from Sun Microsystems. So a group of Linux file system engineers, representing multiple file systems and working at many different companies, got together in November 2007 to design the requirements for a "next generation file system" with the goal of convincing the management from multiple companies to work together to make a new file system possible.

The consensus of the engineers who attended this workshop was that adding simple extensions to ext3 (to create ext4) was the fastest way to improve the capabilities of Linux's file system; however, in the long term, it would be necessary to create a new file system to provide support for more advanced features such as file system-level snapshots, and metadata and data checksumming. A number of potential file system technologies were considered for the latter, including the Digital Equipment Corporation's Advanced File System (AdvFS, which HP/Compaq had offered to make available under an open source license). Ultimately, though, Chris Mason's Btrfs was thought to have the best long-term prospects.

I warned people at the workshop that from surveying the past experience from other file system development efforts, creating an enterprise-ready, production file system would probably require 50–200 person years of efforts, and five years of calendar time. For example, the development of ZFS started in 2001, but it was not shipped as part of Solaris 10 until 2006;

however, there was concern that companies would not fund it if we stated that it would take that long, so many folks tried to claim that Btrfs would be ready in only 2–3 years. As it turns out, distributions have only been willing to support Btrfs as ready for consumer use in fall of 2012, which was indeed about five years—and it will probably be at least another two years before most system administrators will be willing to start using it on mission-critical systems.

People tend to underestimate how long it takes to get a file system ready for production use; finding and fixing all of the potential problems, and then doing performance and scalability tuning, takes a long time.

Rik: I've heard that while Google uses ext4, the extent-based version of ext3, for cluster file systems like GFS, you disable journaling. Could you explain the reasoning behind that?

Ted: It's a question of costs versus benefits. Journaling requires extra disk writes, and worse, journal commits require atomic writes, which are extremely expensive. To give an extreme example, consider a benchmark where 10K files are written with an fsync() after each file is written. Regardless of whether ext3, ext4, XFS, or ReiserFS is used, only about 30 files per second can be written; that's because the bottleneck is the CACHE FLUSH command. With journaling disabled, 575 files can be written per second. I used the following fs_mark [5] line for testing:

```
fs_mark -s 10240 -n 1000 -d /mnt
```

Against these costs, what are the benefits? First, journaling allows a machine to reboot after a crash much more quickly, since the need to run fsck on all of the file systems can be avoided. Secondly, journaling provides a guarantee that any files written after an fsync() will not be lost after a crash. Mail servers rely on this guarantee quite heavily to ensure that email won't be lost; however, cluster file systems need to survive far more than a crash of a single machine. When there are hundreds or thousands of servers, the likelihood that a hard drive dies, or a power supply fails, or a network switch servicing a rack of servers quits, is quite high. To deal with this, Google File System (GFS) stores redundant copies of each 64 MB chunk across multiple servers, distributed in such a way so that a single failure—whether of a hard drive, an entire server, or the loss of a network switch—will not cause any interruption of service. The GFS chunkserver also maintains checksums of each 64K block; if any data corruption is found, GFS will automatically fetch the chunk from another chunkserver.

Given that GFS has all of this redundancy to protect against higher level failures, the benefits of journaling at the local disk file system level are redundant. And so, if we don't need the benefits of journaling, why pay the costs of journaling? It is for this reason that Google used ext2 and never bothered switching to ext3. The ext4 file system has extent-mapped files, which are more efficient than the files mapped using indirect blocks. This is more efficient both for reading and writing files, as well as when running the file system checker (fsck). An ext2 or ext3 file system that requires 45 minutes to fsck might only require 4 or 5 minutes if the same set of files is stored on the same disk using ext4 instead.

Rik: HDD vendors are preparing to launch Shingled Magnetic Recording (SMR) drives, which have increased capacities but may perform best when the file system understands the issue of working with SMR. Do you know of any plans to support these drives in host- or coop-managed mode in Linux?

Ted: There is a saying: "Those who try to use flash as a fast disk, generally tend to be very happy. Those who try to use flash as slow memory, tend to be very frustrated." I suspect the same will be true with shingled drives. Specifically, people who use SMR drives as very fast tape drives will be very happy; however, people who try to use shingled drives as slow disk drives will be very frustrated.

For many applications, we are no longer constrained by hard drive capacity, but by seek speeds. Essentially, a 7200 RPM hard drive is capable of delivering approximately 100 seeks per second, and this has not changed in more than 10 years, even as disk capacity has been doubling every 18–24 months. In fact, if you have a big data application which required a half a petabyte of storage, what had previously required 1024 disk drives when we were using 500 GB drives, now that 3 TB disks are available, only 171 disk drives are needed. So a storage array capable of storing half a petabyte is now capable of 80% fewer seeks.

SMR drives make this problem far, far worse. As a result, so far, I'm not aware of a lot of work with shingled drives in the Linux development community. There are some research groups that are experimenting with SMR drives, but at the moment, there has been much more interest in trying to use flash devices—either very high speed, PCIe-attached flash, or dealing with the limitations of extremely inexpensive MMC flash found in mobile or consumer electronics devices.

Rik: Finally, why are there so many file systems?

Ted: There are lots of different reasons. Sometimes we have file systems for interoperability / data exchange. That explains file systems such as FAT/VFAT/MS-DOS, iso9660, HFS, NTFS, MINIX FS, FreeVXFS, BFS, QNX4, QNX6, etc. When you take a look at the list of file systems, there are more of those than most people might first suspect.

Then there are all of the various network file systems, and the reason why we have so many is because of interoperability requirements. So that explains NFS, NFS4, CIFS, AFS, 9P, etc. And the various cluster file systems: GFS2, OCFS2, and Ceph. There probably isn't much excuse for the existence of both GFS2

and OCFS2 since they fill essentially the same niche, but that's more about an accident of timing—the same reason why we have both Git and Mercurial.

There are also a number of pseudo file systems where the file system abstraction is really useful: hugetlbfs, ramfs, debugfs, sysfs, dlm, etc.

And, of course, there are the file systems used essentially for specialized bootup applications: cramfs, SquashFS, romfs.

Finally, we have file systems that are highly specialized for a specific use case, such as file systems that are designed to work on raw flash interfaces (MTD), or a file system designed to work on object-based storage devices (EXOFS).

Basically, "it's complicated," and there are a lot of different reasons.

References

[1] Lions' Commentary on UNIX 6th Edition, with Source Code: http://en.wikipedia.org/wiki/Lions%27_Commentary_on_UNIX_6th_Edition,_with_Source_Code.

[2] Val Aurora (Henson), "Soft Updates, Hard Problems": http://lwn.net/Articles/339337/.

[3] E. Bina and P. Emrath, "A Faster fsck for BSD Unix," Proceedings of the USENIX Winter Conference, January 1989.

[4] Josef Bacik, "Btrfs: The Swiss Army Knife of Storage," *;login:*, February 2012: https://www.usenix.org/publications/login/february-2012/btrfs-swiss-army-knife-storage.

[5] fs_mark: http://sourceforge.net/projects/fsmark/.

Shingled Magnetic Recording

Areal Density Increase Requires New Data Management

TIM FELDMAN AND GARTH GIBSON

Tim Feldman works on drive design at the Seagate Technology Design Center in Longmont, Colorado. His current work focuses on object storage. He also spends time randonneuring, Nordic skiing, and logging. timothy.r.feldman@seagate.com

Garth Gibson is Professor of Computer Science at Carnegie Mellon University and the co-founder and Chief Scientist at Panasas Inc. He has an MS and PhD from the University of California at Berkeley and a BMath from the University of Waterloo in Canada. Garth's research is centered on scalable storage systems, and he has had his hands in the creation of the RAID taxonomy, the SCSI command set for object storage (OSD), the PanFS scale-out parallel file system, the IETF NFS v4.1 parallel NFS extensions, and the USENIX Conference on File and Storage Technologies. garth@cs.cmu.edu

Shingled Magnetic Recording (SMR) is the next technology being deployed to increase areal density in hard disk drives (HDDs). The technology will provide the capacity growth spurt for the teens of the 21st century. SMR drives get that increased density by writing overlapping sectors, which means sectors cannot be written randomly without destroying the data in adjacent sectors. SMR drives can either maintain the current model for HDDs by performing data retention behind the scenes, or expose the underlying sector layout, so that file system developers can develop SMR-aware file systems.

The hard disk drive industry has followed its own version of Moore's Law, known as Kryder's Law [1], for decades. While gate density has increased for integrated circuits, bit density has increased at a similar compound annual growth rate of about 40% through the application of a sequence of technologies from inductive to magneto-resistive to perpendicular recording. Technologies that are still in development include Heat-Assisted Magnetic Recording and bit-patterned media, each of which has its own innovative method of packing bits even more closely together. Preceding those technologies, however, the industry is faced with the challenge of increasing areal density of perpendicular recording.

Conventional recording, shown schematically in Figure 1, uses a track pitch that is sized to match the writer gap width such that tracks do not overlap, and the reader gap width is sized such that the signal from only one track is read. Conventional recording has scaled by decreasing both the reader and writer gap sizes, which allows bits to be packed more densely in the down track direction as well as the track pitch in the cross track direction. Further decrease of the writer gap size is extremely difficult. Small write gaps do not produce enough flux density to record the magnetic domains effectively on the disk surface. But reader gap widths can continue to be scaled to narrower dimensions.

SMR, shown schematically in Figure 2 with less than one track of overlap, enables higher areal density by recording at a track pitch appropriate for the as-narrow-as-possible reader. Recording a sector at this track pitch with an as-wide-as-necessary writer means that neighboring sectors are affected. SMR records in a strict sequence and with overlap in only one direction, leaving previously recorded data in the other direction in a readable state. This overlapping is like the placement of shingles on a roof, hence the name Shingled Magnetic Recording.

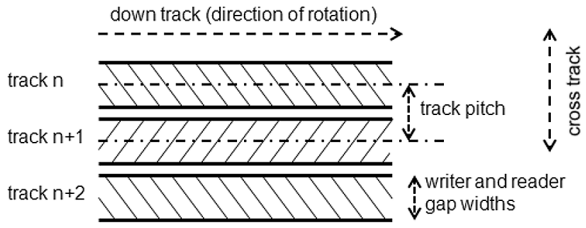


Figure 1: Schematic of conventional magnetic recording

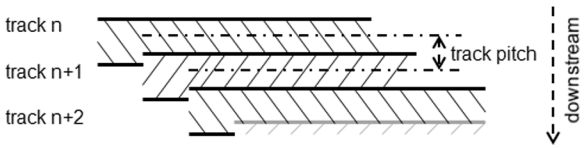


Figure 2: Schematic of Shingled Magnetic Recording

SMR Data Management Challenge

Historically, magnetic recording has used isolated areas of media for each sector such that sectors can be updated without overwriting neighboring sectors. Down track each sector is spaced sufficiently to accommodate spin speed fluctuation, and cross track they are spaced so that writes do not affect neighboring tracks. This is a match with the random block access model of the interface to disk drives. SMR breaks this model of independently writable sectors.

SMR mandates that tracks be written in the shingled direction. Sectors are still the atomic unit of media access, but SMR requires that the overlapped sectors on downstream tracks that get overwritten do not contain data of interest to the system. Either drive firmware, host software, or a combination of the two must take on the data management challenge of dealing with the data in the overlapped sectors.

Data management for SMR poses an emerging challenge for storage systems and drive design. This article covers the challenge of data placement in disk drive design, the range of solutions, and some of their issues. There are two major solution spaces. Drive-managed SMR retains the current random block write model where the most recently written data for every logical sector is retained regardless of accesses to any other sector. This is referred to as data retention in this article. Host-managed SMR, in contrast, shifts data retention responsibility to the host. This article further introduces a third SMR data management type that attempts to blend some drive- and host-managed characteristics, an approach we call cooperatively managed SMR.

Contribute to the Discussion

Host and cooperatively managed SMR are still in definition. This article serves as a notice to the systems community on the various ways SMR may impact storage design.

The industry will be defining standards for interfaces to SMR disk drives in the traditional committees: T10—SCSI Storage Interfaces and T13—ATA Storage Interface of the International Committee for Information Technology Standards (INCITS). A T10 SMR Study Group exists as a forum for discussion.

The Disk Physical Layout Model

Hard disk drive media is organized as a set of surfaces, each having at least one read/write head and each consisting of a set of tracks. The tracks are organized in concentric circles. Each track is a set of non-overlapping sectors. The sector constitutes the atomic unit of access; partial sector reads and writes are not supported.

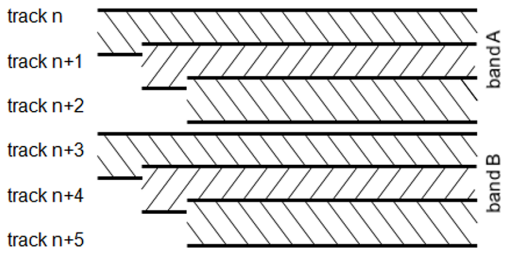


Figure 3: Schematic of Shingled Magnetic Recording with two 3-track bands

The sectors of a track are accessed consecutively as the disk rotates with respect to the head. One sector on each track is designated as being logically the first sector of the track, with subsequent sectors in turn being logically the next.

Often SMR is organized as sets of tracks that overlap each other; these are physically isolated from other sets of tracks by a gap so that there is no overlap between sets. Such a set of tracks is often called a "band." We will use this nomenclature in this article. Figure 3 shows this schematically.

Within a band the shingling happens in a single direction. Thus, the tracks of a band are overlapped much like the overlapping shingles on a roof.

Logical to Physical Mapping

Modern block command sets, notably ATA and SCSI command sets used by SATA and SAS, use a linear sector address space in which each addressable sector has an address called a logical block address, or LBA. This obfuscates the physical, three-dimensional characteristics of the drive: number of surfaces, tracks per surface, and sectors per track. It allows drives to manage defects without perturbing the host using the drive. Decoupling of logical and physical mapping has allowed drives to evolve without being synchronized to changes in host software.

A particular expectation needs to be acknowledged: LBA x and LBA x+1 are related in such a way that if LBA x is accessed, then accessing LBA x+1 is very fast. This is not an absolute requirement, and is not true 100% of the time, but it is generally the case for conventional drives.

Static Mapping

The conventional approach to mapping LBAs to physical sectors is to map the lowest LBA to the first sector on the outermost track and follows the sector progression—leaving known defective sectors unused in the mapping—and then follows the track

0	1	2	3	4	5	6	7	8	9	10	11
23	12	13	14	15	16	17	18	19	20	21	22
34	35	24	25	26	27	28	29	30	31	32	33
44	45	46	36	37	38	39	40	41	42	43	
54	55	56	57	47	48	49	50	51	52	53	
64	65	66	67	68	58	59	60	61	62	63	
73	74	75	76	77	78	69	70	71	72		
82	83	84	85	86	87	88	79	80	81		
91	92	93	94	95	96	97	98	89	90		

Figure 4: An example of a static mapping layout with tracks shown as rows of sectors labeled with their LBA

progression to map all of the rest of the LBAs. A rotational offset from the last sector on one track to the first sector of the next track is called "track skew" and allows a seek to complete in the rotational time so as to optimize sequential throughput. This mapping does not change dynamically, say in response to a new write command. There are rare exceptions to the static nature of this mapping in conventional disk drives such as when a grown defect is discovered and the LBAs for the affected sectors are remapped to spare sectors that are part of a small over-provisioning of the media for the purposes of defect management.

Figure 4 shows an example of static mapping for a drive with three tracks of 12, three tracks of 11, and three tracks of 10 sectors per track. In this example the track skew is one sector. For simplicity, this example is a single surface and has no skipped defects.

Static mapping on an SMR drive has some critical implications. The first is that an arbitrary LBA cannot be updated without affecting the data retention of the LBAs assigned to sectors overlapped by the sector to which the LBA to be updated is mapped. Accommodating this means making either a significant change in the disk's data retention model, because writing one sector modifies the data in one or more other sectors, or a significant change to write performance, because firmware must pre-read all the collaterally impacted sectors and rewrite them in downstream order. Caches and other techniques can be used to moderate either or both of these effects.

Note that with static mapping, each LBA has a couple of key characteristics determined by the set of LBAs that it overlaps. One characteristic is the distance from the written LBA to the largest overlapped LBA. We refer to this as the Isolation Distance as it describes the minimum number of unused LBAs that will isolate the written LBA from all LBAs further away. The magnitude of this distance depends on the downtrack overlap of the write, number of sectors per track, track skew, and skipped defects. Another characteristic is that for each written LBA there is an extent of contiguous LBAs that it does not overlap, ending at the largest higher LBA that the LBA does overlap. We refer to the size of this extent as the No Overlap Range as it describes a range within which writes do not affect other LBAs. The size again depends on the number of sectors per track, track skew, and skipped defects. These distances can be used by the data management scheme as is described later in the section on Caveat Scriptor.

Figure 5 repeats the layout example of Figure 4, with a writer overlap of two neighboring tracks and with LBAs increasing in the downstream direction. This means, for example, that LBA 0 overlaps LBAs 23 and 34; thus, its Isolation Distance is 34. The extent following LBA 0 that is not overlapped extends to LBA 11; thus, its No Overlap Range is 12. In contrast, LBA 68 overlaps LBAs 76, 77, 85, and 86 for a Isolation Distance of 18. The extent following LBA 68 that is not overlapped goes through LBA 75 for a No Overlap Range of 8.

Figure 5 shows that for static mapping, maintaining the data in every LBA requires all downstream LBAs to be read and then rewritten. For instance, a write to LBA 68 not only requires LBAs 76, 77, 85, and 86 to be read and then rewritten, but also LBAs 94 and 95 because writes to LBAs 76 and 85 overlap LBA 94, and writes to LBAs 77 and 86 overlap LBA 95. A simpler data retention algorithm is to read and then rewrite all higher LBAs to the end of the band; thus, a random write may, on average, cause half of its band to be read and rewritten. Alternatively, if data retention is not required, then LBAs can be updated in place. A simple model is that writing an LBA can cause loss of data retention in all higher LBAs to the end of the band.

Dynamic Mapping

An alternative to static mapping is to allow LBAs to be mapped to physical sectors dynamically by drive firmware. This is analogous to the Flash Translation Layer (FTL) model for solid state drives (SSD).

Specifically, an SMR drive can employ dynamic mapping in which it maintains a logical to physical map, sometimes called a

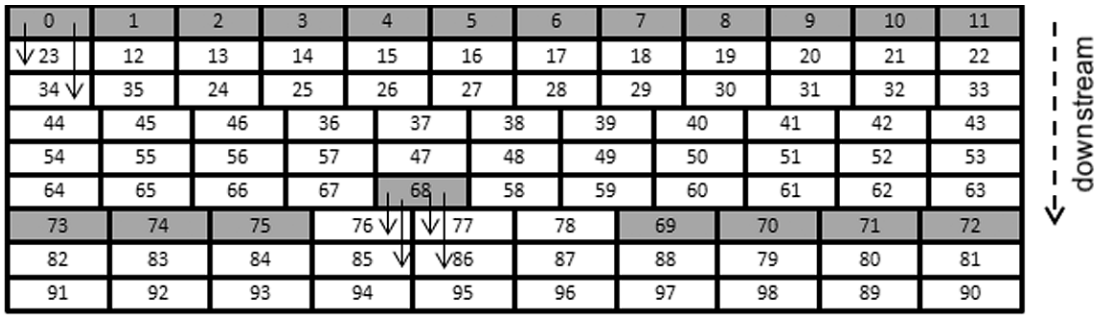


Figure 5: The static mapping layout example with shading indicating selected no overlap ranges and arrows indicating selected overlaps for a two-track overlap width

forward map in SSD, and assign the LBAs for write commands based on the drive's internal mapping policies. The forward map must then be referenced to satisfy read requests to determine which sectors are currently assigned to the requested LBAs.

Any and all of the techniques in an SSD FTL can be leveraged on an SMR drive with dynamic mapping. This includes policies for write performance so that new data can be placed in sectors that do not overlap data that must be retained, policies for read performance so that a minimum number of media accesses are required, and garbage collection so that data requiring retention can be relocated before media is reused.

Data Management for SMR

Handling data accesses, their performance, power, data retention impact, and restrictions on access patterns collectively are the data management done by a drive. This section covers the solution space for SMR.

Choices in SMR Data Management

SMR data management makes specific choices in data retention, restrictions on data accesses, the physical sector layout, and the logical to physical mapping. Specific examples of data management choices are described later in the sections on drive- and host-managed SMR.

Conventional drives deliver complete data retention for all LBAs at all times to a specified error rate, such as 1 nonrecoverable read error per 10^{15} bits read. SMR drives can deliver the same data retention model, or explicitly embrace a different model in which LBAs may not have data retention depending on the sequence of writes to other LBAs.

Conventional drives allow an LBA to be accessed at any time, either read or write accesses. SMR drives can deliver the same data access model, or explicitly embrace a different model in which only specific LBAs may be written and specific LBAs may be read depending on the state of the drive. The pertinent state is expected to be dependent on the sequence of writes and, possibly, temporal separation between the writes.

SMR data management often makes use of many mutually isolated bands of tracks. The bands may be constant in the number of tracks, or might be constant in the number of sectors. The specifics of where band boundaries are in the physical sector layout are a choice of SMR data management.

SMR data management has choices of what logical to physical mapping to employ. Static or dynamic mapping can be used. Dynamic mapping has a wide range of choices that include examples from Flash Translation Layers and other innovations [2].

Drive-Managed SMR

In drive-managed SMR, the drive autonomously delivers the conventional data retention model of maintaining the data of every LBA without any restrictions on the host access patterns. No changes are needed to the interface for drive-managed SMR. Choices of layout and mapping do not need explicitly to be exposed externally, but the choices do impact the performance and power profiles. Drive-managed SMR is responsible for garbage collection if the mapping choice can leave unmapped data in physical sectors. Drive-managed SMR is likely to be stateful in that the performance may be dependent on the state of the drive as determined by the usage history. Provisioning of additional memory and storage resources typically provides a choice of better performance at the cost of the expense of those resources.

Drive-managed SMR is a data management approach that can most directly leverage the technologies developed for SSD and FTLs. This includes over-provisioning of media. For instance, sometimes an SSD is populated with N gibibytes of Flash media but delivers N billion bytes of usable host capacity, in which case the SSD has the approximately 7% difference between 2^{30} and 10^9 as over-provisioning. Similar over-provisioning is possible in an SMR drive.

Drive-managed SMR allows an SMR drive to be used in any existing storage stack, albeit with a different performance and power profile compared to conventional drives.

Host-Managed SMR

The term "host-managed SMR" is defined to mean an SMR drive that does not autonomously deliver data retention for every LBA or restricts host accesses. Changes to the interface may be needed for host-managed SMR.

Strictly Append is a type of host-managed SMR that restricts host writes to occur only at the append point of a band. The append point is the ending position of the last write to the band; that is, an appending write implicitly moves the append point. Strictly Append also restricts reads to occur only before the append point of a band. That is, only written LBAs can be read. In its simplest implementation, Strictly Append presents a single band, and the drive may be written once in strictly sequential LBA order. ShingledFS [3] is a file system for Hadoop that uses this model. More complexity and versatility can be added by supporting multiple bands and thus multiple append points, and by allowing reuse of a band after an explicit event that moves the append point.

Exposed SMR is a different type of host-managed SMR where the host is aware of the layout and mapping. By specification or query through the interface, the host knows the details of the location of bands. Static mapping is the obvious choice such that each band is a consecutive set of LBAs. With this information, a host can know that writing an LBA obviates the data retention of all subsequent LBAs to the end of the band. Exposed SMR does not restrict host accesses, but instead moves the ownership of the data retention model to the host. This has further impact on defect management and other reliability constraints that are beyond the scope of this article. A specific Exposed SMR proposal, Caveat Scriptor, is described in a later section.

The logical to physical mapping for host-managed SMR does not have to be static; however, within a band, LBAs must be mapped to sectors that do not overlap sectors mapped to lower LBAs. This blurs the distinction between logical blocks and physical sectors. Nonetheless, the LBA is retained as the address semantic, which, for instance, allows dynamic mapping of defects.

Host-managed SMR can include a small fraction of unshingled space, some unshingled bands, for random writes.

Cooperatively Managed SMR

Cooperatively managed SMR is a type of SMR data management that is not purely drive or host managed, but has characteristics of each. For instance, bands may have append points but perhaps not require all writes to be at the append point. Band locations may be exposed to the host and explicit methods may need to be invoked to move the append point. A specific cooperatively managed SMR proposal, Coop, is described in a later section.

Alignment of Drive-Managed SMR to Applications

Drive-managed SMR delivers drives that have performance profiles that are notably different from conventional drives. Write performance is commonly sensitive to the availability of safe-to-write sectors, which in turn can be a function of the number and location of stale sectors. A drive that does internal garbage collection may sometimes be ready to accept a burst of new writes, or may have to proceed in its garbage collection to service new writes. This is the scope of a file system problem brought into the domain of the disk drive.

Read performance is sensitive to data layout. If dynamic mapping is part of the drive-managed SMR policies, LBA x and LBA x+1 can frequently not be proximate to each other, causing the read of a single LBA extent to require multiple disk media accesses. This read fragmentation issue is the typical file fragmentation problem brought into the domain of the disk drive. Drive-managed SMR includes the memory and storage resources and the embedded computing costs for over-provisioning and the FTL-like firmware.

Despite the performance differences with respect to conventional drives, drive-managed SMR is well aligned to many applications. Not only can it be deployed without modifications to the host, it is also a good match to the requirements in a lot of markets. This section describes the alignment of selected use cases to drive-managed SMR.

Personal External Drives, Backup Storage and Archive

External drives for personal use and backup or archival storage are suitable applications for drive-managed SMR. The ingress of data is very bursty and sequential enough that the drive can handle writes efficiently. Long idle times between writes and reads allow the drive to defragment appropriately and prepare for subsequent write workloads. Low duty cycle and low performance requirements help the introduction of new technology, too. A paper on deduplication of desktop VMs [4] discovered that as much as 85% of desktop data collected from Microsoft developers disk traces is write-once.

Log-Structured Files Systems and Copy-on-Write

With log-structure file systems (LFS) and copy-on-write (COW) policies in databases, file systems and other applications create a drive workload that is purposefully dominated by sequential writing. Drive-managed SMR can be optimized to handle a sequential write stream efficiently, making these applications a good match.

Applications with Writes Primarily Small or Spatially Dense

Natural workloads always have some spatial locality. Sufficient spatial locality makes a limited amount of over-provisioning useful for drive-managed SMR just as it does for SSD. Many workloads are dominated by relatively small random writes of 4 KiB or so. Databases, online transaction processing, and many other applications commonly exhibit these traits. Such workloads are a good match for FTL-style technologies, and in fact can lead to drive-managed SMR performance that is superior to conventional drives—if the writes are small enough and/or the spatial density is high enough.

Applications Dominated by Reads

Drive-managed SMR that bounds the read fragmentation can have read performance that is at or near parity with conventional drives. Applications such as content distribution, Web servers, and reference material hosting such as wikis are dominated by reads. These applications are a good match for drive-managed SMR.

Legacy and Installed Base

The most important quality of drive-managed SMR is that it conforms to the same protocol and data retention model as conventional drives, albeit with a different performance profile. Drive-managed SMR allows the areal density increase of SMR to be employed in a legacy application and serves the entire installed base of disk-based storage.

Alignment of Host and Cooperatively Managed SMR to Applications

Acknowledging that drive-managed SMR has different performance means that some applications, if unmodified for SMR, will have performance sensitivities for which drive-managed SMR is not always an optimal match. This is the main motivation for considering host and cooperatively managed SMR and its attendant impact to host implementations.

Sequential Write Workloads

While drive-managed SMR can be optimized for sequential writes, it does not always deliver conventional drive performance. In particular, if a sequential write does not go all the way from LBA 0 to LBA max, and in natural workloads sequential writes never span the whole capacity of the drive, there is a start and end to each sequential write. When the start and end do not align with band boundaries for the logical to physical mapping of the drive, there is work required in the drive to "mend" the data at the edges of the write. Host and cooperatively managed SMR provide the context in which sequential writes can be restricted to start and end at band boundaries. These schemes additionally deliver read performance with fragmentation only at band boundaries, which closely approximates conventional read performance.

Log-Structured Files Systems and Copy-on-Write

While the LFS and COW are generally a good match for drive-managed SMR, they eventually have a garbage collection requirement so that space can be reused. Garbage collection on an undifferentiated LBA space is likely to produce the same sort of performance challenges just described for sequential write workloads in general. Host and cooperatively managed SMR are an opportunity for garbage collection that is optimized for SMR.

High Performance Storage

Lastly, given the opportunity to purpose-build a storage system for SMR, host and cooperatively managed SMR enable the system to be optimized for performance. Such systems may further optimize the over-provisioning and other attributes that contribute to cost, power, and reliability.

Caveat Scriptor: An Exposed SMR Proposal

Caveat Scriptor is Latin for "let the writer beware" and is used here as a name for a more specific proposal for Exposed SMR. The layout model for Caveat Scriptor is static mapping with critical drive parameters exposed.

Drive Parameters

As described in the section on static mapping, above, each LBA has two notable parameters: No Overlap Range and Isolation Distance.

Remember that No Overlap Range is the minimum distance of contiguous, non-overlapping LBAs that follow each written LBA, and Isolation Distance is the maximum LBA distance in which some LBA might be overlapped. An Exposed SMR drive could simply make these parameters available to the host for every LBA. A Caveat Scriptor drive instead exposes a single No Overlap Range and Isolation Distance value that apply to every LBA. It determines the possible drive parameters as follows:

- Drive No Overlap Range <= minimum (No Overlap Range for all LBAs)
- Drive Isolation Distance >= maximum (Isolation Distance for all LBAs)

For a given model of Caveat Scriptor drives, all will have the same DNOR and DID values. That is, Caveat Scriptor selects a Drive No Overlap Range (DNOR) to be small enough for all drives of the model, and a Drive Isolation Distance (DID) to be large enough for all drives of its model. This allows software to be specialized to a model and not to individual drives.

For example, for a model of drives in which all layouts are described by Figure 5, the minimum No Overlap Range is at LBA 68 where the following no overlap extent goes through LBA 75,

so DNOR is 8, and the maximum Isolation Distance is at LBA 0 as described previously, so DID is 34.

Host Band Construction

With the DNOR and DID parameters, the determination of band boundaries is left up to the host. Leaving at least DID LBAs unused between bands is sufficient to provide isolation. In the example of Figure 3, 34 LBAs is the amount of unused space required to isolate bands; LBAs 0 to 29 could constitute a 30-sector band, LBAs 64 to 98 a second 35-track band, with LBAs 30 to 63 as the 34 unused sectors that isolate the two.

Three specific uses cases are described:

1. Random write band: Making a band no bigger than DNOR LBAs creates a random write band if the range is sufficiently isolated by DID LBAs on both ends. A band of this size has the attribute that no LBA in the band is overlapped by any in-use LBA—that is, LBAs that are used as part of the band isolation. Such a band is one whose LBAs can be randomly written without obviating the data retention of any in-use LBA. In the example of Figure 3, 8 LBAs is the maximum random write band size; LBAs 50 to 57, inclusive, can be a random write band. Note that DID will typically be much larger than DNOR, so random write bands are inefficient in their ratio of in-use to not-in-use LBAs.
2. Sequential write band: A band of any size that is sufficiently isolated by DID LBAs can be used as a sequential write band in which data is retained for LBAs that precede the most recent write. Such a band has no LBAs that are overlapped by LBAs in a different band, and no LBAs overlap any LBA in a different band.
3. Circular buffer band: A band can be managed as a circular buffer if a sufficient distance is maintained between the end and the start. The minimum required distance is DID. Thus the effective size of a circular buffer is less than its band by at least DID. A circular buffer could be used, for instance, to have intra-band garbage collection in which non-stale data is shuttled from the start to the end. In this instance, when stale data is present at the sta rt of the buffer the start position can traverse forward without a concomitant copying of data to the end, thus increasing the distance from the end to the start and allowing new data to be added to the buffer.
4. In the example of Figure 3, if all 99 sectors are used as a single circular buffer band and the end of the band is, say, at LBA 40, then the start must not be in the LBA range 41 to 74, inclusive. Figure 6 shows this state. Before data can be added at LBA 41, LBA 75 must become unused or stale to comply with the spacing requirement of DID = 34.

Value Proposition

The Caveat Scriptor Exposed SMR proposal delivers the following value propositions.

- Performant: Fast, static mapping can be used with all accesses going straight to media.
- Predictable: There is a low probability of internal drive management operations causing response times that the host does not expect.
- Versatile: Circular buffers can be deployed as well as random and sequential bands.
- Efficient: Isolation occurs only where the host needs LBA extents to be isolated.
- Flexible: Hosts can construct bands of any size.
- Host-owned data retention: The data retention of logical blocks is determined by the host, matching the usage model of the storage stack.

0	1	2	3	4	5	6	7	8	9	10	11
23	12	13	14	15	16	17	18	19	20	21	22
34	35	24	25	26	27	28	29	30	31	32	33
44	45	46	36	37	38	39	40	41	42	43	
54	55	56	57	47	48	49	50	51	52	53	
64	65	66	67	68	58	59	60	61	62	63	
73	74	75	76	77	78	69	70	71	72		
82	83	84	85	86	87	88	79	80	81		
91	92	93	94	95	96	97	98	89	90		

Figure 6: The static mapping layout example deployed as a circular buffer with its start at LBA 40 and its end at LBA 95. The shading shows 34 LBAs that are unused between the start and end of the circular buffer.

Coop: A Cooperatively Managed SMR Proposal

Coop is a specific proposal for cooperatively managed SMR. It blends some characteristics of drive-managed SMR with some of host-managed SMR. Coop has the data retention model of drive-managed SMR and the performance characteristics of host-managed SMR when the host conforms to a constrained band reuse model.

Coop is targeted to applications that are dominated by sequential writes through large LBA extents, optionally with a small set of randomly written extents. Coop additionally targets applications where there may be infrequent exceptions to the sequential write behavior even outside the randomly written extents.

Band Life Cycle

Coop bands go through a cycle of state transitions from empty to filling to full and back to empty. The full to empty state transition occurs due to an explicit host command such as Trim that unmaps the entire LBA extent of a band. This command can also be issued to a band in the filling state, moving it to the empty state.

Well-Known Bands and High Water Marks

Coop is built on a layout model of same-sized bands and regularly placed band boundaries. Band boundaries are at strict integer multiples of the band size. Each band has a High Water Mark that represents the highest address written since the most recent empty state. The High Water Mark is the optimum write location, but is not an enforced append point.

It is proposed that the band size is standardized to either 256 MiB or 1 GiB. These power-of-two sizes are sufficiently large to allow for a minimum of space to be devoted to band isolation.

Host Policies

The host write behavior on a Coop drive should be dominated by writes at the High Water Mark of the respective band. Writes at the High Water Mark can be serviced by conventional policies since higher LBAs are "trimmed" and do not require data retention. Writes not at the High Water Mark, at either lower or higher LBAs, are allowed and impact the drive policies as described in the next subsection.

Host read behavior is not restricted. Hosts may read trimmed LBAs.

Before reusing a band, it is incumbent on the host to issue the appropriate command to unmap the whole band. Before issuing this command the host must first copy any non-stale data to some other band. Garbage collection in a Coop drive is the responsibility of the host.

Drive Policies

Writes not at the High Water Mark may need to be serviced with drive-managed-style data management techniques. Note that writes not at the High Water Mark but within the No Overlap Range can potentially be optimized with policies that are similar to conventional data management.

Support for a small set of randomly written extents is also provided through drive-managed-style data management, possibly with an appropriate amount of over-provisioning. The amount of over-provisioning is likely to determine the amount of randomly written space that can be handled with higher performance.

Reads comply with the full data retention model of Coop. Reads of mapped sectors return the most recently written data. Reads of unmapped sectors return the appropriate unmapped-sector data pattern, possibly all zeros. For bands that have been written in strict sequential order, reads of LBAs below the High Water Mark of the respective band return the most recently written data, and reads above the High Water Mark return the appropriate unmapped-sector pattern.

Value Proposition

The Coop proposal for cooperatively managed SMR delivers the following value propositions:

- Performant: Fast, static mapping can be used for bands that are sequentially written with sequential writes and all reads below the High Water Mark serviced directly from media. Drive performance for sequential writes at the respective High Water Mark will be like that of a conventional drive.
- Tolerant: Not all random writes have to be eliminated, just minimized. Software can be deployed without 100% removal of random writes.
- Versatile: The targeted applications represent a diverse set of common use cases.
- Efficient: The amount of over-provisioning can be bounded by the amount of randomly written space and the frequency of writes that are not at a High Water Mark.
- Low barriers to adoption: The conventional data retention model and standard commands allow straightforward adoption.
- Flexible: Random write extent locations can be anywhere in LBA space and can be non-stationary.
- Standardized: Current standard command sets continue to be used, albeit likely with a few additional queries for discovery of parameters and High Water Mark values.

Further Work

Areal Density Gains

Shingled Magnetic Recording offers the opportunity for disk drives to continue to deliver increasing areal density. The recording subsystem and the head, disk, and channel designs need to evolve to take maximum advantage of SMR.

Harvesting the areal density requires more than recording subsystem work. Storage systems need to prepare file systems, application software, and utilities to be well suited to SMR data management at the drive.

Call to Action

Caveat Scriptor and Coop are two proposals for SMR interfaces. These proposals and others will be discussed at the T10 SMR Study Group, the open forum where changes to the SCSI standard are being discussed. Now is the time to add your voice to help move the technology in the best possible direction.

References

[1] C. Walter, "Kryder's Law," Scientific American, July 25, 2005: http://www.scientificamerican.com/article.cfm?id=kryders-law.

[2] Tim Feldman, Seagate, US Patent Application 20070174582.

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

[4] Dutch T. Meyer, William J. Bolosky, "A Study of Practical Deduplication": http://www.usenix.org/event/fast11/tech/full_papers/Meyer.pdf.

Ganeti: Cluster Virtualization Manager

GUIDO TROTTER AND TOM LIMONCELLI

Guido is a Senior Engineer at Google Munich, where he leads the development of Ganeti, an open source cluster virtualization manager. His interests and speaking topics are in virtualization, distributed systems, and advanced networking for Linux. He has presented Ganeti and taught classes about it at LISA, Fosdem, KVM Forum, LinuxCon, and many more gatherings. He is also a Debian developer and active member of the community. His activities can mostly be seen through the Ganeti lists (ganeti@googlegroups.com and ganeti-devel@googlegroups.com). gt@google.com

Tom is an internationally recognized author, speaker, and system administrator. His best known books include Time Management for System Administrators (O'Reilly) and The Practice of System and Network Administration (Addison-Wesley). In 2005 he received the USENIX SAGE Outstanding Achievement Award. He blogs at http://EverythingSysadmin.com. tal@everythingsysadmin.com

Ganeti is a software stack that allows easily managing a collection of physical machines (nodes) to host virtualized machines (instances). This article explains its background, principles, usage, and direction.

Virtualization and the Cloud

Virtualization is an important building block in today's computer infrastructures. Whether you're running a small lab or a huge datacenter, the decoupling of the physical hardware from the services that run on it is key for increasing uptime, resource utilization, and maintainability.

Many solutions exist both in the open source world and in the proprietary one for virtualizing workloads. Some of these solutions are stand-alone, whereas others require many different components in order to work. All of them are based on the same basic necessary components: (1) a hypervisor, which provides the basic layer on which the virtualized system runs; (2) a storage backend that hosts VMs' data; and (3) a network infrastructure that allows VMs to send and receive traffic. Optionally, extra software can be used to run and manage these systems, and provide abstractions on top of them. Ganeti is software that fulfills this role, and it can be used to manage the previously listed components.

Why Ganeti?

People choose Ganeti to run their virtualized infrastructure for many reasons. Most of them are captured by the low barrier to entry at which they can get started on Ganeti; they don't need to configure a huge system if all they need is to build a small infrastructure while, at the same time, retaining the option to scale easily to thousands of nodes.

Other reasons include its architecture (see below), which they can easily plug in to and extend, and the friendly community. Ganeti also has a good test and QA suite, which hopefully reflect in high-quality releases.

Of course, like anything, Ganeti doesn't suit all needs, but we'd like you to try it and see if it serves your organization. We're interested in getting feedback about your experience, what does or doesn't work for you. Ganeti is an enterprise production-ready system, used at Google and in many other organizations, which might help you build part of your infrastructure.

Building Blocks

As mentioned above, virtualization hosting needs many different building blocks in order to work. We'll now go through the most important ones, focusing on technologies that are supported by Ganeti. In each section we'll show how we can mix and match the underlying technology to get a virtualized environment that is customized to our needs, while being managed in a common way.

Note that the ability to "mix and match" is a key feature of Ganeti often not found in other environments, which tend to pick all the technologies that one must use; this makes Ganeti a powerful "building block" rather than a prebuilt "one-size-fits-all" system.

Hypervisor

The following hypervisors are supported by Ganeti:

- Xen is a microkernel style stand-alone hypervisor that runs "under" its guest systems and provides them resources with the help of one or more trusted VMs (e.g., dom0). Xen can run different types of virtualized environments depending on the guests' support.
- KVM is a system to run a hardware-aided virtualization environment on top of a standard Linux system.
- LXC (experimental support) is a way of running independent containers on top of the same kernel. The virtualization level is provided by giving the processes running in each container a different "view" of the system by insulating its namespaces (user, network, process, etc.).

Xen and KVM are two of the most adopted open source hypervisors available today, especially in the server world. Using them, you can easily configure one machine to run virtualized workloads. The two have a different architecture and approach to virtualizing machines.

Naturally, they are not the only options, and research is needed to see which virtualization system is a better fit for your infrastructure and workload.

Storage

Possible storage solutions for Ganeti virtualized environments include:

- Simple files residing on the host machine
- Local volumes on the host machines (e.g., physical block devices or LVM ones)
- Local data with over-the-network replication (e.g., using DRBD)
- Volumes (or files) hosted on an external dedicated hardware (SAN)
- Volumes hosted on a distributed object store (e.g., RADOS)

Ganeti will automatically manage disks and create/delete them as needed for its instances. Stand-alone management of disks is not supported yet but is considered in the roadmap.

Networking

Ganeti allows running VMs with any number of virtual network interfaces. Each of them can be connected to a Linux kernel bridge, an Open vSwitch, or have its traffic routed by the host system.

Currently, all "connection" systems (bridges, Open vSwitches, routing) must be configured on the nodes, but dynamic connection/disconnection of them is in the product roadmap.

"The Cloud"

Although the cloud is a vague term, in the virtualization context it is usually used to mean "Infrastructure as a Service" (IaaS). This is usually implemented as a collection of software providing an API that can be used to provide the resources described above. Ganeti can be used as a basic building block to provide such a service and effectively run a "private cloud."

Other Choices

Of course it might happen that your favorite technology is not yet working well with Ganeti. Although we couldn't implement all possible choices, we tried to keep our APIs standard; as such, perhaps it will be possible with some development work to design and support your architecture and make it part of the Ganeti ecosystem. After all, most of these choices were added during the product lifetime, rather than being born into it, and as such we hope to be able to accommodate more in the future, as the product and the technology space develops.

Ganeti's Principles

Ganeti is built on the following ideas:

- Extremely simple to install and use: We strive to make Ganeti as simple as possible, not requiring hand configuration of many different components, at least for a basic usage.
- Usable as a basic infrastructure building block: A Ganeti cluster must be able to host your DNS, DHCP, and many other basic services. As such, Ganeti itself will not depend on them at startup time, allowing you to cold-power-on a cluster without those services. Some of them may still be needed during normal run, for example, to add new nodes or instances.
- Multi platform: We want Ganeti to run on and with your favorite distribution of Linux, hypervisor, backend storage, and networking system. We try to make sure it exports a customizable interface that you can use to plug in your customizations.
- Simple software dependencies: We try not to depend on too new libraries and languages, to make the system easy to build and deploy. To be extremely conservative, we chose Debian stable (plus, occasionally, backports.org) as our reference for choosing library dependencies.
- Minimal hardware/platform/system dependencies: We avoid depending on any component or infrastructure choice that we feel is not strictly necessary. We are happy to support extra features when those are present, but we don't want to impose them on everybody's installation.
- Good open source citizen: We discuss designs and code on the public development list before committing to it. We make sure our entire development process is transparent, and we don't do "code dumps" at release time. This allows for ease of cooperation between members of the community.

We encourage external contributions and designs, and are happy to cooperate on the direction of this software platform. Issues as well as designs that we're working on are publicly visible, as is each code change and the related discussion before it gets merged. Patches from team members, other Google teams, or external contributors go through exactly the same review process in order to become part of the tree.

Quick Ganeti Example

Ganeti administration is done through command-line tools or the Ganeti Remote API (RAPI). Web-based admin tools have also been created, such as the Ganeti Web Manager project at Oregon State University Open Source Lab (http://ganeti-web-mgr.readthedocs.org/en/latest/); however, this being *;login:,* we will present some command-line examples.

Ganeti commands all start with `gnt-` and require a subcommand. For example `gnt-instance info foo` outputs information about an instance named `foo`.

Initializing the Cluster

This is the first step for setting up Ganeti and requires an unused host name for the cluster, associated with an unused IP address. Ganeti will set up the IP address automatically on the master node:

```
# gnt-cluster init [-s secondary_ip] cluster.example.com
```

Note that the basic initialization has many default assumptions. You may want to configure your enabled hypervisors, their parameters, and much more. See `gnt-cluster modify` for more information. The `-s` parameter configures the cluster with a separate replication network. If it is set, all nodes will also need to be added specifying the `-s` option, and their secondary IP.

Creating an Instance

Creating an instance can be simple if cluster-wide defaults have been set; it can be as simple as specifying an operating system image name, and amount of RAM to allocate to the VM, size of the virtual disk to be created, the storage type, and the instance's name:

```
# gnt-instance add -o ubuntu_std -B memory=1024M -s 100G -t drbd inst1.example.com
Thu Mar 21 14:16:04 2013 * creating instance disks...
Thu Mar 21 14:16:08 2013 adding instance inst1.example.com to cluster config
Thu Mar 21 14:16:08 2013 * wiping instance disks...
Thu Mar 21 14:16:09 2013  - INFO: * Wiping disk 0
Thu Mar 21 14:16:20 2013  - INFO:  - done: 1.0% ETA: 18m 44s
Thu Mar 21 14:17:26 2013  - INFO:  - done: 7.0% ETA: 17m 4s
[...]
Thu Mar 21 14:34:18 2013  - INFO:  - done: 91.0% ETA: 1m 48s
Thu Mar 21 14:35:21 2013  - INFO:  - done: 96.0% ETA: 48s
Thu Mar 21 14:36:12 2013  - INFO: Waiting for instance inst1.example.com to sync disks.
Thu Mar 21 14:36:12 2013  - INFO: Instance inst1.example.com's disks are in sync.
```

Ganeti will use its allocation algorithm to find nodes that have room and create the instance. The default allocation algorithm can be overridden by specifying the exact nodes the instance should live on (primary:secondary) with `-n node1:node2`. Other instance parameters can be specified, such as configuring the virtual NIC to have a specific MAC address: e.g., `--net 0:mac=aa:00:00:fa:3a:3f`

Listing Instances

Here is an example that shows a list of instances:

```
# gnt-instance list
Instance            Hypervisor OS          Primary_node      Status      Memory
george.example.com  xen-pvm    ubuntu_std node3.example.com running     4.0G
inst1.example.com   xen-pvm    ubuntu_std node2.example.com running     512M
john.example.com    xen-pvm    ubuntu_std node2.example.com ADMIN_down 4.0G
paul.example.com    xen-pvm    ubuntu_std node4.example.com running     2.0G
ringo.example.com   xen-pvm    ubuntu_std node1.example.com running     2.0G
```

Listing Nodes

Here is an example of listing all the nodes. Note that node5 is offline for repairs and therefore not all information about it can be displayed. Before sending it to repairs, the instances were migrated to other nodes; thus, the Pinst/Sinst values (number of primary and secondary instances) are 0.

```
# gnt-node list
Node                  Dtotal   Dfree  Mtotal  Mnode Mfree  Pinst Sinst
node1.example.com     671.9G  83.1G  16.0G  1023M  5.8G       4     3
node2.example.com     671.9G  99.1G  16.0G  1023M  8.3G       4     3
node3.example.com     671.9G 212.9G  16.0G  1023M  6.8G       3     5
node4.example.com     671.9G 268.9G  16.0G  1023M  6.3G       4     4
node5.example.com          *      *      *      *     *       0     0
```

Adding a New Node

Adding a new node to the cluster is surprisingly easy. Once the software is installed and storage/network configurations are complete, the command to add the node only requires specifying the two things Ganeti cannot determine on its own: the nodes name and the IP address of its replication NIC.

```
# gnt-node add -s 192.168.20.2 node6.example.com
-- WARNING --
Performing this operation is going to replace the ssh daemon keypair on the target machine
   (node6.example.com) with the ones of the current one and grant full intra-cluster ssh root
   access to/from it

Sat Mar 16 14:53:04 2013  - INFO: Node will be a master candidate executed successfully
```

Verifying Consistency

In our final example, we run the Ganeti cluster command to check the system health and warn of any issues. In this case, we see a warning about an expired certificate used to authenticate RAPI requests. The command also checks for connectivity problems between the master and each node, storage problems, and much more:

```
# gnt-cluster verify
Submitted jobs 74499, 74500
Waiting for job 74499 ...
Thu Mar 21 14:33:23 2013 * Verifying cluster config
Thu Mar 21 14:33:23 2013 * Verifying cluster certificate files
Thu Mar 21 14:33:23 2013  - ERROR: cluster: While verifying
   /var/lib/ganeti/rapi.pem: Certificate is expired
   (valid from 2011-12-09 07:01:06 to 2012-12-09 07:11:06)
```

```
Thu Mar 21 14:33:23 2013 * Verifying hypervisor parameters
Thu Mar 21 14:33:23 2013 * Verifying all nodes belong to an existing group
Waiting for job 74500 ...
Thu Mar 21 14:33:23 2013 * Verifying group 'GROUP1'
Thu Mar 21 14:33:23 2013 * Gathering data (2 nodes)
Thu Mar 21 14:33:26 2013 * Gathering disk information (2 nodes)
Thu Mar 21 14:33:26 2013 * Verifying configuration file consistency
Thu Mar 21 14:33:26 2013 * Verifying node status
Thu Mar 21 14:33:26 2013 * Verifying instance status
Thu Mar 21 14:33:26 2013 * Verifying orphan volumes
Thu Mar 21 14:33:26 2013 * Verifying N+1 Memory redundancy
Thu Mar 21 14:33:26 2013 * Other Notes
Thu Mar 21 14:33:26 2013 * Hooks Results
```

The Ganeti commands are extensively documented with detailed man pages plus help summaries when the `--help` flag is given.

Running Ganeti in Production

Besides Ganeti itself, we recommend the use of other tools in order to have a scalable enterprise level environment:

- Self-installing nodes: These can be achieved from any automated installer, coupled with a good configuration management system.
- Monitoring: Various products can be configured to do black-box and white-box monitoring of Ganeti nodes, its storage devices, and its instances.
- Self-healing products: Ganeti can be coupled with Linux-HA or your monitoring system can be instrumented to perform cluster self-healing operations, and not require manual intervention on node downtime or other hardware errors. If this is coupled with white-box monitoring, nodes can be evacuated when they start to show problems but before they fail, thus avoiding any downtime.
- Administration interfaces: These allow users to self-service create/delete and modify their instances, and to access the instance console.

Ganeti Internals

The Ganeti platform is a collection of daemons and command line utilities written in Python and Haskell. The platform's main components are:

- The CLI scripts, which take user commands and transmit them to the master daemon via the LUXI protocol.
- The RAPI daemon, which accepts Ganeti operations over https and transmits them to the master daemon via the LUXI protocol.
- The Master daemon, which performs most Ganeti operations (opcodes) by executing logical units of code. Opcodes are units of Ganeti work, and do things such as starting instances, creating new ones, and so on. They can be serialized together in jobs, or submitted separately to allow the master daemon to run them concurrently in safety, without conflicting with each other. Jobs (consisting of one or more opcodes) are accepted via a JSON-on-UNIX-sockets protocol, and are sent mostly by the CLI utilities, the RAPI daemon, or our extra tools.
- The Node daemon runs on all nodes and performs the sub-units of actual work needed to implement the logical units. It performs minimal self-standing operations on each target node (e.g., creating a block device, writing a file to disk, executing a command), and it is controlled by the master daemon by an RPC system.
- htools are Ganeti's "cluster optimizations" suite. They include tools to rebalance the cluster's load (hbal), to allocate instances automatically in the best possible place (hail), to calculate how many more instances a cluster can accommodate (hspace), or to calculate how to best run maintenances on a cluster (hroller).

Other less crucial, and sometimes optional, Ganeti components are:

- The confd daemon, which has read-only access to the cluster config, and uses it to answer config queries.
- The monitoring agent daemon (mond, which provides real-time per-node information via http+json, and can be queried by a monitoring system to perform white-box monitoring on the cluster.

How to Reach the Community

Ganeti is discussed at the ganeti@googlegroups.com list. There you can ask questions, get help debugging issues, or just discuss your setup.

Development happens at ganeti-devel@googlegroups.com in the format of patches (which can be sent via git format-patch plus git send-email) and design docs. Contributing requires signing the Google Code Contributor License Agreement (CLA) by the copyright owner (either the individual or the company, depending), who retains copyright to his or her contributions.

The Ganeti project can be found at https://code.google.com/p/ganeti, the source code is at http://git.ganeti.org/, and documentation is built from the git tree and exported in real-time at http://docs.ganeti.org/.

Ganeti Roadmap

We have many ideas about Ganeti, which will be tackled as time and priority allow. In the near-to-medium future, we want to focus on:

1. better storage alternatives, and promoting disks from second-class citizens to first-class ones, which can be managed without being just part of a virtual machine;
2. dynamic networking, to make the datacenter networking architecture more efficient, scalable, and not dependent on preconfiguring; and
3. better integration with other clouds, harnessing the private/public cloud interconnection.

But the first idea we work on could be your idea. Just install Ganeti, and if you find something missing, let's discuss a design. We'll be happy to help you get up to speed and upstream your features, for the benefit of your installation and the entire community.

PRObE: A Thousand-Node Experimental Cluster for Computer Systems Research

GARTH GIBSON, GARY GRIDER, ANDREE JACOBSON, AND WYATT LLOYD

Garth Gibson is a Professor of Computer Science at Carnegie Mellon University and the co-founder and Chief Scientist at Panasas, Inc. He has an MS and PhD from the University of California at Berkeley and a BMath from the University of Waterloo in Canada. Garth's research is centered on scalable storage systems, and he has had his hands in the creation of the RAID taxonomy, the SCSI command set for object storage (OSD), the PanFS scale-out parallel file system, the IETF NFS v4.1 parallel NFS extensions, and the USENIX Conference on File and Storage Technologies.
garth@cs.cmu.edu

Gary Grider is the Acting Division Leader of the High Performance Computing (HPC) Division at Los Alamos National Laboratory. Gary is responsible for all aspects of high performance computing technologies and deployment at Los Alamos. Gary is also the US Department of Energy Exascale Storage, I/O, and Data Management National Co-coordinator. ggrider@lanl.gov

Andree Jacobson joined the New Mexico Consortium (NMC) in August 2010 as the Computer and Information Systems Manager and the project manager for the $10M NSF-sponsored PRObE project. Prior to NMC, he spent five years as a Computer Science Senior Lecturer at the University of New Mexico (UNM). During his time at UNM, he also spent his summers teaching the highly successful Computer Systems, Clusters, and Networking Summer Institute, now run as a part of the PRObE project.
andree@newmexicoconsortium.org

Wyatt Lloyd is a PhD candidate in Computer Science at Princeton University. His research interests include the distributed systems and networking problems that underlie the architecture of large-scale Web sites, cloud computing, and big data. He received his master's degree in Computer Science from Princeton University, and a bachelor's degree in Computer Science from Penn State University. wyatt.lloyd@gmail.com

If you have ever aspired to create a software system that can harness a thousand computers and perform some impressive feat, you know the dismal prospects of finding such a cluster ready and waiting for you to make magic with it. Today, however, if you are a systems researcher and your promised feat is impressive enough, there is such a resource available online: PRObE. This article is an introduction to and call for proposals for use of the PRObE facilities.

Server computing is increasingly done on clusters containing thousands of computers, each containing dozens of traditional cores, and the exascale supercomputers expected at the end of this decade are anticipated to have more than 100 thousand nodes and more than 100 million cores in total [1, 2]. Unfortunately, most academic researchers have only dozens of nodes with a handful of cores each. One of the best responses today is to rent a virtual datacenter from a cloud provider, such as Amazon or Google. We expect increasing numbers of papers to report experiments run on these virtual datacenters, but virtualization makes some experiments more difficult. Performance repeatability, network topology, and fault injection, for example, are not as well controlled on virtual datacenters as they

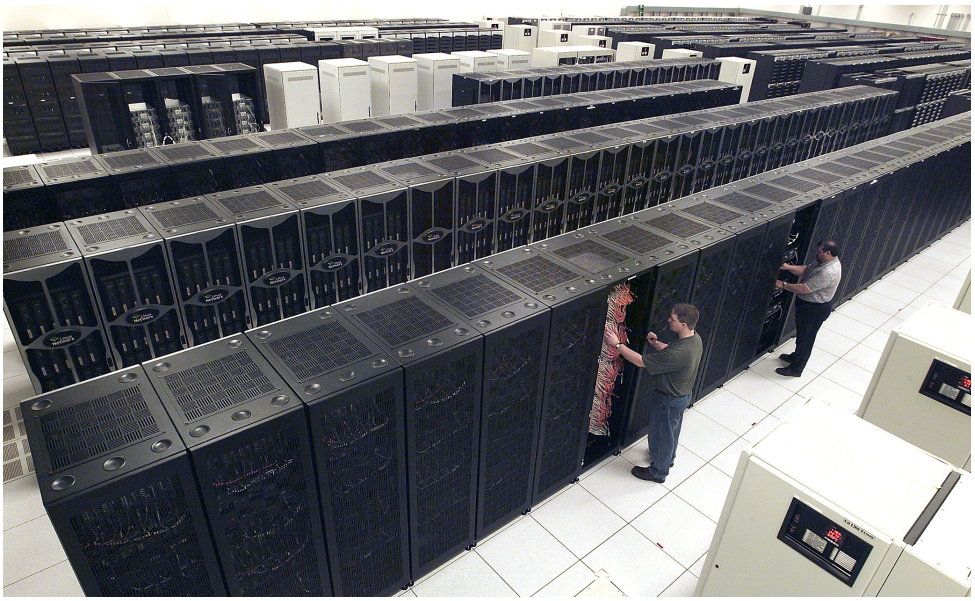


Figure 1: About one quarter of a Los Alamos National Laboratory supercomputer recently decommissioned and, probably, destroyed

are on physical datacenters. Moreover, debugging performance at scale is hard enough when all the hardware and software is known and controllable; learning from and perfecting innovative systems software in virtual datacenters is even harder. The systems research community needs access to larger-scale physical clusters, especially for the training of the next generation of computer systems scientists.

PRObE (Parallel Reconfigurable Observational Environment) is a systems research community resource for providing physical access to a thousand-node cluster. Made available by National Science Foundation operating support, equipment donations from the Los Alamos National Laboratory, and the facilities of the New Mexico Consortium, PRObE offers multiple clusters totaling more than 1,500 computers, with one cluster of more than 1,000 computers. The equipment in PRObE comes from computers at Los Alamos National Laboratory, such as shown in Figure 1, which have been decommissioned to make room for faster, more cost- and energy-efficient replacement computers. Researchers using PRObE have complete remote control of the hardware and software while running experiments and can inject both hardware and software failures as they see fit. Any operating system can be deployed on the systems using Emulab for physical cluster allocation [3].

PRObE is operational today. One of the first uses of PRObE's largest cluster was published in the 2013 Networked Systems Design and Implementation (NSDI '13) conference in a paper that validated the scalability of a geo-replicated storage system with causal consistency called Eiger [4]. Eiger's predecessor, called COPS, had been validated on only 16 nodes, whereas Eiger's use of PRObE allowed validation on up to 128 nodes (which, through replication, actually used 384 machines). Because a key contribution of Eiger is to scale to a large number of nodes per datacenter, while providing causal consistency and low latency with a rich data model and write-only transactions, having a large testbed was essential to the experiment. To quote the paper, "This experiment was run on PRObE's Kodiak testbed [results shown in Figure 2], which provides an Emulab with exclusive access to hundreds of machines. Each machine has 2 AMD Opteron 252 CPUS, 8GB RAM, and an InfiniBand high-speed interface." The Eiger paper is a fine example of the purpose of PRObE: enabling innovative systems to be tested at scale after they have been developed and explored on smaller private facilities.

To become a user of PRObE resources, follow these steps. First, all users of PRObE agree to publish, or otherwise make public, the results of PRObE use and give credit to the funders and providers of PRObE. Second, PRObE is an NSF-funded facility, so the organizations that request its use must be eligible to receive NSF funding. These constraints are explained in a user agreement on the PRObE Web site [5].

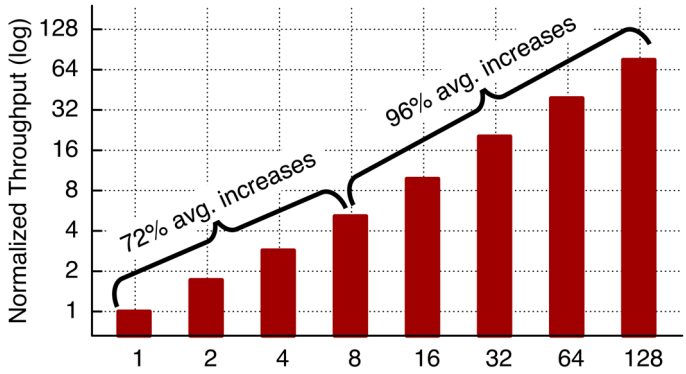


Figure 2: This figure shows the normalized throughput of multiple N-server clusters running the Facebook TAO workload [4]. Throughput approaches linear for up to 128 machines per cluster, using a total of 384 machines on PRObE's Kodiak cluster.

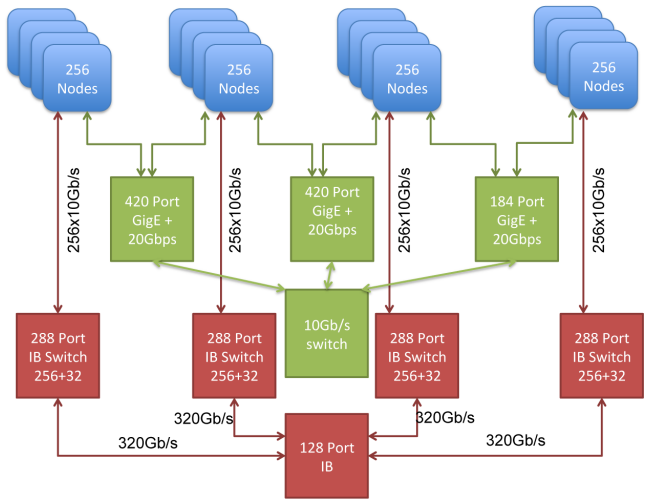


Figure 3: Block diagram of PRObE's Kodiak cluster

A new PRObE user is also an Emulab user. Emulab has been providing physical machine allocation and management in smaller clusters for more than a decade, and much of the systems research community already has experience with it. A new user logs in to a head node, launches a single node experiment with an existing base OS image, logs in to that node to customize the OS image as needed, instructs Emulab to record the customized image, then launches a multi-node allocation naming the customized image. Storage on the nodes is replaced with every launch but is fully available for experiments. Shared storage for images, inputs, and logging/monitoring results is available from Emulab head nodes and an NFS service.

PRObE's largest cluster, Kodiak, is intended to be allocated in its entirety to one project for days to weeks. New users should first log in to one of the smaller (~100 nodes) staging clusters, Denali or Marmot, to port their systems and demonstrate success on a small-scale experiment. Users then propose to use the

Machine	Nodes	Cores	Memory/node	Disk/node	Network/node
Marmot	128	256	16 GB	1 x 2 TB	GE, SDR Infiniband
Denali	64+	128+	8 GB	2 x 1 TB	GE, SDR Infiniband
Kodiak	1024	2048	8 GB	2 x 1 TB	GE, SDR Infiniband

Table 1: Currently available PRObE cluster capabilities

large cluster, with evidence of their readiness to be effective and an explanation of their project's goals and anticipated results. PRObE leadership and a community selection committee, when needed, will prioritize and arbitrate the use of the largest cluster.

The Parallel Reconfigurable Observational Environment (PRObE) is a collaboration between the National Science Foundation (NSF), under awards CNS-1042537 and CNS-1042543, New Mexico Consortium (NMC), Los Alamos National Laboratory (LANL), Carnegie Mellon University (CMU), and the University of Utah (Utah). PRObE facilities are available now and will be available for at least two years. For more information, visit the PRObE Web site at www.nmc-probe.org.

References

[1] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," DARPA IPTO AFRL FA8650-07-C-7724, 2008: citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.6676.

[2] John Shalf, "Exascale Computing Hardware Challenges," keynote talk at 5th Petascale Data Storage Workshop, New Orleans, LA, November 2010: slides available at http://www.pdsw.org/pdsw10/resources/slides/PDSW_SC2010_ExascaleChallenges-Shalf.pdf.

[3] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, Abhijeet Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," USENIX Symposium on Operating Systems Design and Implementation (OSDI '02), Dec. 2002: https://www.cs.utah.edu/flux/papers/netbed-osdi02.pdf.

[4] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, David G. Andersen, "Stronger Semantics for Low-Latency Geo-Replicated Storage," Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13), Lombard, IL, April 2013: https://www.usenix.org/conference/nsdi13.

[5] PRObE User Agreement version 1, Feb. 1, 2013: www.nmc-probe.org/policies/user-agreement.

Programming Models for Emerging Non-Volatile Memory Technologies

ANDY RUDOFF

Andy Rudoff is an Enterprise Storage Architect at Intel. He has more than 25 years of experience in operating systems internals, file systems, and networking. Andy is co-author of the popular *UNIX Network Programming* book. More recently, he has focused on programming models and algorithms for Non-Volatile Memory usage. andy.rudoff@intel.com

Upcoming advances in Non-Volatile Memory (NVM) technologies will blur the line between storage and memory, creating a disruptive change to the way software is written. Just as NAND (Flash) has led to the addition of new operations such as TRIM, next generation NVM will support load/store operations and require new APIs. In this article, I describe some of the issues related to NVM programming, how they are currently being resolved, and how you can learn more about the emerging interfaces.

The needs of these emerging technologies will outgrow the traditional UNIX storage software stack. Instead of basic read/write interfaces to block storage devices, NVM devices will offer more advanced operations to software components higher up in the stack. Instead of applications issuing reads and writes on files, converted into block I/O on SCSI devices, applications will turn to new programming models offering direct access to persistent memory (PM). The resulting programming models allow applications to leverage the benefits of technological advances in NVM.

The immediate success of these advances and next generation NVM technologies will rely on the availability of useful and familiar interfaces for application software as well as kernel components. Such interfaces are most successful when key operating system vendors and software vendors agree on an approach, terminology, and a strategy for widespread adoption. I will describe some of the more interesting changes on the horizon for NVM programming and outline new solutions to address these changes. Finally, I'll explain how the industry is driving commonality for these interfaces using a Technical Work Group (TWG) recently formed by the Storage Networking Industry Association (SNIA).

NVM Programming Models

Although there are surely countless possible programming models for using NVM, I'll focus on the four most relevant models. The first two represent the most common storage interfaces in use for many decades, which I will call NVM Block Mode and NVM File Mode. The remaining two models, which I will call PM Volume Mode and PM File Mode, specifically target the emergence of persistent memory.

NVM Block Mode

The diagram in Figure 1 represents a portion of a common software stack, where the dashed red line represents the interface providing the NVM Block Mode programming model.

There are many variations on the exact details of the software stack both above and below the dashed red line in Figure 1. The point of the diagram is to illustrate how the interface works, not to focus on a specific set of software components using it. As shown, the NVM Block Mode programming model provides the traditional block read/write interface to kernel modules such as file systems and, in some cases, to applications wanting to use the block device directly (for example, by opening /dev/sda1 on a Linux system).

Why is this decades-old interface interesting for a discussion of NVM Programming? Advances in NVM technology make it interesting by motivating change to an interface that

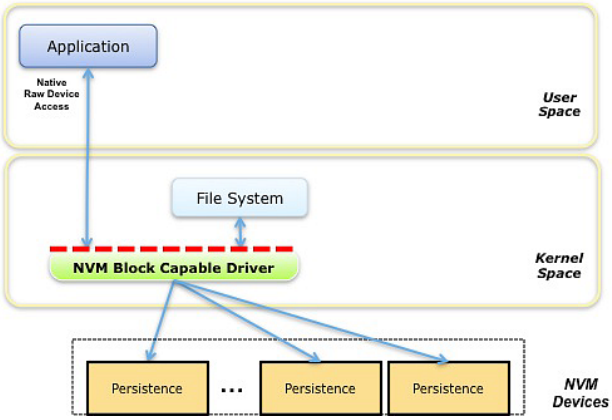


Figure 1: The NVM Block Mode interface, depicted by the red dashed line

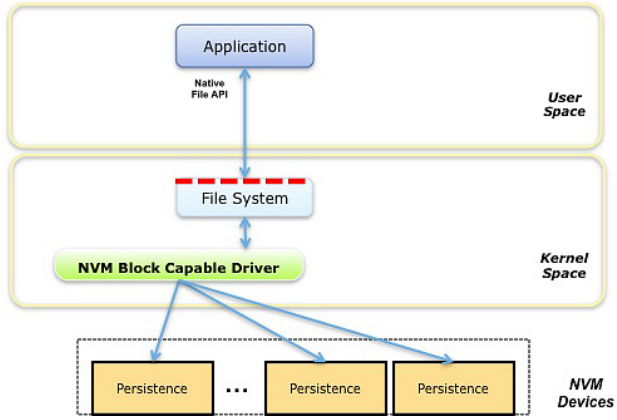


Figure 2: The NVM File Mode interface, providing the usual file operations enhanced with additional NVM operations

has otherwise barely changed in many years. A fairly recent but widely adopted example is the addition of software support for the TRIM command on the modern solid state drive (SSD). The TRIM command allows file systems to inform an SSD which blocks of data are no longer in use. Although useful for virtual arrays that support thin provisioning, this information was not necessary at the basic drive level until the emergence of Flash drives, where the wear-leveling management required by the drive can benefit from it [1].

Just as the simple block read/write interface to the driver required extensions to support TRIM, other emerging NVM features, such as support for atomic operations, will require similar broadening of the interfaces [2]. Additionally, simply exposing certain attributes of an NVM device to applications may prove just as useful. Applications can optimize I/O for performance using information on optimal I/O sizes, supported granularity of I/O, and attributes such as powerfail write atomicity. By arriving at common definitions for these extended operations and attributes, the NVM industry can provide a more effective ecosystem for software writers to develop NVM-aware applications that better leverage NVM features across multiple system types. Exactly how this ecosystem is created is covered later in this article.

NVM File Mode

Figure 2 illustrates the NVM File Mode programming model. Again, the red dashed line depicts the interface of interest, and the rest of the diagram is simply one possible layout of software components to show how the interface is used.

In this mode, a common file system such as ext4 on Linux uses a block-capable driver in the usual fashion. As with NVM Block Mode, this long-standing programming model will gain some incremental additions to the standard file API to allow applications to take advantage of advances in NVM.

For an example of how the NVM File Mode can evolve to benefit applications, consider the double write technique used by the MySQL database. This technique is used to protect database tables from corruption due to system interruption, such as a power failure. These tables are typically stored in files, and the double write technique is used to protect MySQL from partial page writes, that is, the write of a page of data that is torn by a system interruption. If the MySQL application were able to discover that writes of up to a certain size (the database page size) are guaranteed untearable by a system interruption, the double writes could be avoided [3]. Providing an interface for applications to request the powerfail write atomicity of the underlying NVM allows applications like MySQL to discover these attributes automatically and modify their behavior accordingly. Without this interface system, administrators must determine obscure attributes of the storage stack and edit MySQL configuration files to inform the application of these attributes.

PM Volume Mode

In Figure 3, the block diagram looks similar to NVM Block Mode, but here the device is not just NVM, but PM-Capable NVM. To be PM-capable means the NVM is usable directly via the processor load and store instructions. Although one might argue that any storage element might be connected to the system in a way the processor can load directly from it, the practicality of stalling a CPU while waiting for a load from technology such as NAND Flash prevents such a direct connection. But more advanced NVM technology, as well as innovative caching techniques, is allowing a generation of PM devices to emerge.

PM Volume Mode, as shown in the diagram, allows a kernel component to gain access to the persistence directly. The diagram shows a PM-Aware Kernel Module communicating with the NVM driver. This interface allows the kernel module to fetch the physical address ranges where the PM is accessed. Once the

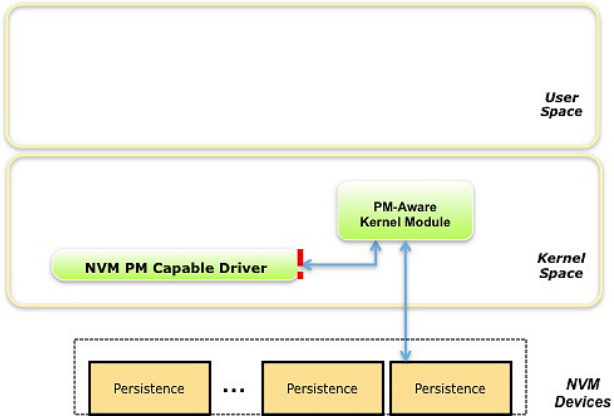


Figure 3: The PM Volume Mode interface, allowing a PM-Aware Kernel Module to look up the physical addresses in the volume

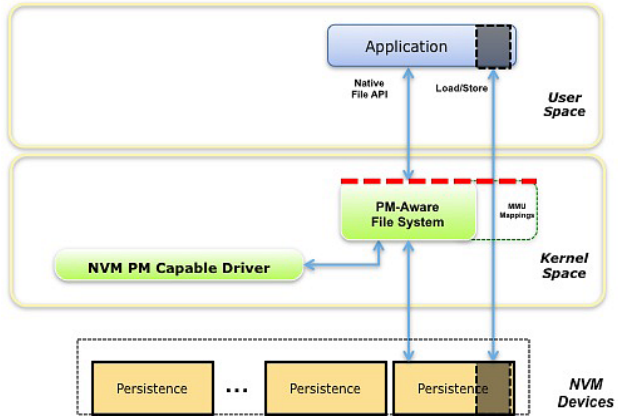


Figure 4: The PM File Mode interface, depicted by the red dashed line, providing the standard file operations but with memory mapped files going directly to NVM

kernel has that information, it need not ever call back into the NVM driver, instead accessing the PM directly as shown by the blue arrow in the diagram connecting the PM-Aware Kernel Module directly with the persistence in the NVM device. This fairly raw access to the PM allows the kernel module to add its own structure to the ranges of persistence and use it however it chooses. Examples include using the PM as a powerfail-safe RAID cache, a persistent swap space, or, as we'll discuss next, a PM- Aware File System.

A product providing PM may also provide NVM Block Mode, or any of the other modes; these programming models are not mutually exclusive, I am simply describing them separately because they are independent of each other.

PM File Mode

Our fourth NVM programming model is shown in Figure 4. PM File Mode is similar to the NVM File Mode we described earlier, but in this case the file system is specifically a PM-Aware File System.

Notice the interfaces associated with this programming model (the red dashed line again). The PM-Aware File System typically provides all the same file APIs as a traditional file system. In fact, a PM-Aware File System is likely created by enhancing an existing file system to be PM-aware. The key difference is in what happens when an application memory maps a file. As shown by the far right blue arrow in the diagram, memory mapping a file allows the application direct load/store access to the PM. Once the mapping is set up, accesses to the PM bypass any kernel code entirely since the MMU mappings allow the application physical access. This diverges from a traditional file system where memory mapped files are paged in and out of a page cache.

Why a file system? Why does this programming model center around the file APIs? This will be explained in the next section where I focus on persistent memory.

Persistent Memory

Now that all four NVM programming models have been described, I'll turn to the details of persistent memory. PM deserves special attention because, unlike the incremental improvements occurring with the NVM Block and NVM File modes, PM offers a much more disruptive change. Just as the ecosystem reacted to the change from faster clock rates on single CPUs to higher core counts (forcing performance-sensitive applications to revise their algorithms to be multithreaded), PM will cause the ecosystem to rethink how data structures are stored persistently. PM offers a combination of persistence and the ability to access the data structures without first performing block I/O and then converting the blocks of data into memory-based structures. As with any new technology, the benefits of PM come with a set of new and interesting challenges.

Allocation of Persistent Memory

Every C programmer is familiar with the standard malloc interface for dynamically allocating memory:

```
ptr = malloc(len);
```

Given a length in bytes, an area of RAM is returned to the calling process. This well-worn interface is simple and easy to use, although one could argue it is also easy to misuse, causing hours of debugging memory leak and memory corruption issues. But with so many decades of use and millions of lines of C code depending on malloc, a natural way to expose PM seems to be simply adding another version of malloc:

```
ptr = pm_malloc(len);  /* the naïve solution */
```

This simple solution gives the application programmer a choice between allocating RAM (using malloc) and PM (using pm_malloc), which seems like a fine solution on the surface but quickly falls short on further examination. Presumably the application was allocating PM in order to store something in it persistently, since that's the whole point. So the application will need a way to get back to that range of PM each time it is run, as well as each time the system is rebooted or power cycled. To allow this, the PM must be given a name that the application can provide to reconnect with it.

Many naming schemes for PM are possible, from some sort of numeric object ID to URL-like strings. But once the PM is named, the next issue immediately comes up: How to determine if an application has permission to connect to an area of PM? Like naming, many permission schemes are possible, but as you delve into the management of PM, you start to find even more issues, such as how does the system administrator change the permissions on PM? How are old areas of PM removed or renamed? Even more importantly, how are areas of PM backed up to protect against hardware failure? For traditional storage, the file system semantics provide answers to all these questions, so even though PM is much more like system memory, exposing it like files provides a convenient solution. The file API provides a natural namespace for PM ranges—ways to create, delete, resize, rename the ranges—and many off-the-shelf backup tools will simply work. The net effect of this approach is that if an application wants volatile memory, it calls malloc, and if it wants PM, it opens (or creates) a file on a PM-Aware File System and uses the mmap API to map it into its address space.

Making Changes to Persistent Memory Durable

With volatile memory, there's no need to worry about the durability of stores because all memory-resident information is assumed lost when the application or system shuts down. But with storage, that data stored is often cached and must be committed to durable media using some sort of synchronization API. For memory mapped files, that API is msync [4]. Although a strict interpretation of the traditional msync call is that it flushes pages of data from a page cache, with PM the application has direct load/store access without involving the page cache. The msync call for PM is instead tasked with flushing the processor caches, or any other intermediate steps required to make sure the changes are committed to the point of being powerfail safe.

Position-Independent Data Structures

With PM available to applications, for those applications to store data structures such as arrays, trees, heaps, etc. is convenient. On start-up, the application can use the file APIs described above to memory map PM and immediately access those data

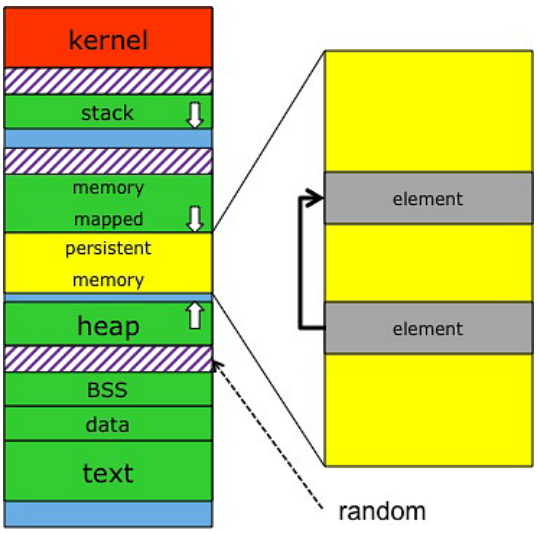


Figure 5: Typical process address space layout, with slightly different positions each run due to the randomly sized areas

structures; however, there's an issue around position-independence of the data structures as shown in Figure 5.

On the left side of the diagram, the typical address space layout of a process on a UNIX system is shown. Because PM is accessed as memory mapped files, it gets mapped into the area with the other memory mapped files, such as shared libraries (growing downwards). The striped areas on many systems are the spaces between ranges, such as stack and memory mapped files, and the exact sizes of the striped areas are often random. This is a security feature designed to make some types of attacks more difficult [5]. For data structures stored in PM, the result is that any pointers, like the one depicted on the right side of the diagram, will be invalid between any two runs of the application. Of course, this isn't a new problem; storing absolute pointers in a memory mapped file has always been problematic, but the emergence of PM means this is expected to be a much more common problem to solve.

The obvious solution, to only store relative pointers in PM, can be somewhat error prone. Every pointer dereference must account for the fact that the pointer is relative and add in some sort of base offset. Higher-level languages with runtime virtual machines, such as Java, or languages without explicit pointers, may be able to handle this transparently, which is an area of research, but the first goal is to expose PM in the basic low-level system call and C environment. One potential answer is the idea of based pointers, a feature available in some C compilers, such as Microsoft's C++ compiler [6]. With this feature, a new keyword, `__based`, is added to the language so that declarations such as this linked list example are possible:

```
void *myPM;

struct element {
      …
       struct element __based (myPM) *next;
}
```

The result is that when the PM file is memory mapped, the location of the PM area is stored in the pointer `myPM`, and due to the `__based` declaration, every time the next field is dereferenced, the compiler generates code to adjust it by the value of `myPM`, creating a convenient position-independent pointer for the programmer.

So far I've described only one of the many issues around position-independent data structures and the storing of data structures in PM. Fortunately, there is quite a bit of research going on in academia on this topic, and two bodies of work demand special mention here. The NV-Heaps work [7] and the Mnemosyne project [8] both attack the issue described here in different and innovative ways. These works also look into language extensions and other runtime solutions to these problems and are recommended reading for anyone interested in PM.

Error Handling

The main memory of a computer system is typically protected against errors by mechanisms such as error correcting codes (ECC). When that memory is used by applications, such as memory allocated by calling malloc, applications do not typically deal with the error handling. Correctable errors are silently corrected—silently as far as the application is concerned (the errors are often logged for the administrator). Uncorrectable errors in which application memory contents are corrupted may be fixed by the OS if possible (for example, by re-reading the data from disk if the memory contents were not modified), but ultimately there are always cases in which the program state is known to be corrupted and it is not safe to allow the program to continue to run. On most UNIX systems, the affected applications are killed in such cases, the UNIX signal SIGBUS most often being used.

Error handling for PM starts off looking like memory error handling. Using Linux running on the Intel architecture as an example, memory errors are reported using Intel's Machine Check Architecture (MCA) [9]. When the OS enables this feature, the error flow on an uncorrectable error is shown by the solid red arrow in Figure 6, which depicts the mcheck module getting notified when the bad location in PM is accessed.

As mentioned above, sending the application a SIGBUS allows the application to decide what to do; however, in this case, remember that the PM-Aware File System manages the PM and that the location being accessed is part of a file on that file system. So even if the application gets a signal preventing it from

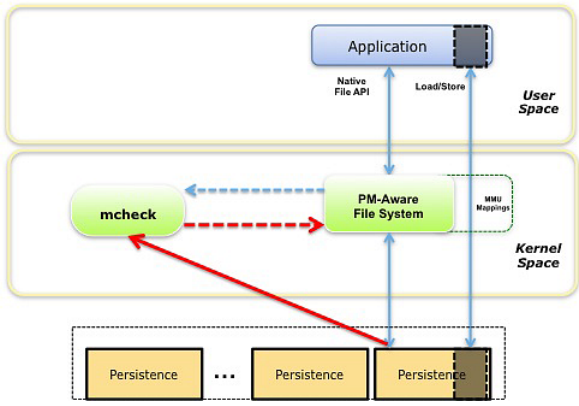


Figure 6: The machine check error flow in Linux with some proposed new interfaces depicted by dashed arrows

using corrupted data, a method for recovering from this situation must be provided. A system administrator may try to back up the rest of the data in the file system before replacing the faulty PM, but with the error mechanism we've described so far, the backup application would be sent a SIGBUS every time it touched the bad location. In this case, the PM-Aware File System needs a way to be notified of the error so that it can isolate the affected PM locations and then continue to provide access to the rest of the PM file system. The dashed arrows in Figure 6 show the necessary modification to the machine check code in Linux. On start-up, the PM-Aware File System registers with the machine code to show it has responsibility for certain ranges of PM. Later, when the error occurs, the PM-Aware File System gets called back by the mcheck module and has a chance to handle the error.

Here I've provided an abbreviated version of the error-handling story for PM. This is still a developing area and I expect the error-handling primitives to continue to evolve.

Creating an Ecosystem

The rapid success of PM and other emerging NVM technologies depends on creating an effective ecosystem around new capabilities as they become available. If each operating system vendor and hardware vendor creates its own divergent API for using these features, the ability of software vendors, kernel programmers, and researchers to exploit these features becomes limited. To avoid this, a group of industry leaders has worked with SNIA to create the NVM Programming Technical Work Group. Here is how the TWG describes itself:

The NVM Programming TWG was created for the purpose of accelerating availability of software enabling NVM (Non-Volatile Memory) hardware. The TWG creates specifications, which provide guidance to operating system, device driver, and application developers. These specifications are vendor agnostic and support all the NVM technologies of member companies. [10]

The TWG is currently working on a specification describing the four NVM programming models I covered in this article. The specification will cover the common terminology and concepts of NVM, including PM, and it will describe the semantics of the new actions and attributes exposed by emerging NVM technology. But the TWG intentionally stops short of defining the APIs themselves. This approach of providing the semantics but not the syntax is done to allow the operating systems vendors to produce APIs that make the most sense for their environments. The TWG membership includes several operating system vendors that are actively participating in the definition of the programming models. In fact, in the few months that the TWG has existed, a remarkable number of companies have joined. As of this writing, the membership list is: Calypso Systems, Cisco, Dell, EMC, FalconStor, Fujitsu, Fusion-io, Hewlett-Packard, Hitachi, Huawei, IBM, Inphi, Integrated Device Technology, Intel, LSI, Marvell, Micron, Microsoft, NetApp, Oracle, PMC-Sierra, QLogic, Samsung, SanDisk, Seagate, Symantec, Tata Consultancy Services, Toshiba, Virident, and VMware. (This list illustrates the scale of the collaboration and will surely be out-of-date by the time this article is published.)

Summary

A software engineer will see countless incremental improvements in hardware performance, storage capacity, etc. through a long career. That same career will witness high impact, game-changing developments only a few times. The transition of NVM from something that looks like storage into something that looks more like memory is one such disruptive event. By pulling the industry together to define common ground, we can enable software to rapidly and fully exploit these new technologies. The SNIA NVM Programming Technical Work Group is our effort to make this happen, and it has gained considerable industry traction in just a few months.

References

[1] Intel, Intel High Performance Solid-State Drive—Advantages of TRIM: http://www.intel.com/support/ssdc/hpssd/sb/CS-031846.htm.

[2] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, D.K. Panda, "Beyond Block I/O: Rethinking Traditional Storage Primitives," 17th IEEE International Symposium on High Performance Computer Architecture (HPCA-17), February 2011, San Antonio, Texas.

[3] Peter Zaitsev, "Innodb Double Write," MySQL Performance Blog (Percona): http://www.mysqlperformanceblog.com/2006/08/04/innodb-double-write/.

[4] The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 edition—msync: http://pubs.opengroup.org/onlinepubs/009695399/functions/msync.html.

[5] Shacham et al., "On the Effectiveness of Address-Space Randomization," Proceedings of the 11th ACM Conference on Computer and Communications Security, 2004, pp. 298-307.

[6] Microsoft Developer Network, Based Pointers (C++): http://msdn.microsoft.com/en-us/library/57a97k4e(v=vs.80).aspx.

[7] J. Coburn et al., "NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories," The 16th ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11), March 2011, Newport Beach, California.

[8] Haris Volos, Andres Jaan Tack, Michael M. Swift, "Mnemosyne: Lightweight Persistent Memory," The 16th ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11), March 2011, Newport Beach, California.

[9] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, Chapter 15, March 2013: http://download.intel.com/products/processor/manual/325462.pdf.

[10] Storage Networking Industry Association, Technical Work Groups: http://www.snia.org/tech_activities/work/twgs.

Practical Perl Tools
Constant as the Northern $*

DAVID N. BLANK-EDELMAN

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010. dnb@ccs.neu.edu

The bad Perl wordplay in the title can mean only one thing: We have another column about programming in the Perl language underway. This issue's column is inspired and derived from an article I stumbled on by Neil Bowers. Back in 2012 he wrote an article on 21 different modules for defining constants in Perl. The original is at http://neilb.org/reviews/constants.html. If reading my take on his research gets you interested in the subject, do be sure to seek out his article. One other thing I should mention before we discuss his work: He's clearly cooler than I will ever be. When he encountered some bugs in one of the modules he reviewed, he "took over maintenance of the module and released a new version which addresses all known issues." Now that's thorough!

For this article I'm not going to discuss all of the 21 modules he reviewed. Rather, I thought it would be good to talk about why modules like this are not only a best practice sort of thing but downright handy, and then dive into some of the more accessible/interesting modules from Bowers' list.

What and Why

This may be fairly basic programming language terminology, but to make sure we're on the same page let me share one pragmatic view of what constants are and why you want to use them. Constants come into play when you want to write code that uses variables that don't change for the life of the program.

I realize that sounds a little strange—after all, why use a variable if it isn't going to change? Why not just use a value? It all comes down to code readability and maintainability. Let's say you are writing code that logs information using syslog and you want to specify which priority to log at. If we look at the C include file on the machine I'm typing on, we can see the following is defined:

```
#define LOG_EMERG      0    /* system is unusable */
#define LOG_ALERT      1    /* action must be taken immediately */
#define LOG_CRIT       2    /* critical conditions */
#define LOG_ERR        3    /* error conditions */
#define LOG_WARNING    4    /* warning conditions */
#define LOG_NOTICE     5    /* normal but significant condition */
#define LOG_INFO       6    /* informational */
#define LOG_DEBUG      7    /* debug-level messages */
```

This means I could write code that looks like:

```
log( 'It is getting kind of hot in here', 4 );
```

or

```
if ( $log_level == 4 ) { do_something_with_the_warning };
```

but unless I knew about /usr/include/sys/syslog.h, I'd still be shaking my head when I came back to this code in a year. You can just imagine the internal dialogue that would start with "'4', what the heck does '4' mean?"

A better version of those lines of code might be:

```
our $LOG_EMERG   = 0;  # system is unusable
our $LOG_ALERT   = 1;  # action must be taken immediately
our $LOG_CRIT    = 2;  # critical conditions
our $LOG_ERR     = 3;  # error conditions
our $LOG_WARNING = 4;  # warning conditions
our $LOG_NOTICE  = 5;  # normal but significant condition
our $LOG_INFO    = 6;  # informational
our $LOG_DEBUG   = 7;  # debug-level messages
```

which lets you then write lines that are considerably easier to read, like:

```
log( 'It is getting kind of hot in here', $LOG_EMERG );
                or
if ( $log_level == $LOG_EMERG)  { do_something_with_the_warning };
```

So this is all well and good until a little while later when your code base gets passed to a colleague who isn't as familiar with it and she's asked to make some "minor changes." While making these changes, she notices the use of $LOG_INFO sprinkled throughout the code and thinks, "Great, that's where I should store my log messages before they get sent out." She adds this to the code:

```
$LOG_INFO = "Everything is peachy."; # set the log message
```

and lo and behold things start failing in a weird way (immediately if you are lucky, months later when no one remembers that changes were made if you are not). Here's a case in which you really want to use variables, but you want them to be immutable. Once you set a variable like this, you want it to stay at that value and scream bloody murder (or at least deny the request) if there are any attempts to change it from that point on.

There's no special variable type (as in scalar, list, hash) built into the Perl language to make this happen, so that's where the modules we'll be discussing come into play.

Behind the Scenes

Before we actually see any of these modules, I think it is useful to have in the back of your head a rough idea of how they work. As Bowers points out in his article, there are essentially two different ways to cast this particular spell.

First, you can use a mechanism already built into the language to associate some code with a variable. This code denies attempts to do anything but retrieve the value. Associating code with a variable is exactly what the tie() function does. There have been a number of columns here in which I've talked about the white and black magic associated with tie() so check out the archives if this notion intrigues you.

The other way some modules make variables read-only is to reach into the guts of the Perl core and use what is essentially an undocumented but well-known function called Internals::SvREADONLY. As Bowers notes in his article, the source code for the Perl interpreter has this to say where the function is defined:

```
XS(XS_Internals_SvREADONLY)    /* This is dangerous stuff. */
{
   dVAR;
   dXSARGS;
   ...
```

I realize this is a little scary. The conclusion I've come to after looking into this is SvREADONLY is well known enough and has been used in enough modules that I don't think I would be concerned about actually making use of it (indirectly via a module).

There are definitely pluses and minuses to each technique. Bowers does an excellent job of summarizing them toward the end of his article, so rather than rehashing them there, I'd recommend you look at his Comparison section.

Let's Do It

Okay, let's actually look at a number of the more straightforward modules out there. The first that should get mentioned is the one that has shipped with Perl since Perl 5.004 and is actually a pragma (a pre-processor directive). The constant pragma (to quote the doc) "allows you to declare constants at compile-time." What this means is the constant gets created before the actual program begins running (i.e., during the compilation phase when Perl is reading in the program and deciding how to execute it). I'll show you why that detail matters in just a second.

To use the pragma, you can write code like:

```
# define a number of constants in one fell swoop
 use constant {
  LOG_EMERG           => 0,
  LOG_ALERT           => 1,
  LOG_CRIT            => 2,
  LOG_ERR             => 3,
  LOG_WARNING         => 4,
  LOG_NOTICE          => 5,
  LOG_INFO            => 6,
  LOG_DEBUG           => 7,
 }
# ... or we could do this one at a time like this:
#   use constant LOG_EMERG => 0;
#   use constant LOG_ALERT => 1; ... etc.
```

```
# now let's use it
log('Here is a notice', LOG_NOTICE);
```

To prove the immutability of what we've defined, if we wrote

```
LOG_NOTICE = "some other value";
```

it would immediately fail with an error message like

```
Can't modify constant item in scalar assignment
```

Before we look at another module, let me explain the importance of the compile-time detail. To use an example modified from the docs, if I were to create a constant like

```
use constant DEBUG => 0;
```

and use it in code like

```
if (DEBUG) {
  # lots of debugging related code
    # yes, lots of it
    # ...
}

}
```

Perl will be smart enough to optimize that entire chunk of code out of the program before it runs because the value of DEBUG is false.

The second module I'd like to show you has actually made an appearance in this column before because it is the one recommended in Damian Conway's most excellent book *Perl Best Practices.* Conway recommends using the Readonly module because it allows you to do things like variable interpolation.

Quick aside: when you install Readonly, you may also want to install Readonly::XS. Readonly::XS is never called directly, but it lets Readonly use the Internals::SvREADONLY method for scalar constants (thus making it much faster than its usual use of tie()). Note: if you do want to use Readonly::XS, there is a long outstanding bug in Readonly that requires you to use Readonly::Scalar explicitly.

Here's the way Readonly gets used:

```
use Readonly;
Readonly my $LOG_EMERG          => 0;
Readonly my $LOG_ALERT          => 1;
Readonly my $LOG_CRIT           => 2;
Readonly my $LOG_ERR            => 3;
Readonly my $LOG_WARNING        => 4;
Readonly my $LOG_NOTICE         => 5;
Readonly my $LOG_INFO           => 6;
Readonly my $LOG_DEBUG          => 7;
```

Then we do the usual:

```
# note it is $LOG_NOTICE, not LOG_NOTICE
log( 'Here is a notice', $LOG_NOTICE );
```

One difference between the constant pragma and Readonly is with Readonly we could write this:

```
print "The value for the current log level is $LOG_NOTICE\n";
```

because string interpolation works. Readonly can also be used to make entire lists and hashes read-only if desired (though it does so using the slower tie() interface).

Although Readonly appears to be the most popular module of its ilk, possibly because of the Conway stamp of approval, it really hasn't seen much love in a while. The latest version on CPAN as of this writing is from April 2004 (though Readonly::XS did see a release in February of 2009). In his article, Bowers gives the nod to Const::Fast as one potentially worthy successor to Readonly. The doc for Const::Fast does indeed say it was written to work around some of Readonly's issues and actually says, "The implementation is inspired by doing everything the opposite way Readonly does it."

Like Readonly, it also lets you create read-only scalars, arrays, and hashes using Readonly-esque syntax as the example code in the doc demonstrates:

```
use Const::Fast;

const my $foo    => 'a scalar value';
const my @bar    => qw/a list value/;
const my %buz    => (a => 'hash', of => 'something');
```

Off the Beaten Path

Up until now we've looked at modules that have the same basic form and function. Before we end our time together in this column, I thought it might be interesting to look at a few modules that take this basic concept and extend it in some way.

The first module in this category is Config::Constants. Config::Constants encourages a potentially good development practice where the configuration for your program is (1) represented as constants and (2) stored in a separate file from the rest of the code. That separate file is in either XML or Perl data structure format. An example XML config file might look like this:

```
<config>
  <module name "MyConstants">
    <constant name='LOG_EMERG'          value='0' />
    <constant name='LOG_ALERT'          value='1' />
    <constant name='LOG_CRIT'           value='2' />
    <constant name='LOG_ERR'            value='3' />
    <constant name='LOG_WARNING'        value='4' />
    <constant name='LOG_NOTICE'         value='5' />
    <constant name='LOG_INFO'           value='6' />
    <constant name='LOG_DEBUG'          value='7' />
```

```
  </module>
</config>
```

with the equivalent Perl data structure version looking like this:

```
{
  'MyConstants' => {
          LOG_EMERG    => 0,
          LOG_ALERT    => 1,
          LOG_CRIT     => 2,
          LOG_ERR      => 3,
          LOG_WARNING  => 4,
          LOG_NOTICE   => 5,
          LOG_INFO     => 6,
          LOG_DEBUG    => 7,
           }
}
```

We'd typically create a module responsible for exporting constants to the rest of our program, as in:

```
package MyConstants;
use Config::Constants qw/LOG_EMERG LOG_ALERT LOG_CRIT
                         LOG_ERR LOG_WARNING LOG_NOTICE
                         LOG_INFO LOG_DEBUG/;
# define some functions that use these constants

    sub emerg_log {
      $message = shift;
          log($message, LOG_EMERG);
     }

 1;
```

To use this module, our main program would look like this:

```
use Config::Constants xml        => 'config.xml';
                # or perl        => 'config.pl';
use MyConstants;

emerg_log('Houston, we have a problem');
```

Another module that also deals with the question of where the constants are defined is Constant::FromGlobal. With Constant::FromGlobal you'd write something like this:

```
use Constant::FromGlobal LOG_LEVEL =>
                                { env     => 1,
                                  default => 0, };
```

and it will attempt to create a constant called LOG_LEVEL and set it to a value retrieved from a hierarchy:

- First it will see if there is a global variable $LOG_LEVEL set in the package, but if there is no global variable set...
- it will look for an environment variable called $MAIN_LOG_LEVEL, but if there is no environment variable set...
- it is given the default value (0).

In case you are curious about the name of the environment variable given in the second step above, Constant::FromGlobal wants the name of the current namespace prepended to variable name. By default, everything runs in the "main" namespace, although if we were using this in a module definition, we might write:

```
package Usenix;
use Constant::FromGlobal LOG_LEVEL =>
                                  { env     => 1,
                                    default => 0, };
```

and instead the module would look for an environment variable of USENIX_LOG_LEVEL instead.

Okay, last module of the day and then we can all go home. Constant::Generate module sets itself apart by being able to create values for you on the fly. Let's say you didn't care what the values were for constants, just that they had some. Constant::Generate lets you write:

```
use Constant::Generate [ qw( LOG_EMERG LOG_ALERT LOG_CRIT
                             LOG_ERR LOG_WARNING LOG_NOTICE
                             LOG_INFO LOG_DEBUG ) ];
```

and the constants get integer values starting at 0 (which coincidentally are the same values we've been setting by hand previously). For a slightly cooler self-assignment, we could instead say:

```
use Constant::Generate  [ qw( EMERG ALERT CRIT
                              ERR WARNING NOTICE
                              INFO DEBUG/)],
                                  prefix  => 'LOG_',
                                  dualvar => 1;
```

and not only do we get the LOG_something constants from before, but they act differently depending on the context they are used in, for example:

```
my $log_level = LOG_DEBUG;

print "Current log level: $log_level\n";
print "Yes, debug\n" if $log_level eq 'LOG_DEBUG';
print "Definitely debug\n" if $log_level == LOG_DEBUG;
```

In the first two print lines, the LOG_DEBUG constant is used in a string context. The constant appears to represent a string value that is identical to its name; however, in the third print statement we're making a numeric comparison and that still works fine. And with that little bit of magic, we'll stop here. Take care and I'll see you next time.

A PyCon Notebook

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009) and *Python Cookbook* (3rd Edition, O'Reilly & Associates, 2013). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply/index.html). Beazley is based in Chicago, where he also teaches a variety of Python courses. dave@dabeaz.com

As I begin to write this, I'm returning on the plane from PyCon 2013, held March 13-17 in Santa Clara, California. When I started using Python some 16 years ago, the Python conference was an intimate affair involving around a hundred people. This year's conference featured more than 2,500 attendees and 167 sponsors—bigger than ever for an event that's still organized by the community (full disclaimer, I was also one of the sponsors). If you couldn't attend, video and slidedecks for virtually every talk and tutorial can be found online at http://pyvideo.org and https://speakerdeck.com/pyconslides.

There are any number of notable things I could discuss about the conference, such as the fact that everyone received a Raspberry Pi computer, there were programming labs for kids, or the record-setting conference attendance by women; however, in this article I'm primarily going to focus on the one project that seems to be taking over the Python universe—namely, the IPython Notebook project.

If you attend any Python conference these days, you'll quickly notice the widespread use of the IPython Notebook (http://ipython.org) for teaching, demonstrations, and day-to-day programming. What is the notebook and why are so many people using it, you ask? Let's dive in.

The IPython Shell

Before getting to the notebook, knowing about the more general IPython project that has evolved over the past ten years will help. In a nutshell, IPython is an alternative interactive shell for Python that provides a broad range of enhancements, such as better help features, tab completion of methods and file names, the ability to perform shell commands easily, better command history support, and more. Originally developed to support scientists and engineers, IPython is intended to provide a useful environment for exploring data and performing experiments. Think of it as a combination of the UNIX shell and interactive Python interpreter on steroids.

To provide a small taste of what IPython looks like, here is a sample session that mixes Python and shell commands together to determine how much disk space is used by different types of files in the current working directory:

```
bash-3.2$ ipython
Python 2.7.3 (default, Dec 10 2012, 06:24:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.1 -- An enhanced Interactive Python.
?       -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help     -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: cd ~
/Users/beazley
```

```
In [2]: ls
Desktop/     Junk/      Music/     Public/     Tools/
Documents/   Library/   Pictures/  Sites/
Downloads/   Movies/    Projects/  Teaching/

In [3]: cd Pictures
/Users/beazley/Pictures

In [4]: import collections

In [5]: import os

In [6]: size_by_type = collections.Counter()

In [7]: for path, dirs, files in os.walk('.'):
   ...:     for filename in files:
   ...:         fullname = os.path.join(path, filename)
   ...:         if os.path.exists(fullname):
   ...:             _, ext = os.path.splitext(filename)
   ...:             sz = os.path.getsize(fullname)
   ...:             size_by_type[ext.upper()] += sz
   ...:

In [8]: for ext, sz in size_by_type.most_common(5):
   ...:     print ext, sz
   ...:
.JPG 50389086278
.MOV 38328837384
.AVI 9740373284
.APDB 733642752
.DATA 518045719

In [9]:
```

As you can see, a mix of UNIX shell commands and Python statements appear. The "In [*n*]:" prompt is the interpreter prompt at which you type commands. This prompt serves an important purpose in maintaining a history of your work. For example, if you wanted to redo a previous sequence of commands, you could use rerun to specify a range of previous operations like this:

```
In [9]: cd ../Music
/Users/beazley/Music

In [10]: rerun 6-8
=== Executing: ===
size_by_type = collections.Counter()
for path, dirs, files in os.walk('.'):
    for filename in files:
        fullname = os.path.join(path, filename)
        if os.path.exists(fullname):
            _, ext = os.path.splitext(filename)
            sz = os.path.getsize(fullname)
            size_by_type[ext.upper()] += sz

for ext, sz in size_by_type.most_common(5):
    print ext, sz

=== Output: ===
.M4A 9704243754
.MP3 2849783536
.M4P 2841844039
.M4V 744062510
.MP4 573729448

In [11]:
```

Or, if you wanted to save your commands to a file for later editing, you could use the save command like this:

```
In [11]: cd ~
/Users/beazley


In [12]: save usage.py 4-8
The following commands were written to file `usage.py`:
import collections
import os
size_by_type = collections.Counter()
for path, dirs, files in os.walk('.'):
    for filename in files:
        fullname = os.path.join(path, filename)
        if os.path.exists(fullname):
            _, ext = os.path.splitext(filename)
            sz = os.path.getsize(fullname)
            size_by_type[ext.upper()] += sz

for ext, sz in size_by_type.most_common(5):
    print ext, sz

In [13]:
```

Should you be inclined to carry out more sophisticated shell operations, you can usually execute arbitrary commands by prefixing them with the exclamation point and refer to Python variables using $ variable substitutions. For example:

```
In [13]: pid = os.getpid()

In [14]: !lsof -p$pid
COMMAND PID   USER    FD   TYPE DEVICE  SIZE/OFF       NODE NAME
Python  8686 beazley cwd  DIR 14,2      238 2805734    /Users/...
Python  8686 beazley txt  REG 14,2      12396 2514070  /Library/...
...

In [15]:
```

You can capture the output of a shell command by simply assigning it to a variable:

```
In [15]: out = !lsof -p$pid -F n

In [16]: out
Out[16]:
['p8686',
 'n/Users/beazley/Desktop/UsenixLogin/beazley_jun_13',
```

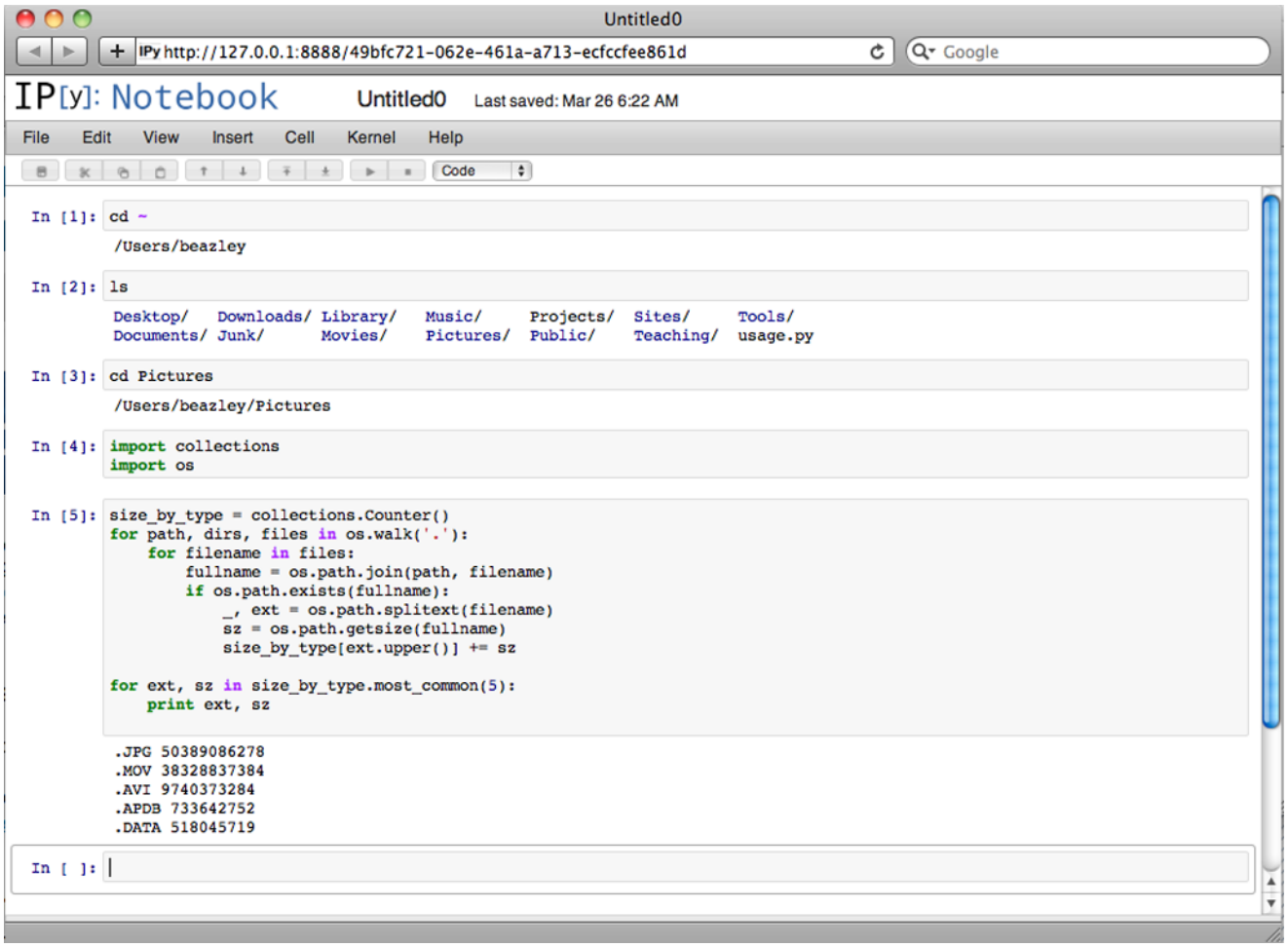


Figure 1: Notebook works with IPython, and at first appears not that different from using IPython alone

```
 'n/Library/Frameworks/Python.framework/Versions/7.3/Python',
 ...
]

In [17]:
```

This session gives you a small glimpse of what IPython is about and why you might use it; however, this is not the IPython Notebook.

From the Shell to the Notebook

Imagine, if you will, the idea of taking the above shell session and turning it into a kind of interactive document featuring executable code cells, documentation, inline plots, and arbitrary Web content (images, maps, videos, etc.). Think of the document as the kind of content you might see written down in a scientist's lab notebook. Well, that is basically the idea of the IPython Notebook project. Conveying the spectacle it provides in print is a little hard, so a good place to start might be some of the videos at http://pyvideo.org.

To get started with the IPython notebook yourself, you'll need to spend a fair bit of time fiddling with your Python installation. There are a number of required dependencies, including pyzmq (https://pypi.python.org/pypi/pyzmq/) and Tornado (https://pypi.python.org/pypi/tornado). Additionally, to realize all of the IPython notebook benefits, you'll need to install a variety of scientific packages, including NumPy (http://numpy.org) and matplotlib (http://matplotlib.org). Frankly, working with a Python distribution in which it's already included, such as EPDFree (http://www.enthought.com/products/epd_free.php) or Anaconda CE (http://continuum.io/anacondace.html), is probably easier. If you're on Linux, you might be able to install the required packages using the system package manager, although your mileage might vary.

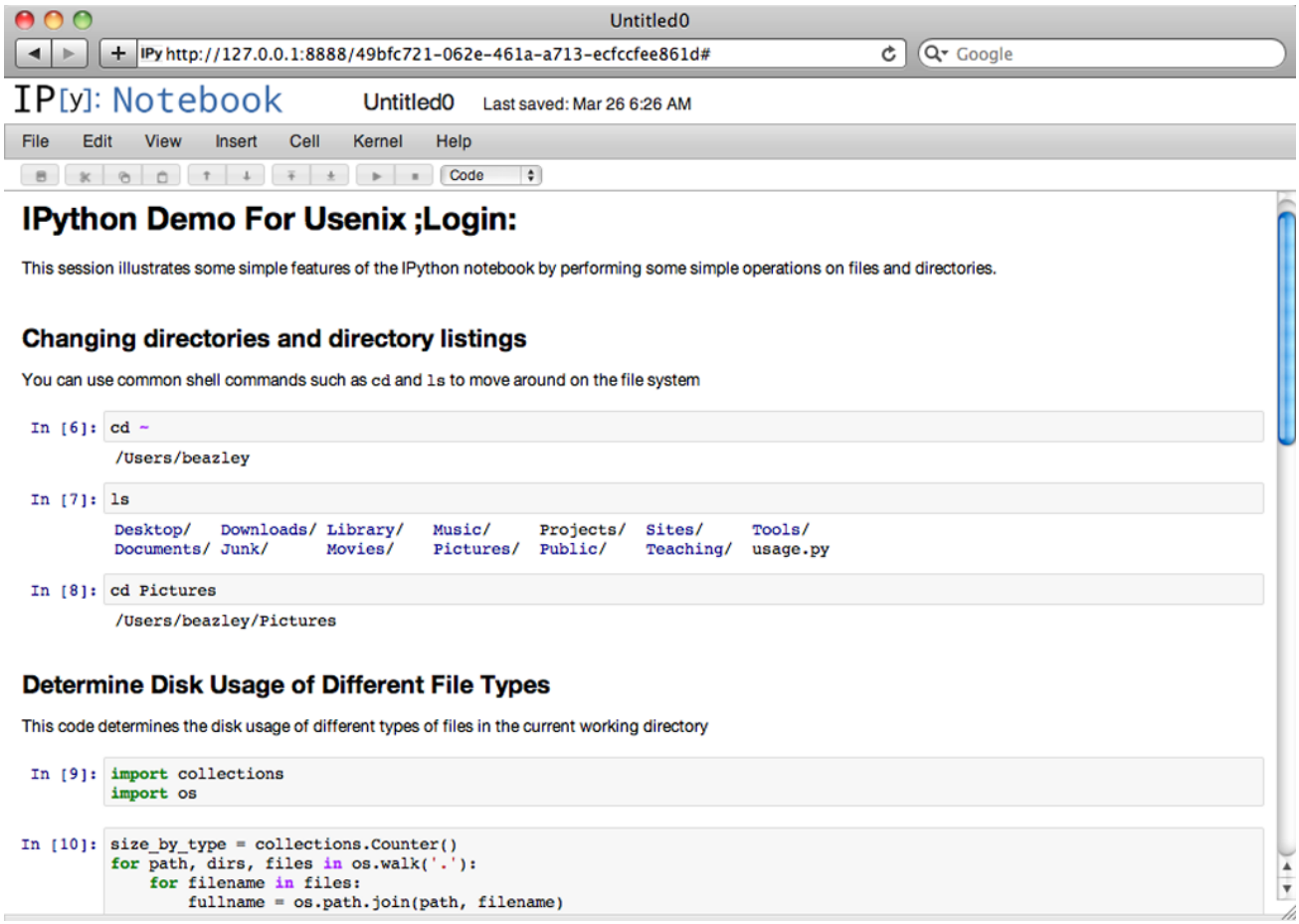


Figure 2: Notebook includes the ability to document what appears in a notebook, using Markdown (https://pypi.python.org/pypi/Markdown)

Assuming you have everything installed, you can launch the notebook from the shell. Go to the directory in which you want to do your work and type "ipython notebook". For example:

```
bash $ ipython notebook
[NotebookApp] Using existing profile dir: u'/Users/beazley/.
   ipython/profile_default'
[NotebookApp] Serving notebooks from /Users/beazley/Work
[NotebookApp] The IPython Notebook is running at:
   http://127.0.0.1:8888/
[NotebookApp] Use Control-C to stop this server and shut down
   all kernels.
```

Unlike a normal session, the Notebook runs entirely as a server that needs to be accessed through a browser. As soon as you launch it, a browser window like the one in Figure 1 should appear.

If you click on the link to create a new notebook, you'll be taken to a page on which you can start typing the usual IPython commands, as in Figure 1.

At this point, the notebook doesn't seem much different from the shell; however, the benefits start to appear once you start editing the document. For example, unlike the shell, you can move around and edit any of the previous cells (e.g., change the code, re-execute, delete, copy, and move around within the document). You can also start to insert documentation at any point in the form of Markdown. Figure 2 shows the above session annotated with some documentation.

Assuming you've installed matplotlib and NumPy, you can also start making inline plots and charts. For example, Figure 3 shows what it looks like to take the file-usage data and make a pie chart.

Needless to say, the idea of having your work captured inside a kind of executable document opens up a wide range of possibilities limited only by your imagination. Once you realize that these notebooks can be saved, modified, and shared with others, why the notebook project is quickly taking over the Python universe starts to become clear. In that vein, I've shared the above

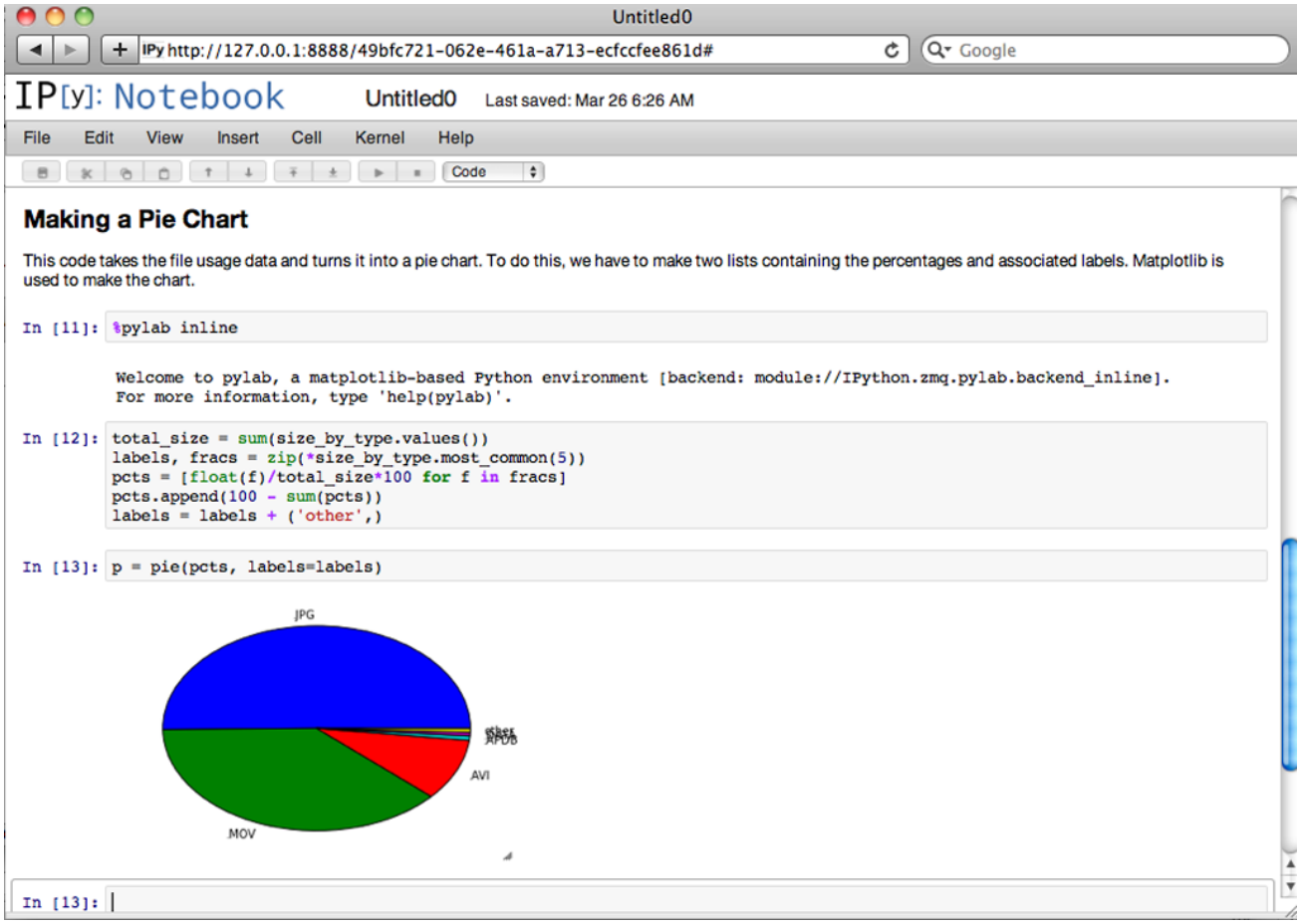


Figure 3: Notebook works with matplotlib and NumPy so you can include inline plots and charts

notebook at http://nbviewer.ipython.org/5244469. You can go there to view it in more detail.

Other Notable PyCon Developments

Although this article has primarily focused on IPython, a few other notable developments were featured at the recent conference. First, there seems to be a general consensus that the mechanism currently used to install third-party packages (the procedure of typing `python setup.py install`) should probably die. How that actually happens is not so clear, but the topic of packaging is definitely on a lot of people's minds. Somewhat recently, a new binary packaging format known as a "wheel file" appeared and is described in PEP-427 (http://www.python.org/dev/peps/pep-0427/). Although I have yet to encounter wheels in the wild, it's something that you might encounter down the road, especially if you're the one maintaining a Python Installation.

Also worthy of note is the fact that Python seems to be gaining a standard event loop. Over the past several years, there has been growing interest in asynchronous and event-driven I/O libraries (e.g., Twisted, Tornado, GEvent, Eventlet, etc.) for network programming. One of the benefits of such libraries is that they are able to handle a large number of client connections, without relying on threads or separate processes. Although the standard library has long included the asyncore library for asynchronous I/O, nobody has ever been all that satisfied with it; in fact, most people seem to avoid it.

Guido van Rossum's keynote talk at PyCon went into some depth about PEP 3156 (http://www.python.org/dev/peps/pep-3156/), which is a new effort to put a standard event loop into the standard library. Although one wouldn't think that an event loop would be that exciting, it's interesting in that it aims to standardize a feature that is currently being implemented separately by many different libraries that don't currently interoperate with each other so well. This effort is also notable in that the PEP involves the use of co-routines and requires Python 3.3 or newer. Could asynchronous I/O be the killer feature that brings everyone to Python 3? Only time will tell.

iVoyeur
Approaching Normal

DAVE JOSEPHSEN

Dave Josephsen is the author of Building a *Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project. dave-usenix@skeptech.org

In the past, I have been known to indulge in hyperbole. I freely admit this, but to say that our Christmas tree erupted into flames wouldn't come close to an adequate description. It exploded into a wholly new form of matter that transcended flames. I tell you, one moment it was a chopped up, dead Christmas tree in a fire-pit, and the next moment it was a churning column of violent, malevolent living fire. It was as if there were something inside that tree. As if the tree had been imbued with some small piece of the soul of Christmas, or maybe anti-Christmas, and we had set it alight. The tree had arms, and a face, and it writhed and screamed and reached out for us, beckoning for us to come closer. We took a few steps back.

"Is that normal?" my wife asked, not taking her eyes off the 15-foot columnar fire-being.

Normal; I paused to consider. To *know* whether a thing is normal, we'd have to define normal, by quantitative measurement and comparison. To honestly *know* the truth of the "normal" interaction between Christmas trees and matches, we'd have to burn every Christmas tree on earth to establish a mean, and compute our Christmas tree's standard deviation from that mean. Then we could objectively say whether or not this was "normal." This is probably an impossible task (but I'm willing to try if you're able to fund me).

"I don't know." I replied, taking note of the location of the garden hose.

Of course, it isn't true that we'd need to burn every Christmas tree on earth. There are many problems that require us to quantify the "normalness" of a given property in a population. And it's not unusual for the population to be large enough to make infeasible the measuring of every member. All we really need is a statistically relevant sample from which we can estimate the normal interaction between Christmas trees and matches.

By "statistically relevant," I'm referring not only to our ability to accurately approximate the normal interaction between Christmas trees and matches, but also to our ability to compute the accuracy of our estimates. Statistics was invented to answer just this sort of (ahem) burning conundrum. Really, this is a data sampling problem of the sort that is important in many systems monitoring and analytics contexts, and will only become more important as the data grows. Further, sampling can help us scale monitoring systems, not only by reducing the amount of measuring we have to do, but also by reducing the metric data we need to transmit.

The Simple Random Sample

Most of what humanity knows about the science of data sampling comes from statistics problems involving subjective human opinions on this or that political issue. Even when the problems involve "harder" subjects, such as crime statistics, our humanity can intrude in ways that make it notoriously difficult to acquire data samples that are statistically relevant. For this reason there are myriad data sampling methodologies that we can ignore entirely in a monitoring context. For our purposes, we will usually be looking for a "simple random sample," or "SRS" [1].

Taking humans out of the sample-collecting picture (mostly) helps us to avoid the common pitfalls that have a negative impact on the accuracy of conclusions statistically derived from sampled data. Selection bias, overcoverage, undercoverage, processing errors, and measurement errors are all out the window. This makes data sampling a really effective technique for the kinds of problems we're usually concerned with in a monitoring context.

Gleaning a simple random sample is, well, simple. We just need to choose a subset of measurements from a larger body of data in a way that every measurement has an equal probability of being chosen as a sample. Imagine, for example, that you've written a program that calls a function foo(). You want to monitor the length of time foo() takes to return, but your program is distributed across several hundred compute nodes, so foo() gets called several million times per second. You could obtain an SRS for foo() return times by generating a pseudo-random number every time foo() is called, and comparing it to a threshold. If the current random number is less than the threshold, then you take a sample return time. If that's too expensive for your purposes, the random requirement can usually be adequately satisfied in practice by simply decrementing a counter and sampling when it hits zero (especially in large populations like foo()).

Sample Size

If we generated a random number between 0 and 1, and used .0025 (a quarter of one percent) for our threshold, we should collect around 2,500 samples for every million calls to foo(). The Central Limit Theorem [2] allows us to make a few assumptions about our sample. For instance, we can assume that our sample mean pretty much equates to our population mean. By "pretty much" I mean our sample approximates the population within some margin of error, which we can compute [3]. In my example our sample should approximate the population within about a 5% margin of error.

As you might expect, if we reduce the sample size, we increase the error percentage. This relationship directly affects the accuracy of our estimates, and means we can increase accuracy by either sampling more often, or sampling for a longer period of time. It also means that it is important to deliberately choose your sample size [4].

Here's something you might not expect: as long as the population is an order of magnitude larger than the sample, the accuracy of our predictions does not vary with the size of the population. In other words, it wouldn't matter if foo() was being called a million times per second, or ten thousand. My 2,500 samples would give me the same accuracy in either case.

That's weird but useful; it means we don't need to think about data samples as a percentage of the total population for many of the problems we're interested in, which certainly helps us scale. It also introduces the irony that smaller populations are more difficult to sample accurately.

Sometimes, Sampling Ruins Everything

sFlow [5] is a great example of a controversial use of data sampling. Sometimes we're interested in knowing the number of occurrences of X in a population, like the number of packets that make up the total population of packets that traversed a given switch port, or the number of bytes sent that were BitTorrent protocol in the population of all bytes.

These numbers are expensive (in a computational sense) to gather and process. Traditional approaches, such as hardware packet taps, span ports, and NetFlow-enabled switches, burn every Christmas tree by either measuring each packet directly or copying each packet to an external entity. The cost of this brute-force endeavor is actualized as more expensive (in dollars and cents) network gear, or slower network gear.

sFlow, by comparison, gleans a simple random sample by decrementing a counter per interface, and sampling the current packet when the counter hits zero. By modeling the sample as a binomial distribution [6] sFlow can, at near zero cost, answer questions like the BitTorrent-related ones above with sufficient accuracy for the purpose of customer billing in the real world. This is clever engineering, and the sFlow creators have obviously [7] put careful thought into its design and implementation. The accuracy of its estimates are guaranteed by math.

What sFlow cannot guarantee, however, is that all classes of traffic actually make it into the sample set. It's entirely feasible that small, one-time bursts of traffic (the packets making up a port scan, for example) might never be sampled, and therefore never be represented in sFlow's output (and this property, by the way, does vary with the size of the population). So while flow statistics of the sort that are interesting to network operations folks are accurately represented, the kind of statistically aberrant thing that might interest the security folks is maybe not.

The Buddha said that it's not what we carry with us but what we let go of that defines us, which is an apropos sentiment in this case. I'm sure it was not a design goal for sFlow to capture these aberrant, one-off micro-flows, but their exclusion renders sFlow unusable for a huge chunk of the reason many of us run packet taps, span ports, and NetFlow, which is to say, intrusion detection and forensics, and, therefore, since we're doomed to incur the cost of those other solutions anyway, belies our use of sFlow entirely. That's kind of sad because I personally like clever engineering, statistics, and things that are cheap (in any sense), but I also think it's possible that data sampling and traffic inspection might not be compatible undertakings.

It's easy, especially for statistics geeks, to become overly fascinated with the center mean line of that perfectly symmetrical bell curve: the normal distribution. When we plot a sample that approximates the normal distribution for some attribute, we know that we've methodologically nailed it. But the prudent monitoring engineer should, in many endeavors, be concerned with the statistically irrelevant, with whether the flaming Christmas tree is, in fact, abnormal instead of normal. Even if the cost is burning them all, it is a cost that should be weighed against the loss of statistically irrelevant but really interesting weirdo observations.

Take it easy.

References

[1] The simple random sample: http://www.ma.utexas.edu/users/mks/statmistakes/SRS.html.

[2] The Central Limit Theorem: http://en.wikipedia.org/wiki/Central_limit_theorem.

[3] Sample means: http://www.stat.yale.edu/Courses/1997-98/101/sampmn.htm.

[4] Choosing a sample size: http://www.itl.nist.gov/div898/handbook/ppc/section3/ppc333.htm.

[5] sFlow: http://www.inmon.com/technology/index.php.

[6] sFlow packet sampling basics: http://www.sflow.org/packetSamplingBasics/index.htm.

[7] sFlow sampling theory: http://www.sflow.org/about/sampling_theory.php.

For Good Measure

The Price of Anything Is the Foregone Alternative

DAN GEER AND DAN CONWAY

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc.
dan@geer.org

Daniel G. Conway is founding director of the MBA in Analytics program at Loras College and is a faculty member in the McCormick School of Engineering at Northwestern University. He previously served on the faculty of Indiana University and the University of Notre Dame.
daniel.conway1@northwestern.edu

Cybersecurity insurance has been talked about forever, but absent some ill-advised government mandate, the insurance market is not going anywhere useful without better data.

A demand for insurance emerges as soon as traditional risk management providers of family, clan, and tribe become too small to help. The first formal insurance was supplied to enable risk transfer around physical assets, which are susceptible to harm by physical forces.

In Nature, physical forces are local. Physical risk mitigation strategy thus requires pooling of risk based on locality independence. For example, the risk of a fire in NY is uncorrelated with, and thus offset by, the risk of a typhoon in Taipei, which is uncorrelated with, and thus offset by, the risk of an earthquake in Istanbul. A successful insurance company diversifies risks geographically so as to remove the impact of the correlation implied in locality. You don't write fire insurance for abutting tenements.

In information security, locality is manifested by systems which, when compromised, have a correlated impact on value. These systems include operating systems, ubiquitous applications, standardized protocols, and a host of other vulnerable single points of failure. For any operating system code base, all instances of it are "virtually local." In essence, this means we have only a few digital cities, each built within the digital world's "ring of fire." Insurance providers cannot offer affordable insurance without a means of diversifying locality, that is to say without limiting the provider's own exposure to cascade failure among their insureds.

In a recent DHS workshop on cyber insurance [1], many suggestions were offered to drive adequate coverage alternatives and thus maturity in the cyber insurance industry. The report cited the difficulty insurance providers faced:

1. a lack of actuarial data, which requires high premiums for first-party policies that many can't afford;
2. the widespread, mistaken belief that standard corporate insurance policies and/or general liability policies already cover most cyber risks; and
3. fear that a so-called "cyber hurricane" will overwhelm carriers who might otherwise enter the market before they build up sufficient reserves to cover large losses.

Difficulty (3) results from locality, so an insurance company would prefer to provide coverage for potential insureds that have system diversity. This can be encouraged through discounts for those with diverse system characteristics and verified through audit or embedded monitoring tools. Difficulty (2) is beyond the scope of this column. We focus on difficulty (1), which has been at the heart of For Good Measure since its inception (http://geer.tinho.net/fgm).

The most reflexive strategy to collect better actuarial data is to impose data sharing through regulation, and that approach can have positive results if accompanied by liability protection; however, the incentives for reporting events completely and accurately are generally unaligned with the organization's individual reward structure, viz., full disclosure exposes firms to litigation and potential additional cyber risks that far exceed any value to be gained from such disclosures. Moral hazard has a digital counterpart.

Year	Incidents
2005	156
2006	643
2007	774
2008	1048
2009	727
2010	828
2011	1088
2012	1586

Table 1: Data Loss Events per Year

A Market Approach

A market approach would induce sharing of actuarial data by providing a framework for rewarding contributed value, which would, as a result, provide inference into event trends. Rewards tend to attract further market participants, often resulting in the maturing of metrics programs and improved management techniques. Analytics in baseball has been good for maturing baseball. Analytics in the cyber insurance industry would catalyze maturation in cyber risk management and are a necessary component of re-insurance.

What would such a market look like, and how might it be used to improve security? A participant in the DHS workshop described (1) the frequency and (2) the severity of events as the "Holy Grail" of cybersecurity risk management, so we start with that. Severity is in the eye of the beholder, and thus subject to stakeholder appetite for risk. Financial markets use spot price of money for this measure, and ignore the beholder's current position and/or money demand. Futures markets and money markets extend the spot concept to a price-over-time concept, and thus allow for better capital planning. (Time-lapsed pricing permits the incorporation of data points, such as the frequency of certain events occurring over time, that spot pricing cannot capture.)

What would a market for "event frequency" as a commodity look like? For data, we take 96 months of event frequency, from 2005–2012, using the Data Loss Database [2] as a proxy. Events by year are represented in Table 1, and events by month in Figure 1.

Financial analytics professionals have created markets to buy and sell probabilities for many domains, including who will be the next president, the next pope, and the next winner of "Dancing With the Stars." During the most recent MIT sports analytics workshop [3], major league baseball teams suggested that their players were evaluated as if they were financial assets and a team was a portfolio of such options on those assets.

If changes in cybersecurity event frequency were important to us, we could treat that frequency as if it were a financial asset,

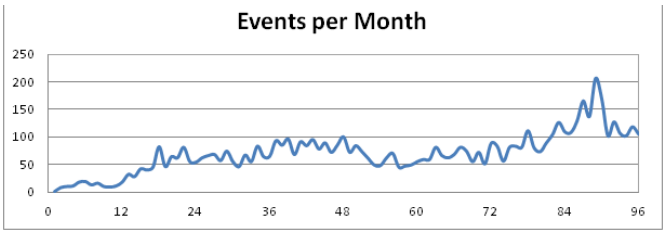


Figure 1: Data Loss Events by Month (2005-2012)

and, more importantly, we could price futures in cybersecurity event frequency. For our example, we will use ticker symbol XEF. This market could be used as a hedge against risks for those most susceptible to an increase or decrease in event frequency, such as cyber insurance providers. Increases in XEF "price" would mean that the market predicts an increase in the frequency of cybersecurity events.

For example, if an email company were measuring the frequency of ".exe" attachments over time and saw a spike in that metric, they could purchase shares of XEF in anticipation of an increase in future cybersecurity events. Any market participant who was sensitive to an increase in such events might purchase an option to buy XEF in the future for a small price today as a risk mitigation instrument. This market would likely be more responsive in terms of expectations than data collected through regulatory imposition.

Option Pricing via Black-Scholes

Black-Scholes option pricing [4] is a widely used calculation method in finance for providing future price information on assets, and is used to price grain futures, weather futures, and the value of major league baseball players. In our case, it would have a price on the future of XEF, that is to say the future frequencies of cyber events. A mature options market in XEF would allow a market participant to purchase the right (but not the obligation) to buy XEF in the future at a set price in exchange for an amount today. Such prices are determined by the volatility of the underlying stock where, in the case of XEF, the underlying is security debt as defined by Wysopal [5].

To be concrete, and again using the monthly data from datalossdb.org as a proxy, if the investor wanted to obtain the right to purchase (call) a share of XEF at a price of 90 in three months from December of 2012, the investor would identify the following:

- Spot price today: 106
- Future strike price: 90
- Risk-free rate (historical monthly increase): 1%
- Volatility: 27%

Months	Call	Put
1	$ 21.00	$ 4.10
2	$ 25.24	$ 7.45
3	$ 28.68	$ 10.02
4	$ 31.65	$ 12.12
5	$ 34.28	$ 13.89
6	$ 36.68	$ 15.44
7	$ 38.88	$ 16.79
8	$ 40.92	$ 18.00
9	$ 42.83	$ 19.09
10	$ 44.63	$ 20.06
11	$ 46.33	$ 20.95
12	$ 47.94	$ 21.76

Table 2: Call and Put Option Prices for XEF

The Black-Scholes calculation would then price at 28.68 (dollars) the option of purchasing a share of XEF in three months at 90 (dollars). Table 2 lists various option prices for a future price of 90. We also include the price for the option to sell (put) a 90 (dollar) share of XEF in the future.

A futures market for event frequency in cybersecurity might offer a way for security professionals to infer future events as well as provide a mechanism to insure against the associated risks. The amount we invest in future calls/puts reflects our perceived impact of an event, thus pushing the severity half of the Holy Grail metric to the beholder.

John Poindexter and others demonstrably understood the potential of derivative markets to serve as predictors of future events, although they were unable to navigate the political obstacles to realize such markets [6]. Now is the time to revisit those ideas; cybersecurity is in crisis, and crises must not be allowed to go to waste.

References

[1] Cybersecurity Insurance Workshop Readout Report: http://www.dhs.gov/sites/default/files/publications/cyber-security-insurance-read-out-report.pdf.

[2] DataLossDB: http://datalossdb.org/.

[3] MIT Sloan Sports Analytics Conference: http://www.sloansportsconference.com/.

[4] Investopedia: http://www.investopedia.com/terms/b/blackscholes.asp.

[5] C. Wysopal, "A Financial Model for Application Security Debt": http://www.veracode.com/blog/2011/03/a-financial-model-for-application-security-debt.

[6] R. Hanson, "The Policy Analysis Market: A Thwarted Experiment in the Use of Prediction Markets for Public Policy," Innovations, Summer, 2007: http://hanson.gmu.edu/innovations.pdf.

/dev/random

ROBERT G. FERRELL

Robert G. Ferrell is a fourth-generation Texan, literary techno-geek, and finalist for the 2011 Robert Benchley Society Humor Writing Award. rgferrell@gmail.com

At first I thought Ganeti clusters were some form of yummy confection, like peanut brittle, only gooier. Then it occurred to me that they might instead be an exotic stellar architecture discovered since I had my college astronomy courses in 1977. Or perhaps they were a neuroanatomical feature that wasn't covered during my abortive attempt at medical school in the late '80s, like the lateral geniculate nucleus, only gooier. I was discombobulated to discover that I was, as too often happens, mistaken. I didn't really mind, though, as I had been combobulated since the last cold front came through and I was running out of muscle rub.

Clustering as a task list item first buzzed around in my personal lateral geniculate nucleus back in 1997, when I was given the job of constructing a load balancing high-availability cluster for the USGS using four Sun Enterprise 450 servers running that fancy new Solaris 2.6. Back then clustering software was a bit less sophisticated than it is today. The flow control algorithm resembled the digital version of four old men sitting around in rocking chairs on a porch when the phone rings inside. Those of you under 30 may need to consult Wikipedia for some of this. Believe it or not, there was a time when phones were anchored to the wall and we got our news from large sheets of flimsy paper folded and rolled into a tube for delivery by boys on bicycles. We had to be alert and snag those tubes off the front lawn before the dinosaurs got to them.

Server #1 (Clem): The phone's ringin'.

Server #2 (Wally): Yep.

Server #3 (Rufus): 'Peers so.

Server #4 (Cooter): Uh huh.

Clem: Think we oughta answer it?

Rufus: I answered it last time.

Cooter: Probably not important, anyways.

Wally: My hip's not right. Somebody else get it.

Clem: Fine. I'll get it.

Wally: Here comes the paper boy. Anybody got five bucks?

Rufus: I thought the paper was four-fifty.

Wally: Gotta tip the little skeeter or he drops it out in the sticker patch.

Cooter: I think I got some cash left over from Bingo last night.

Wally: What happened to Rufus?

Cooter: Had to go to the john.

Wally: Dang. My wife's hollerin' for me. Gotta git.

Cooter: Hope the phone doesn't ring.

Apparently, somewhere in the middle of this imaginary exchange, I forgot exactly what point I was trying to make, but what the heck: you get the gist (or if you don't, send a self-addressed stamped tweet to @whatwasItalkingaboutagain and whoever owns that tag will tell you to shove off). Load balancing and high availability were largely left to chance in those far off days of six-shooters and Conestoga tape drives. I am pleased to report that not only has clustering software made quantum leaps since then, some of it is even open source. What's not to like? So cluster away, my fine fellows, and th' divil tak' th' hindmost!

Moving on to a topic about which I know a little bit more (prepositions at the end of sentences is something up with which I, along with Churchill, will not put), I recently developed a rather elementary training guide for people who are Windows literate but who have had little to no exposure to *nix. The guide covers a few specific areas integral to the job these folks are expected to perform, nothing more. You would think that might not be much a challenge, but brothers and sisters I'm here to testify that this seemingly simple task is deceptively difficult. I'd rather train a wombat to do my taxes.

My presentation starts out laudably enough: a few useful systems administration commands, such as ls, su, sudo, cd, ifconfig, ps, chmod, and so on; nothing too complex there. But the intellectual pudding begins to congeal quickly as I try to figure out some straightforward way to explain file permissions. Remember, these are dyed-in-the-wool Windows people, and all I have is an hour or so and PowerPoint to work with.

I try several approaches. The first results in something that reads like instructions for assembling a double helix using the little-known r, w, and x nucleotides. (What time does the nucleotide come in today, please?) My second attempt is a reasonably decent representation illustrating the effects of the dread affliction alphabetiasis dementiae. Following these abject failures are a spastic Scrabble irruption, four or five paragraphs of what can only be described as toxic literary sludge, and finally, a frankly puzzling series of statements that when read aloud make a noise that calls to mind a mechanical dog barking whilst immersed in a bath of peanut butter, potato chips, and ball bearings. I decide to go with this one.

Because I am already firmly in the tempestuous throes of incoherency by this point, I go for broke and tackle (not to be confused with block and tackle, which is the process of using football players to carry heavy stuff for you) the thorny subject of auditing. Where file permissions were an abecedarian's febrile nightmare, the subject of audit flags and settings comes out sounding like the lyrics to a Tom Lehrer song, minus the rhyme scheme and comprehensibility. I suggest reading them to an accompanying soundtrack; Nine Inch Nails' "Head Like a Hole" might work well. Or not.

Access control is a little easier to understand, but my cumulative brain damage from formulating the preceding lessons results in a rather uninformative series of short, choppy statements that give the impression of having been lopped off the main narrative by some automatic trimming machine that goes "whoosh whoosh" with nasty long pointy blades slicing smoothly through the conjunctions and prepositions as though they were nothing more than insubstantial denizens of a labored metaphor embedded in a run-on sentence. And baby makes three.

Come to think of it, the entire project is a total washout. I think I'll just scrap it and write a cookbook, instead.

22nd USENIX SECURITY SYMPOSIUM

WASHINGTON, D.C. • AUGUST 14–16, 2013

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks.

USENIX Security '13 will feature:

Keynote Address

"Dr. Felten Goes To Washington: Lessons from 18 Months in Government"

Edward W. Felten, *Director, Center for Information Technology Policy, and Professor of Computer Science and Public Affairs, Princeton University; former Chief Technologist, U.S. Federal Trade Commission*

A 3-day Technical Program including:

- Paper presentations on large-scale systems security, attacks, mobile security, and applied crypto
- Invited talks by industry leaders
- Poster session
- Rump session
- Birds-of-a-Feather sessions (BoFs)

Register by July 22 and Save!
www.usenix.org/sec13

Co-Located Workshops Include:

EVT/WOTE '13: 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections
August 12–13, 2013

CSET '13: 6th Workshop on Cyber Security Experimentation and Test
August 12, 2013

HealthTech '13: 2013 USENIX Workshop on Health Information Technologies
Safety, Security, Privacy, and Interoperability of Health Information Technologies
August 12, 2013

LEET '13: 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats
August 12, 2013

FOCI '13: 3rd USENIX Workshop on Free and Open Communications on the Internet
August 13, 2013

HotSec '13: 2013 USENIX Summit on Hot Topics in Security
August 13, 2013

WOOT '13: 7th USENIX Workshop on Offensive Technologies
August 13, 2013

Stay Connected...

twitter.com/usenixsecurity
www.usenix.org/youtube
www.usenix.org/gplus
www.usenix.org/facebook
www.usenix.org/linkedin
www.usenix.org/blog

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks, and operating systems, book reviews, and reports of sessions at USENIX conferences.

Access to *;login:* online from October 1997 to this month: www.usenix.org/publications/login/

Access to videos from USENIX events in the first six months after the event: www.usenix.org/publications/multimedia/

Discounts on registration fees for all USENIX conferences.

Special discounts on a variety of products, books, software, and periodicals: www.usenix.org/membership/specialdisc.html.

The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.

For more information regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

USENIX Board of Directors

Communicate directly with the USENIX Board of Directors by writing to board@usenix.org.

PRESIDENT
Margo Seltzer, *Harvard University*
margo@usenix.org

VICE PRESIDENT
John Arrasjid, *VMware*
johna@usenix.org

SECRETARY
Carolyn Rowland, *National Institute of Standards and Technology*
carolyn@usenix.org

TREASURER
Brian Noble, *University of Michigan*
noble@usenix.org

DIRECTORS
David Blank-Edelman, *Northeastern University*
dnb@usenix.org

Sasha Fedorova, *Simon Fraser University*
sasha@usenix.org

Niels Provos, *Google*
niels@usenix.org

Dan Wallach, *Rice University*
dwallach@usenix.org

CO-EXECUTIVE DIRECTORS
Anne Dickison
anne@usenix.org

Casey Henderson
casey@usenix.org

New Exclusive Electronic Edition: *;login: logout*

If you haven't seen it already, please take a look at our new electronic-only supplement to *;login:* magazine, *;login: logout*. Published every other month, *;login: logout* will appear during the months when *;login:* magazine is not published, giving you year-round *;login:* content. Each issue will contain at least three new articles. The inaugural issue for May 2013 features:

- James Mickens on "The Saddest Moment"
- Selena Deckelmann on "The Disambiguator: Learning about Operating Systems"
- Rik Farrow on "So many filesystems..."

Enjoy!

Notice of Annual Meeting

The USENIX Association's Annual Meeting with the membership and the Board of Directors will be held on Wednesday, June 26, 2013, in San Jose, CA, during USENIX Federated Conferences Week, June 24-28, 2013.

Tell Us What You Think

Please watch your inboxes and the *;login:* Web site at www.usenix.org/publications/login/ for a link to a *;login:* readership survey. We'd like to hear what you think about the types of articles and authors we've been featuring; our current columnists; the recently refreshed magazine design; *;login: logout,* our new electronic supplement; and more. Thanks for your assistance in making *;login:* even more valuable to you, our readers.

SAVE THE DATE!

LISA'13

27th Large Installation
System Administration Conference

Sponsored by USENIX
in cooperation with LOPSA

NOVEMBER 3–8, 2013 | WASHINGTON, D.C.

www.usenix.org/lisa13

Come to LISA '13 for training and face time with experts in the sysadmin community.

LISA '13 will feature:

6 days of training on topics including:

- Configuration management
- Cloud Computing
- Distributed Systems
- DevOps
- Security
- Virtualization
- And More!

Plus a 3-day technical program:

- Invited Talks
- Guru Is In sessions
- Paper presentations
- Vendor Exhibition
- Practice and Experience reports
- Workshops
- Posters and Lightning Talks

New for 2013! The LISA Labs "hack space" will be available for mini-presentations, experimentation, tutoring, and mentoring.

Register Now!
www.usenix.org/lisa13

Stay Connected...

twitter.com/lisaconference
www.usenix.org/facebook
www.usenix.org/youtube
www.usenix.org/linkedin
www.usenix.org/gplus
www.usenix.org/blog

Book Reviews

TREY DARLEY AND MARK LAMOURINE

Burdens of Proof: Cryptographic Culture and Evidence Law in the Age of Electronic Documents

Jean-François Blanchette
MIT Press: 2012, 280 pp.
ISBN: 978-0262017510

Reviewed by Trey Darley

Blanchette's thesis is that while cryptographers spend their days in the world of pure mathematics, they exist in messy, human socio-historical contexts, and, consequently, efforts to model that world in protocol form are fraught with latent, ofttimes unconsidered, assumptions. Blanchette provides sufficient background in both the history and practice of cryptography and of evidence law to draw both technical and legal audiences into his discussion. As IANAL, I found the background material on contracts, witnesses, the notarial system (as opposed to common law practices more familiar to me), and the privileged evidentiary status of authenticated acts both fascinating and helpful. On the cryptographic side, Blanchette does an admirable job capturing technical details whilst still writing in language that should be understandable to a general audience, albeit hopefully a well-caffeinated one.

He narrates how, back in the '90s, various interest groups, feeling themselves encroached upon by the advance of technology, drove the legislative reform agenda on cryptographic signatures. Shockingly, it seems that the resultant regulations for the most part failed to address the vital point of signature verification. Blanchette shows how the concept of nonrepudiation flies in the face of traditional judicial discretion. Cryptographers assume, he argues, that judges will think about cryptographic primitives like cryptographers would and, as such, existing protocols make unhelpfully high crypto-literacy demands.

This is a wide-ranging book. I was taken aback by how many avenues for further research it opened. For example, I never considered the impact that format-transcoding (necessary to maintain future-proof digital archives) has on signature verification (and, hence, document authentication). If we're building a paperless world in which 500-year-old documents will be more transparent than 50-year-old ones, then clearly the modeling has gone badly off the rails. Anyway, just something to think about. If you are at all interested in crypto, you'll probably dig this book. Pay close attention to Blanchette's chapter summaries, which are remarkably trenchant.

Practical Lock Picking: A Physical Penetration Tester's Training Guide, Second Edition

Deviant Ollam
Syngress Media, 2012, 296 pp.
ISBN: 978-1597499897

Reviewed by Trey Darley

The idea that information security begins in meatspace is an accepted cliché, but in practice it's all too easy to get distracted by OSI layers 2–7. There's nothing quite like the experience of popping your first lock to awaken your senses to weak physical security all around you. If you haven't had the pleasure already, I would encourage you to let Deviant Ollam be your guide through the world of picking, raking, shimming, and bumping. After you read this book, with its diagrams clear enough to be understood by a child and plenty of helpful hints on assembling a toolkit, you probably won't look at your front door the same way again.

Vintage Tomorrows

James H. Carrott and Brian David Johnson
Maker Media, 2013, 398 pp.
ISBN 978-1-449-33799-5

Reviewed by Mark Lamourine

Carrott and Johnson had a beer and a question. Three questions, actually: "Why Steampunk?" "Why now?" and "What does it mean for the Future?" The book is the story of their inquiries and reflections. Along the way they visit Australia, the UK, Seattle, Dallas, Comicon, and Burning Man (twice), among other places. They accidentally spawn a documentary film, which they end up documenting.

This isn't your average sociology paper. The text alternates between first person accounts by each of the authors as they travel to meet the people they interview, visit conventions, and even host a dinner gathering of Steampunk luminaries. The authors invite the reader to participate in the journey and the conversations.

Carrott is a historian who likes to immerse himself in his subject. As a teen, he was a Civil War reenacter, and for this book he first visits and then participates in Burning Man in Nevada. His tech background includes managing the development of the XBox 360. Johnson is a professional futurist, projecting trends as much as 10 years out to help Intel guide their research. The historian and the futurist use each other as sounding boards for their ideas and questions.

If you're familiar with Steampunk at all you'll probably know at least a couple of the authors that they meet. You may or may not know of the artists, tailors, craftsmen, many of whom were doing what they do before the term was coined. As a long-time reader of Bruce Sterling and William Gibson, I found myself thinking both "Oh, cool" and "Well, of course" within a single sentence more than once.

Carrott and Johnson find that the Steampunk movement isn't a simple one-dimensional fad of nostalgia. The visual and literary trappings of the 19th century resonate with different groups of people and, remarkably, none of them are Luddites who want to live in the past. There are the hangers-on who think that to make something Steampunk you just "stick a gear on it," but a central tenet of the Steampunk movement is individual active participation in the process of shaping our surroundings, clothing, tools, and technology. Participants value the craftsmanship of unique items as a response to what they see as the modern sterile cookie-cutter design ethic. They are optimistic about the use of technology that contrasts sharply with the trend toward dystopian literature since World War II.

Did I mention a movie? The process of writing the book inspired documentary filmmaker Byrd McDonald to follow the authors on many of their visits. A trailer is up at http://www.vintagetomorrows.com. A release date hasn't been announced.

The authors are also adding more to the book over time in the form of a companion (DRM-free) ebook: *Steampunking Our Future: An Embedded Historian's Notebook,* available from O'Reilly (though it will take a bit of search-fu to find it apart from *Vintage Tomorrows*).

I got copies of *Vintage Tomorrows* in several ebook formats as well as hard copy. Each has advantages. The photographs in the paper book are rendered in half-tone black and white. The ebook images are full color images; however, I found that, with one exception, the images were cropped in odd ways rather than re-sizing on my ereader (a no-name 7-inch tablet running vendor and third-party reader apps). The exception was the PDF version, in which the images were scaled nicely; however, the PDF version was almost four times bigger than the EPUB or MOBI versions. Also, when you scale the text for easier reading, you're really zooming, and the text is cropped rather than wrapped.

Corsets and top hats making it back into most people's everyday lives is unlikely, but the Steampunk ethos is having an influence on mainstream thought and sensibilities. The optimism and joy of makers, hackers, and geeks are gradually making enthusiasm for learning and technology acceptable again. The nerd of the 1960s, '70s, and '80s is becoming intelligent, witty, and stylish. *Vintage Tomorrows* shines some light on the way that we are constructing both our future and our past.

Testable JavaScript
Mark Ethan Trostler
O'Reilly Media, 2013, 250 pp.
ISBN 978-1-449-32339-4

Reviewed by Mark Lamourine

There are any number of books that will tell you how important it is to write tests. In the ones I've read, little time is given to the elements of software that can make it hard to test. More than once I've found myself looking at a test routine that just smells bad without understanding why.

In *Testable JavaScript,* Trostler explains how to recognize the characteristics of hard-to-test code and how to avoid writing it. The early chapters cover the concepts of code complexity: cyclomatic complexity, fan-out, and coupling. Trostler proceeds to describe how event or message-driven systems can provide the ultimate in decoupling (with their own set of caveats).

This section makes up about the first half of the book and was the most valuable to me. The concepts of complexity are fairly subtle. Recognizing and then mitigating these elements in code will take some practice. I suspect I'll come back here a number of times over the next few months. This isn't something that was in the college curriculum when I was a student, but I'm guessing the concepts glossed in these three chapters could fill a full semester of undergraduate work.

There are references to a number of books, papers, and articles in those opening chapters. Many of the references are accompanied by permanent bit.ly URLs. While I can fish back through the text to find them later, a proper bibliography would be nice.

From here on the title of the book could be seen as a bit of a misnomer. The remainder of the book seems to go back to the more typical topics.

The unit testing and coverage sections continue the mix of theory and practice, though the practice begins to come to the fore. The chapter on unit testing opens by glossing the concepts of mocks, stubs, and spies (a new one on me). The next few sections introduce testing frameworks for client-side testing in Web browsers and Android devices and closes with more traditional server-side testing in Node.js.

The next chapter introduces the concept of code coverage, that is, the idea of exercising every path and branch in your code. The concept is generally applicable, but the tool and techniques presented are for JavaScript only. Trostler is cautious about the value of code coverage metrics, but shows how the use of automated instrumentation can improve the quality of the results.

The book closes with chapters on integration and performance testing, in-browser debugging, and test automation. The tools

available in most browsers are both impressive and pretty slick, taking advantage of the capabilities of the graphical interface.

This is a book about JavaScript programming. It could take some additional effort to puzzle through for someone who's not fluent. If you can manage, most of the techniques and patterns in the first half of the book are applicable to (and valuable for) other object-oriented and procedural languages. I would recommend this book if only for that. If you're also looking for some new tricks, you'll find something here.

EPUB 3 Best Practices

Matt Garrish and Markus Gylling
O'Reilly Media, 2013, 345 pp.
ISBN 978-1-449-32914-3

Reviewed by Mark Lamourine

I think the most important thing I learned from *EPUB 3 Best Practices* is that there's a lot more to building electronic documents than I would have imagined. The authors sum up an EPUB document this way: It's a Web server in a box.

EPUB 3 is the most recent open electronic document standard. It's actually defined by four specifications. These define the format for the content, structure, packaging, and "media overlays." This last one is new to EPUB and it describes how to sync audio and text for things such as subtitles. The specifications define the function and limitations of each of the features. *EPUB 3 Best Practices* describes how to use them.

Each of the chapters covers an aspect of the EPUB 3 format. While there is a progression, and you can read the book cover to cover, you can also dive into any one of the chapters without missing anything.

EPUB 3 documents are composed using other current standards. The content must be XHTML5 or SVG. Note that this refers to the document as a whole. HTML documents can refer to images in formats other than SVG.

The rest of the glue is XML or CSS. There are a set of standard fonts, and you can embed additional fonts in a document with the OTF or WOFF formats (there are translators for others). While the HTML5 audio and video codec discussions continue, the EPUB 3 specification requires reader software to support MP3 and AAC (MP4) audio. Video is another matter, and the authors stick to describing the implications of the ongoing ambiguities on EPUB 3 documents and reader software.

Interactivity is provided by a required JavaScript engine, which allows the inclusion of dynamic graphics and forms. There is a chapter on language support, another on accessibility, and a third on providing text-to-speech capabilities. Including external resources through standard HTML links is possible, and there are provisions for alternate media if a network is not available.

I like the fact that the authors address several non-technical issues with EPUB production. There is a fairly detailed discussion of the need and means to acquire the rights for proprietary fonts before embedding them. As noted above, the authors devote a portion of the chapter on fonts to their proper use. I think there are a number of instances in which a judiciously placed structure graphic might have helped illuminate how the parts fit together.

As with *Vintage Tomorrows*, I read this book in paper, EPUB, and PDF. In the case of books with code samples, I find the ebooks difficult on small and medium-sized tablet devices. Code often has been laid out carefully in a typeset book, and the otherwise laudable ability of an ebook reader to re-flow the text based on the font size and the device becomes a problem.

I don't expect ever to have the need to create an EPUB 3 document from scratch and by hand. If I do, or if I ever find myself needing to look inside one, I'll keep this book handy. This is a great book for the curious, and I suspect it could be required reading for people meaning to write an EPUB 3 editor, compiler, or reader.

Conference Reports

FAST '13: 11th USENIX Conference on File and Storage Technologies

San Jose, CA
February 12-15, 2013

Summarized by Mathias Bjorling, Thanh Do, Rik Farrow, Min Li, Leonardo Marmol, Muthukumar Murugan, Dorian Perkins, and Morgan Stuart

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Keith Smith began FAST 2013 by telling us that 127 papers were submitted and 24 accepted, and that the attendance had almost reached the same level as 2012, which was the record year for attendance. There were 20 full-length papers, four short ones, nine with just academic authors, five industry-only authors, and ten collaborations. Smith said he enjoyed having people from academia and industry in the same r oom talking.

FAST is a systems conference, and the top topics in submitted papers were those tagged with file-system design and solid state storage. Cloud storage has increased over time, as has caching, while file-system architectures have actually been decreasing.

There were over 500 paper reviews, totaling more than 350,000 words of commentary.

Yuanyuan Zhou, the co-chair, presented the best paper awards. *Unioning of the Buffer Cache and Journaling Layers with Non-Volatile Memory* by Lee et al. won the Best Short Paper award, and *A Study of Linux File System Evolution* by Lu et al. won Best Paper.

File Systems

Summarized by Morgan Stuart (stuartms@vcu.edu)

ffsck: The Fast File System Checker

Ao Ma, EMC Corporation and University of Wisconsin—Madison; Chris Dragga, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Ao Ma considered the creation of a file system that supports a faster checking utility. Ao first reviewed the necessity of file system integrity checkers and repairers. He explained that significant work has been done to prevent corruption or misuse of file systems, but no solution guarantees a system free of such corruption. Therefore, the file-system checker is often thought of as a last resort, but it hasn't seen much improvement in some time. Given capacity increases, complexity growth, and general enterprise dependence, that storage admins must still depend on offline, slow, unpredictable file-system checkers is unfortunate.

In order to significantly improve file-system checking, the standard e2fsck utility was analyzed. The e2fsck checker completes its repairs in five phases, but the authors found that the utility spends more than 95% of its time in phase 1. During this phase, the checker scans all inodes and their corresponding indirect blocks and even requires an additional scan if multiply-claimed blocks are detected. Any improvements to file-system checking clearly should target the actions performed in this phase. Ao introduced a novel pairing of a file system, rext3, and a file system checker, ffsck, both of which complement each other to accelerate file-system checking.

The rext3 file system modifies the layout of a traditional ext3 file system to decouple the allocation of indirect blocks and data blocks. The indirect region in rext3 stores the indirect blocks contiguously to allow quick sequential access. In applying this reformation, rext3 achieves an improved metadata density that a modified checker could leverage. The strict organization also reduces file system aging from fragmentation. The separation proves not to result in extraneous seeks because a drive track buffer will often cache multiple indirect blocks with a single track load.

The fast file system checker (ffsck) leverages the contiguous indirect blocks of rext3 to increase scan speed. Because the indirect blocks and corresponding data blocks are physically rather than logically sequenced, however, ffsck requires that all metadata be read before it can be checked. In order to avoid memory saturation from storing the entire metadata of large systems, ffsck separates the checking process into its inherent two phases: a self-check phase and a cross-check phase. The self-check phase must use all the metadata to verify the file inodes individually, but the cross-check phase only needs a subset of the metadata in order to perform redundancy-based checks across data. Therefore, the self-check is completed first, followed by the removal of data not needed by the cross-check, and finally the cross-check is performed. This method helps reduce the average memory footprint over time for the checker.

The end result of the file-system file-checker cooperation is the ability to scan and correct the file system at nearly ten times the speed of e2fsck and without hindrance from disk fragmentation. In most cases, rext3 performance is similar to ext3, but does incur about a 10% penalty when dealing with smaller files. Impressively, rext3 actually outperforms ext3 by up to 43% for random reads and up to 20% for large sequential writes. These improvements are attained by improving journal checking with the metadata density and by more efficiently using the track buffer.

Andreas Dilger (Intel) asked whether the authors had considered submitting an upstream patch to ext3. Ao said that the source code still needs to be polished, but they do intend to

See the complete FAST '13 reports online at: https://www.usenix.org/publications/login

open source their work. Someone from Symantec Labs asked what level of performance increase was seen without the file-system modifications. Ao explained that they can still achieve between 50% to 200% improvement with only the in-order scan. Brent Welch (Panasas) requested more details about the indirect region—specifically, whether they enforced a hard limit and what they did if the region was filled. Ao said that the size was fixed and that further experimentation is required as an appropriate size is hard to determine.

Building Workload-Independent Storage with VT-Trees

Pradeep Shetty, Richard Spillane, Ravikant Malpani, Binesh Andrews, Justin Seyster, and Erez Zadok, Stony Brook University

Pradeep Shetty began with a simple question: "What should file systems do?" He explained that file systems must allow for crash recovery, perform efficiently for both sequential and random accesses, and provide low overhead application-level transactions. Most real-world solutions don't meet all of Pradeep's requirements, giving him and his co-authors their motivation. Pradeep further alluded to the major discourse for today's administrators: they can either choose fast lookup transaction-based relational databases or instead opt for file systems that support high volumes of sequential and random accesses. Often neither is completely sufficient, as modern workloads are large and complex with randomized access patterns.

The proposed solution describes a new data structure, the VT-Tree, based on LSM-Trees. The LSM-Tree provides fast random insertions but significantly slower queries, making the LSM-Tree popular in large data sets where queries can be parallelized. The LSM-Tree uses a memtable to hold r-tuples of recently inserted items in a buffer. Once the buffer fills, the memtable, along with a Bloom filter and secondary index, is flushed to disk. The combination of these components is referred to as an SSTable. Consequently, the workload produces more SSTables as more tuples are created. Because queries often must search through the majority of the SSTables, the queries slow down over time. To combat this, a limit on the number of SSTables is typically used to bound the lookup and scan latency of the system. A primary weak point of the LSM-Tree is its repeated copying of tuples as SSTables are serialized and compacted to the disk. These copies in the minor compaction allow for the quick lookup, but are considered unnecessary if the incoming data is already sorted.

Following his explanation of the LSM-Tree, Pradeep began to outline the goals of their VT-Tree. To optimize the minor compaction, which produces the extra copies, stitching was introduced. Pradeep described stitching as a way in which their system investigates the need for a merge during compaction. The stitching mechanism allows the VT-Tree to merge only the blocks that overlap and stitch non-overlapping blocks into appropriate locations. The repositioning of the tuples to perform a stitch introduces fragmentation and holes in the tree. This is prevented by storing the VT-Tree on a log-structured block device to allow a LFS-style defragmenter to reclaim lost space. The stitching threshold is the minimum size that a stitched region must accomplish in order for stitching to occur. This threshold therefore helps limit the level of system fragmentation. The method for avoiding I/O in LSM-Trees is to use a Bloom filter, but the VT-Tree uses quotient filters instead to allow rehashing in RAM without the need for an original key.

Pradeep next outlined their file system, KVFS, and how it utilizes VT-Trees to provide novel functionality to a system. In actuality, KVFS translates requests into key-value operations that are then sent to KVDB, which then performs the necessary I/O operations. Three dictionary formats—nmap, imap, and dmap—can be used to create dictionaries, each backed by a VT-Tree. The nmap format is used for namespace entries, the imap format simply stores inode attributes, and the dmap format is used for the data blocks of files. The system's ACID transactions are snapshot-based, where each transaction gets its own private snapshot. This allows the system to avoid double writes and implement the standard begin, commit, and abort operations of transactional systems.

The resulting system, Pradeep described, performs comparable to other systems but achieves the enhanced performance of standard LSM-Trees when performing random writes. This means that the VT-Tree can support both file system and database workloads efficiently. The transactional architecture supported by the VT-Trees provides 4% speedup with 10% overhead.

Peter Desnoyers (Northeastern) expressed concern about the system's background cleanup and asked whether the authors had pursued a way to adjust the stitching threshold to prevent cleaning from overloading the system. Pradeep said that they experimented with many thresholds and found 32 Kb or 64 Kb to work best. He added that while increasing the threshold may reduce fragmentation, it would negate the purpose of including the threshold at all if it was increased too much. Margo Seltzer (Harvard) asked how their implementation differs from LFS segment cleaning. Pradeep agreed that it is indeed similar and that they only look at the sequential data and examine the rate on it. The questioner further encouraged Pradeep to look at the age of the data as well.

A Study of Linux File System Evolution

Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Shan Lu, University of Wisconsin—Madison

Awarded Best Paper!

As winner of Best Paper, this analytic research presented a fascinating look into a significant portion of the Linux 2.6 file system development. The authors painstakingly reviewed 5096 patches across six file systems and categorized each patch as bug, performance, reliability, feature, or maintenance related.

Lanyue Lu began by noting the continued importance of local file systems to a crowd ripe with cloud storage enthusiasts. He explained that local storage is still common on mobile devices, desktops, and as the base file system for many cloud applications. Lanyue said that much can be learned from a large-scale development review, such as understanding where complexity comes from, how to avoid future mistakes, and to help improve current designs and implementations.

This comprehensive study required that Lanyue and his associates comb through eight years of Linux 2.6 file system patches. Utilizing each patch's commit message, code diffs, and community discussions, the authors accumulated granular data describing the process of developing major open source file systems. The file systems examined included ext3, ext4, XFS, Btrfs, ReiserFS, and JFS.

The researchers found that code maintenance and bug fixes account for the majority of the patches sampled, at 45% and just under 40%, respectively. Lanyue noted that the maintenance patches were deemed uninteresting early on and were not investigated in detail. The bug patches were further categorized into either semantic, concurrency, memory, or error code related bugs. Semantic bugs were the biggest offenders, making up more than 50% of all bug-related patches. Concurrency bugs were the next most common at about 20%. Interestingly, nearly 40% of these bug patches occurred on failure paths or error handling. Other than the bugs, performance and reliability patches also made up a significant portion of the patches studied, accounting for 8% and 7% of patches, respectively.

The results suggest that bugs do not necessarily diminish over time, as one might presume. Even the stable, well-tested, file systems seem to have a relatively constant rate of bug patches during the period. Of all the possibilities, data corruption bugs were the most dominant across all the file systems studied and caused the most severe problems, such as system crashes and deadlocks. Lanyue went on to discuss actual patch examples from each patch category, pointing out the types within each category responsible for the most patches.

Lanyue said that, although time-consuming, a large-scale study such as this is manageable and useful. He stressed the importance of research matching reality and said that history does indeed repeat itself.

Akshat Aranya (NEC Labs) asked whether any correlation between feature patches and bug patches was studied. Lanyue recognized this as possible area of study but said that he and his co-authors did not analyze it. Margo Seltzer asked just how in depth the group's initial study of maintenance fixes was before deeming them "uninteresting." Lanyue responded that these maintenance bugs were almost always attempts to simplify the core structure through refactoring and that relating it to a bug patch was difficult. Rick Spillane (Apple) asked about bug fixes introducing another bug. Lanyue confirmed that they found these and even labeled them as "fix-on-fix" patches.

The data set is available at http://research.cs.wisc.edu/wind/Traces/fs-patch/.

Caching

Summarized by Leonardo Marmol (marmoleox@gmail.com)

Write Policies for Host-Side Flash Caches

Ricardo Koller, Florida International University and VMware; Leonardo Marmol and Raju Rangaswami, Florida International University; Swaminathan Sundararaman and Nisha Talagala, FusionIO; Ming Zhao, Florida International University

Ricardo Koller began his presentation by pointing out the big performance gap between write-through (WT) and write-back (WB) policies for caches. Traditional caching solutions for network storage, he said, implement WT policies because these guarantee data consistency at the price of experiencing high latencies for every update. Inspired by the "Designing for Disasters" work, he described two new caching policies for locally attached SSDs, designed to perform similarly to WB while preserving point-in-time consistency. The first policy, ordered write-back (OWB), uses a graph to store the ordering dependencies for I/Os using only issue and completion times, in order to evict the block in the right order. The second policy, journaled write back (JWB), builds a journal on the cache and evicts transactions atomically over the network using an interface similar to that of Logical Disk. This policy also required modification to the network storage in order to write blocks atomically.

The experimental results showed the benefits that come with caching, not only read but also write requests for those applications that can tolerate some level of staleness. The caching solution was evaluated using several benchmarks, and the results showed that, in general, WT performed worse than any other policy. JWB outperformed OWB but not traditional WB. Other experiments were presented showing the throughput and the number of I/O updates sent to storage as a function of the cache size for each policy.

Wenguang Wang (Apple) asked whether it was possible to relax the ordering constraints of the OWB policy, pointing out that, typically, hard disks acknowledge the completion of writes once the data is in their internal buffer, and then these are not necessarily performed in the same order they were issued. Koller disagreed with the questioner's premise, saying that the order of writes in non-volatile caches sent to disk is maintained and therefore matters.

Warming Up Storage-Level Caches with Bonfire

Yiying Zhang, University of Wisconsin—Madison; Gokul Soundararajan, Mark W. Storer, Lakshmi N. Bairavasundaram, and Sethuraman Subbiah, NetApp; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Caching solutions determine the contents of the cache on-demand. As I/O requests come and go, the content of the cache changes to better reflect the current needs of the application. New technologies such as SSDs have made it possible to increase the size of caches to be much bigger than DRAM memory, which slows the process of warming caches. To put things into perspective, Zhang mentioned that a cache of 1 TB takes about 2.5 hours to fill with sequential workloads and six days or more with random workloads. For these reasons, Zhang claimed that the on-demand approach to warming up caches is no longer appropriate. She proposed a solution, Bonfire, that monitors and logs I/O requests with the goal of speeding up the warming of caches by loading data in bulk.

To answer questions such as what and how the data should be monitored and logged, and how to load warmed data into caches efficiently, Zhang et al. performed statistical analysis on the MSR-Cambridge traces [Narayanan '08]. The temporal and spatial access patterns found on the traces were used to shape the design goals of their system. Bonfire monitors I/O requests with a module that sits below the buffer cache and keeps a buffer for its own metadata. When the buffer is nearly full, this is written to a persistent storage in a circular log. When a cache restarts, Bonfire uses its metadata to load warm data from storage into the cache. In addition to metadata, Bonfire could also log data that can be used to further reduce the warm-up time.

The system was evaluated by replaying the MSR-Cambridge traces in a synchronous fashion using both metadata-only and metadata+data logging schemas and comparing them to the on-demand and always-warmed policies. The results showed that Bonfire could warm up caches from 59% to 100% faster than on-demand while reducing the storage I/O load by 38% to 200%. As a consequence, the I/O latency experienced by applications was reduced on average by 1/5 to 2/3 when compared to on-demand. Before concluding, Zhang mentioned the need for making more trace available to the research community and invited everyone to contribute.

Umesh Maheshwari (Nimble Storage) asked Zhang why they assumed that caches are volatile when they could use SSDs as caches and they are persistent. Zhang explained that even persistent caches need rewarming after a repartition of the cache or a server migration. Ajay Gulati (VMware) asked about the case in which workload does not follow the patterns seen in the study's traces. Zhang replied that Bonfire would default to on-demand. Someone asked how they kept stored data and Bonfire's buffer consistent. Zhang answered that the buffer is updated only after the data is written to storage. The questioner pointed out that this requires some form of synchronization among nodes sharing storage. Joe Buck (UC Santa Cruz) mentioned that Zhang's research group's logo is very similar to that of the Nintendo GameCube logo.

Unioning of the Buffer Cache and Journaling Layers with Non-Volatile Memory

Eunji Lee and Hyokyung Bahn, Ewha University; Sam H. Noh, Hongik University

Awarded Best Short Paper!

Eunji Lee pointed out that journaling is one of the most common techniques used by file system architects to provide data consistency. However, it comes at the cost of extra I/O operations. Lee suggested an interesting alternative that eliminates the extra I/Os associated with journaling while maintaining the same level of reliability by effectively using non-volatile memory. In particular, she argued for an architecture called UBJ that unifies the buffer cache and the journal into non-volatile memory. Unlike conventional journaling techniques, committing blocks in UBJ is a simple matter of marking them as frozen, eliminating both the copy operation and data duplicate. In addition, frozen blocks are used as cache, reducing the latency of read requests. As in any journaling system, transactions are eventually checkpointed to storage, but UBJ makes use of copy-on-write techniques to allow the update of frozen blocks. This simple technique significantly reduces the latency of write-intensive workloads.

The UBJ prototype was implemented in Linux, and the NVM was simulated using DRAM. The implementation was compared to the ext4 file system configured to journal both data and metadata, and several I/O benchmarks were used to generate the workloads. The results showed that UBJ outperformed ext4 with 59% higher throughput, which translated into a 30% reduction in execution time on average. Due to the simplicity of the UBJ system's in-place commit mechanism, the latency of I/O is no longer dependent on the frequency of commit operations.

Richard Spillane (Apple) commented on how the buffer cache and in-memory journal were sized on one of the experiments. He suggested that she could have gotten a better performance by reducing the size of the journal and increasing the size of the cache, as opposed to making them equal in size. Youyou Lu (Tsinghua University) asked about the performance penalty associated with protecting frozen blocks and performing the COW. Lee replied that the overhead is expected to be very small, but no data was available at the time.

Conference Luncheon

During the conference luncheon, the Test of Time award was presented for GPFS: A Shared-Disk File System for Large Computing Clusters, by Frank Schmuck and Roger Haskin of IBM Almaden Research Center. You can read this paper via http://static.usenix.org/publications/library/proceedings/fast02/schmuck.html.

Protecting Your Data

Summarized by Morgan Stuart (stuartms@vcu.edu)

Memory Efficient Sanitization of a Deduplicated Storage System

Fabiano C. Botelho, Philip Shilane, Nitin Garg, and Windsor Hsu, EMC Backup Recovery Systems Division

Storage sanitization can be described as any method that removes sensitive data from a device, such that it appears the guarded information never actually existed on the system. Effective sanitization methods have their place in many fields, including the government and highly regulated private sectors. With the rise of massive storage systems and deduplication, there is a need to revisit sanitization mechanisms.

After explaining the modern motivation for advanced sanitization methods, Fabiano Botelho explained that crypto-sanitization isn't a contender in this particular area for several reasons. Key management would be difficult in these large systems where blocks are shared in the namespace. Furthermore, crypto-sanitization sacrifices performance of normal file system operations to achieve its goal. Finally, the NIST and DOD do not accept encryption as a sanitization method. Fabiano solidified their requirements, stating that their solution must completely erase deleted data, maintain the availability of live data, use resources responsibly, and leave the storage system in a usable state while sanitization is being performed.

The widespread technique of deduplication and the need for bulk sanitization are the primary motivators of Fabiano's work. When files are written in a deduplicated storage system, the data is separated into chunks and each chunk's hash value is calculated. The hash value can be used to determine whether or not the chunk is unique, whether or not it needs to be stored. Files in these systems are represented as a list of fingerprints that can be used to reconstruct the original file. This methodology allows only one instance of duplicate chunks to be stored on the system, saving large amounts of storage. However, these chunk references present the primary challenge when attempting to sanitize a deduplicated storage system.

Fabiano and his co-authors investigated several methods of tracking unused data and objects known as dead chunks. After comparing the possible usage of reference counts, Bloom filters, bit vectors, and perfect hashing, they found that perfect hashing can best fulfill their requirements. Perfect hashing allows a mapping without collisions of a static key set, using a minimal number of bits to represent the mapping. The perfect hash function will map to perfect hash buckets that are variable size, but 16K fingerprints per bucket on average worked very well.

The five-step algorithm for read-only file system sanitization has Merge, Analysis, Enumeration, Copy, and Zero phases. The Analysis phase was described in more detail as the point in which the algorithm builds the perfect hash function, walks multiple perfect hash vectors in parallel, and records the range that the data structure is actually covering. An algorithm for read-write systems was also implemented, which must handle incoming fingerprints after both the Merge and Analysis phases. These chunk resurrections are handled by notifying the process of incoming deduplication and utilizing a second consistency point, or snapshot, to enumerate differences.

Three sets of experiments were used to formulate a control for storage systems. First, a system using only local compression with no deduplication wrote its entire file system at about 70 MB/s. Next, deduplication was added with a factor of about 7x, resulting in 5.06 GB/s data rates. This increase in performance correlating to the deduplication factor confirms that a system can scale performance according to the deduplication factor. The next benchmark added the sanitization mechanism as well as data ingest, running at 59% peak throughput and 70% peak throughput, respectively. Fabiano explained that this benchmark showed that the system's data ingest is CPU-intensive while the sanitization is I/O-intensive. A final benchmark removed the deduplication, leaving the sanitization and data ingest variables. Sanitization ran above 45% of its peak throughput in this test, with high CPU usage for the data ingest as well as high I/O usage for both sanitization and ingest.

Cheng Huang (MSR) asked if the deduplication system must hold all the fingerprints in memory in the first place. Fabiano recommended that Cheng attend the HP session the next day, where they describe techniques to avoid holding everything in memory. Cheng then asked whether the authors had looked into options other than the perfect hash data structure. Fabiano explained that they had not seen any better techniques.

SD Codes: Erasure Codes Designed for How Storage Systems Really Fail

James S. Plank, University of Tennessee; Mario Blaum and James L. Hafner, IBM Almaden Research Center

The past ten years have seen rapid growth in utilization of erasure codes to handle disk failures. However, recent research has exposed what James Plank terms the "RAID6 Disconnect": that storage administrators are effectively using entire disks to tolerate common latent sector errors rather than full disk failures. Latent sector errors are particularly bothersome because they are typically only detected once a read access is attempted on the data. This clearly wasteful use of resources has motivated James and his co-authors to develop an erasure code that can tolerate both full disk failures and latent sector errors. The goal is to allow administrators to devote the right amount of coding to match the failure mode.

James explained the theoretical view of a stripe to define their Sector-Disk (SD) code. More specifically, each disk holds r w-*bit* symbols in a system of n disks, where w is relatively small.

The system also uses *m* disks and s sectors per stripe to tolerate simultaneous failures of any *m* disks plus any *s* sectors per stripe. The SD code uses Galois Field arithmetic, where more w-bit symbols decrease speed but make the code more robust. James noted that this Reed-Solomon-like code has large amounts of documentation and open source code, and said he would spare the audience the mathematics.

The SD code is slower than Reed-Solomon, but it outperforms the solutions that the SD code could replace. For example, replacing RAID6 with a *one-disk-one-sector* code achieves higher performance with less dedicated storage. A complete open source implementation, in C, was made available the week of the conference. The source code is intended to act as template for engineers wishing to use the code or experiment with it.

Someone pointed out that the assumption seems to be that the latent errors are somewhat random and therefore small in number, but disk drives, instead of flash drives, could have many kilobytes in error. James explained that the implementation of the code must have the sector size defined as sufficiently large to encompass these larger failures. Geoff Kuenning asked "What am I getting?" since the SD codes don't really solve the two disk failures previously resolved by RAID6. If RAID6 is used, you are already protected from both types of failures. James explained that if you want to allow for more disk failures, you need to increase the *m* for disks. He suggested that models be used to examine the need to tolerate these failures. IBM researchers performed data loss modeling to investigate the data loss of SD coding versus RAID6 and they showed that SD can get higher reliability.

HARDFS: Hardening HDFS with Selective and Lightweight Versioning

Thanh Do, Tyler Harter, and Yingchao Liu, University of Wisconsin—Madison; Haryadi S. Gunawi, University of Chicago; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Thanh Do began by describing the implementation of the cloud's reliability, describing it as complex systems made up of thousands of commodity machines, where once-rare failures become frequent. In these systems, machine crashes and disk failures have been generally mitigated. However, Thanh described "fail-silent" failures as a continuing problem for these large-scale systems. The fail-silents are failures where the machine or program exhibits incorrect behavior but doesn't entirely halt the system. These failures can be caused by many problems, but are often the result of corrupt memory or software bugs. The defining factors of fail-silent failures are that standard crash recovery mechanisms do not combat them, as the issue can quickly propagate across collaborating machines. Current solutions for these failures, found in N-Version programming's redundant implementation methodology, require extensive resources and engineering effort which results in its rare deployment.

Thanh introduced selective and lightweight versioning (SLEEVE) to combat the fail-silent failures that he describes. Rather than "telling a lie" by continuing to run after a silent failure, SLEEVE exploits the crash recovery support for systems if it detects a failure with its trusted sources. Detection of the erroneous operations is achieved by utilizing a second lightweight implementation of the functionality that requires SLEEVE's protection.

SLEEVE is described as selective due its small engineering effort and its ability to target important functionality for protection. For instance, the error checking can target bug sensitive portions of a program or system, such as subsystems that are frequently changed or even currently unprotected with internal mechanisms. The lightweight aspect of SLEEVE describes the absence of full state replication. Instead, SLEEVE encodes states to reduce required space. The hardened version of HDFS (HARDFS), protected with SLEEVE, was able to detect and recover from 90% of random memory corruption faults and 100% of the targeted memory corruption faults. Furthermore, HARDFS was able to detect and recover from five software bugs injected into the system.

SLEEVE is composed of four subsystems: an interposition module, state manager, action verifier, and a recovery module. The state manager only maintains important states of the main version and only adds new states incrementally. The state manager must also understand the semantics of the protocol messages and events in order to correctly update the state. The state manager encodes states with counting Bloom filters, which supports insert, delete, and exist operations. Thanh noted that Bloom filter false positives are rare and that they simply lead to a tolerable yet unnecessary recovery. The action verifier performs micro-checks to detect incorrect actions in the main version. The recovery module supports both full recoveries, described as a crash and reboot, and micro-recoveries in which corrupted states are repaired from trusted sources.

The HARDFS implementation hardens HDFS's namespace management, replica management, and its read/write protocol. Thanh and his co-authors found that HARDFS reduced the number of silent failures from 117 to 9, which ultimately increased the number of crashes from 133 to 268. Additionally, by using the counting Bloom filter, their implementation incurred a relatively small space overhead of 2.6%. Thanh concluded by saying that a crash is better than a lie and that HARDFS turns these lies into crashes and leverages existing recovery techniques to bring systems back online.

John Badger (Quantum) asked about the Bloom filter and how facts are represented. Thanh said that only yes/no verification is supported and that it ultimately depends on the property you want to check; no "magic rule" can be applied. Brent Welch

expressed concern about false positives, crashes, and the potential for a crash loop. Thanh agreed that this was possible and informed the audience that crashes can be counted and a hard limit for stopping crash loops can be enacted. Next, Rick Spillane cautioned against Thanh's statement of the Bloom filter's 2.6% overhead, telling him that it grows linearly. Finally, Jacob Lorch pointed out that since SLEEVE is the ultimate arbiter of the system, a bug in SLEEVE can potentially cause catastrophic consequences.

Big Systems, Big Challenges

Summarized by Min Li (limin@cs.vt.edu)

Active Flash: Towards Energy-Efficient, In-Situ Data Analytics on Extreme-Scale Machines

Devesh Tiwari, North Carolina State University; Simona Boboila, Northeastern University; Sudharshan Vazhkudai and Youngjae Kim, Oak Ridge National Laboratory; Xiaosong Ma, North Carolina State University; Peter Desnoyers, Northeastern University; Yan Solihin, North Carolina State University

Devesh presented Active Flash, an in-situ data analysis method, to help improve the performance and energy efficiency for scientific data analysis tasks. He started with the introduction of a two-step process of scientific data analysis which consists of scientific simulation and data analysis and visualization. Conventionally, data analysis is performed offline on a small-scale cluster involving expensive data migration between compute and storage infrastructure resulting in extra energy cost. Devesh observed enabling trends: SSDs are increasingly adopted in HPC for higher I/O throughput and energy efficiency; SSD controllers are becoming powerful; idle cycles exist at SSD controllers due to the natural I/O burst of scientific workload etc. Devesh proposed conducting scientific data analysis on SSD controllers in parallel with simulation without affecting I/O performance.

He organized the discussion of system design around two questions: (1) if SSD are deployed optimizing only I/O performance, is active computation feasible? (2) how much energy and cost saving can Active Flash achieve? The main constraints of SSD deployment without active computation support are capacity, performance, and write durability. On the other hand, modeling active computation feasibility depends on simulation data production rate, staging ratio, and I/O bandwidth. Their results showed that most data analysis kernels can be placed on SSD controllers without degrading scientific simulation performance. Moreover, he observed, additional SSDs are not required to sustain the I/O requirement of scientific simulations even with active computation enabled. Compared with an alternative approach of running active computation on partial simulation nodes, he suggested that Active Flash is able to achieve the same performance but with lower staging ratio and infrastructure cost.

He went on to analyze the energy and cost saving of Active Flash. Modeling a Samsung PM830 SSD, they considered multiple components such as energy consumption of I/O, compute idle periods, data movement, etc. He also mentioned briefly how they modeled the energy consumption of two other state-of-the-art approaches. The results showed that Active Flash is more cost and energy efficient compared with other approaches in many cases. Finally, he introduced the prototype which they developed, based on the OpenSSD platform, demonstrating that scientific data analytics with Active Flash is viable with OpenSSD.

Dave Anderson (Seagate) wondered whether SSDs have enough resources to perform the complex task designed in the paper. Devesh replied that he had researched several products and believed that the SSD controller will be more powerful and have more cores to do complex tasks, such as data analytics, in the near future. Song Jiang (Wayne State University) asked if some intelligence is implemented on the SSD controller. Devesh replied yes. The implementation allows the SSD controller to communicate with hosts and perform data analytics. Song followed up by asking how the active cache handles data that is striped across SSDs. Devesh said that in that case, they would need frameworks such as MapReduce to help coordinate between different SSDs and perform analysis.

MixApart: Decoupled Analytics for Shared Storage Systems

Madalin Mihailescu, University of Toronto and NetApp; Gokul Soundararajan, NetApp; Cristiana Amza, University of Toronto

Madalin started by pointing out that enabling data analytics platforms, such as MapReduce and Pig, to directly use data on enterprise storage can help eliminate the two-storage-silos problem. The traditional two storage silos require dedicated compute infrastructure and additional time to migrate the data, and increase the hardware cost in terms of expense and number of errors. Madalin then presented MixApart, a scalable on-disk cache which allows distributed computation frameworks to use single enterprise storage and supports transparent on-demand data ingestion.

Effective design of MixApart comes from the analysis and understanding of MapReduce workloads. Madalin introduced three key insights they observed: (1) jobs exhibit high data reuse rate; (2) the input phase of a MapReduce job is usually CPU intensive; (3) the I/O demands of jobs are predictable. He also showed that with a high data reuse rate, MixApart can effectively support around 2000 parallel tasks using an envelope calculation demonstrating the compute scale of MixApart. With the goal of preserving the scalability and performance gained from data locality and efficient bandwidth utilization of storage, cache, compute node, Madalin mentioned that MixApart designed per-job task I/O rates and job scheduling policy to maximally overlap computation with data fetch. More specifically, he introduced two components, a compute scheduler, which allows assigning map tasks to nodes with cached data, and a data

transfer scheduler, which facilitates just-in-time parallel data prefetch within and across jobs based on job I/O rate prediction. He also illustrated MixApart in action by using an example. They reengineered Hadoop by implementing a cache-aware compute scheduler as a variant of the Hadoop task scheduler, and a data transfer scheduler as a module within the namenode. They also reused the namenode as the XDFS metadata manager and added support within HDFS to enable caching stateless data. In their evaluation, they ran MixApart on Amazon EC2 with three types of EC2 instances and compared with Hadoop. They found that MixApart can reduce job durations by up to 28% compared to the traditional ingest-then-compute approach and can closely match the performance of Hadoop when the ingest phase is ignored for HDFS.

Akshat (NEC Labs) asked whether they had considered the workloads that were I/O intensive in the Map phase. Madalin admitted that there is not much they can do if the workloads are I/O intensive in the Map phase. However, the Facebook trace they analyzed had shown that the average task's effective I/O rate is low, which allows moving data from the shared storage to distributed cache. He argued that there are efforts to scale out the shared storage system to provide more bandwidth, which enables MixApart to sustain large clusters. He also mentioned that they had the notion of limiting the network bandwidth consumption of MixApart to make sure it does not compete with regular network traffic. Kadir Ozdemir (EMC) asked whether they had thought of a case in which the system would affect the performance of the enterprise system. Madalin responded that they had done some experiments in terms of performance isolation, arguing that the quanta-based scheduling effectively minimized the interference effects. Joe Buck (UC Santa Cruz) asked whether he had noticed a trace from CMU which demonstrated that 99% of data are actually processed within an hour, which means a better strategy would be to stream the data directly into the cache system instead of just-in-time prefetching. Madalin replied that it was a great use case. Since their approach is more generic, their system could always plug in better prefetching schemes to accommodate special scenarios like the one just mentioned.

Horus: Fine-Grained Encryption-Based Security for Large-Scale Storage

Yan Li, Nakul Sanjay Dhotre, and Yasuhiro Ohara, University of California, Santa Cruz; Thomas M. Kroeger, Sandia National Laboratories; Ethan L. Miller and Darrell D. E. Long, University of California, Santa Cruz

Li began by pointing out that current HPC systems store their sensitive data using an unencrypted or simply encrypted approach, which increases the chance of data leakage due to an increased chance of compromised nodes within these large-scale HPC centers. These HPC systems depend on a vulnerable security model which has a hard exterior and a soft interior. There are also concerns of leaking critical information from both malicious insiders and untrusted service providers. However, he mentioned that traditional data encryption techniques could not be directly applied to peta-scale data sets since they are either coarse-grained or incur high key-management overhead. Moreover, they could not provide security even when few nodes are compromised or when the service provider is untrusted. To solve the problem, Li introduced their system, Horus, which enables fine-grained encryption-based security for peta-scale data sets with low key management overhead. The key idea was to use keyed hash trees (KHT) to generate different keys for each region of a file and allow keys to be produced for variously sized regions based on users' need. He stressed that by carefully designing KHT, Horus greatly simplified key distribution and key storage.

Li explained how Horus is made up of three major components: key distribution cluster (KDC), Horus client library, and key exchange protocol. KDC is stateless and independent from the storage and compute nodes within the HPC system, which can help provide security, scalability, and easy deployment. Because only the KDC knows the root key while compute nodes receive the needed keys , any data leakage is confined when nodes are compromised. He then explained the key distribution process through an animation followed by a description of key distribution protocol. The experiments testing the raw performance of KDS showed that a single KDS can sustain about 140,000 queries per second, and it scales linearly with the number of KDSes. Next, he presented an experiment to adjust the system parameters, KHT branch and depth, in order to explore the tradeoff of shifting workloads between servers and clients. He showed that Horus is flexible enough to balance the compute resource between the KDS client and the network. He concluded that Horus supports fine-grained security, is easily deployed, and has high performance.

Mark Lillibridge (HP Labs) asked how to revoke permissions. Li answered that they chose to have two root keys for a file; when a client tries to access a region, it will test which key works for the file. The paper has a detailed discussion. Xubin He (Virginia Commonwealth University) asked how to handle a case in which keys are randomly scattered. Li replied that the read/write workloads are usually in a range. If the depth of KHT is as big as 28, Xubin followed up, what would be the overhead? Li replied that the KHT needs width not depth, and suggested referring to the paper for more details. Bill Bolosky (Microsoft Research) suggested trying different hash functions. Li responded that the focus here was to study the property of KHT; choosing hash function could be future work. Bill said that using an inappropriate hash function would affect the performance. Li admitted that was true.

Poster Session and Reception I

Summarized by Muthukumar Murugan (muru0007@umn.edu)

SLM: Synchronized Live Migration of Virtual Clusters Across Data Centers

Tao Lu, Morgan Stuart, Xubin He, Virginia Commonwealth University

The authors address the problem of live migration of virtual clusters across geographically distributed datacenters. They claim that synchronizing the migration of all VMs in a virtual cluster can reduce the cost of communication and data sharing among VMs through the low bandwidth WAN and hence can avoid any significant performance degradation in the applications.

The proposed architecture has three components: (1) a status monitor to monitor the available resources and the resources currently used by VMs; (2) a migration simulator that predicts the migration impact on the performance of the VMs based on modeling and profiling of the system; and (3) a migration manager that initiates and schedules the migration of each VM. Contact: Tao Lu, cstao.lv@gmail.com

Energy-Aware Storage

Yan Li, Christina Strong, Ignacio Corderi, Avani Wildani, Aleatha Parker-Wood, Andy Hospodor, University of California, Santa Cruz; Thomas M. Kroeger, Sandia National Laboratories; Darrell D.E. Long, University of California, Santa Cruz

This work tries to address the problem of energy consumption in future large-scale HPC storage systems. The two issues that are addressed are providing high bandwidth and/or capacity under power constraints and reducing data movement to save power. The work proposes a new metric called "energy score," which accounts for the energy consumed by all components in the process of the data object generation and is comparable between systems. The work explores multiple options such as near-node storage, use of SSDs, and extensive use of compression, and it studies the impact of proposed approaches on energy consumption of the storage systems.

In order to evaluate the proposed approaches on large complex computer systems, the authors built a comprehensive energy simulator. They also proposed exploring energy-efficient data allocation to increase idle times in storage devices so that they can be transitioned to low-power modes. Contact: Yan Li, yanli@ucsc.edu

On-Demand Indexing for Large Scientific Data

Brian A. Madden, Aleatha Parker-Wood, Darrell D.E. Long, University of California, Santa Cruz

This work proposes an efficient on-demand indexing scheme for large-scale scientific data. The proposed system consists of three components: the filter, the indexer, and the storage substrate. The filtering process creates a map of files to features and attributes. The indexer manages the indices on the filtered data and avoids expensive parsing of all files by narrowing the search based on the filter data. Transducers specific to different file formats help in the filtering process as data is ingested. The filter and index are stored as column stores which serve as the storage substrate. Currently transducers have been built for CSV and XML formats, and Apache HBase is used as the column store. Contact: Brian A. Madden, madden@soe.ucsc.edu

Efficient Use of Low Cost SSDs for Cost Effective Solid State Caches

Yongseok Oh, Eunjae Lee, University of Seoul; Jongmoo Choi, Dankook University; Donghee Lee, University of Seoul; and Sam H. Noh, Hongik University

In this work the authors propose the use of Hybrid Solid State Cache (HySSC), a combination of SLC (Single Level Cell) and TLC (Triple Level Cell), to reduce the cost of Solid State Caches by integrating high performance SLC with low cost TLC. HySSC manages the SSC device, takes care of page replacement in the cache, and maintains the mapping between logical and physical blocks. HySSC manages SLC SSC as read/write and TLC SSC as read-only. The proposed architecture is evaluated with the extended version of the DiskSim simulator and real-world workload traces. Contact: Yongseok Oh, yongsukoh@gmail.com

Energy-Efficient Cloud Storage Using Solid-State Drive Caching

Jorge Cabrera, Salma Rodriguez, Jesus Ramos, Alexis Jefferson, Tiffany Da Silva, Ming Zhao, Florida International University

This work explores the use of SSDs as a near-node storage layer to reduce the power consumption of storage systems. SSDs consume a lot less power than hard disks and are much faster than hard disks for certain workloads. The work uses a modified version of an existing SSD block-caching solution called DM-Cache to enable a write-back cache for the primary storage. A user-space daemon is implemented to talk to the shared storage layer in order to spin down or spin up the disks.

The experiments are carried out on a shared storage device with and without the SSD cache layer. The authors report significant savings in power consumption when the I/O requests are served from SSDs. Contact: Jorge Cabrera, jcabr020@fiu.edu

Cloud Storage System which Prohibits Information Leakage on Both Client and Server

Kuniyasu Suzaki, Toshiki Yagi, Kazukuni Kobara, National Institute of Advanced Industrial Science and Technology (AIST); Nobuko Inoue, Tomoyuki Kawade, Koichiro Shoji, SciencePark Corporation

This work proposes a mechanism to prevent information leakage on client and servers in a cloud storage system. The proposed system, Virtual Jail Storage System (VJSS), encrypts a file using All-Or-Nothing Transform (AONT), and cuts out a part of the encrypted file as a split tally. The split tally is stored in a local storage in the client, and the remaining portion of the file is stored in the cloud storage system after encoding with Reed-Solomon error correcting code. The original file is only reconstructed in the VJSS which has the corresponding split

tally. The encryption and split tally prevent information leakage from servers.

The reconstructed file in the VJSS can be opened by a suitable application but cannot be copied, printed, or screen-captured and pasted. These actions are prevented by the access control library called NonCopy. NonCopy hooks APIs of the Windows kernel, functions of DLL, and event handler I/O APIs, and prevents the action related to information leakage. The current VJSS implementation is based on Loopback Content Addressable Storage (LBCAS) for Windows, which uses "Dokan" for user-mode file system and BerkeleyDB for managing data. Contact: Kuniyasu Suzaki, k.suzaki@aist.go.jp

Offline Deduplication-Aware Block Separation for Solid State Disk

Jeongcheol An and Dongkun Shin, Sungkyunkwan University
Summarized by Vasily Tarasov (tarasov@vasily.name)

Jeongcheol An presented a deduplication-based technique that increases the lifespan of Solid State Disks (SSDs). The method consists of inline and offline steps. During the inline step, the SSD computes a CRC32 checksum of every incoming chunk (the size of a chunk is equal to SSD's page size). CRC32 is not a collision-resistant hash, so it is used to classify chunks into those containing unique data and those of undetermined status. CRC32 is 12.5 times faster than collision-free hash functions such as SHA-1, so write latency is not severely penalized by the inline step. The data that is classified as unique is separated on SSD from undetermined data. Later, during the offline step, the actual deduplication with strong SHA-1 hashes is performed. The number of pages invalidated by the deduplication in the undetermined area is significantly higher than when no block separation is used and, consequently, the number of page copies during garbage collection decreases considerably (by up to 5 times in some experiments). Associated write amplification diminishes and the lifespan of the SSD increases. Contact: Jeongcheol An (luckyjc7@skku.edu)

Extension of S3 REST API for Providing QoS Support in Cloud Storage

Yusuke Tanimura, National Institute of Advanced Industrial Science and Technology (AIST); Seiya Yanagita, National Institute of Advanced Industrial Science and Technology (AIST) and SURIGIKEN Co., Ltd.

Though popular today, the S3 REST API does not allow a user to specify performance reservations for read and write throughput. Yusuke Tanimura presented an extension to the S3 REST API that provides a QoS capability to the base protocol. The extension adds new optional arguments to the already existing 'PUT Bucket' and 'Put/Get Object' operations. In the 'Put Bucket' operation, a user can specify the bucket size, its lifetime, and read/write throughput reservations. In the 'Put/Get Objects' operation, one can specify a reservation ID. Reservations in this case are made using an external tool, but in the future such commands can be added to the main protocol. The authors implemented the extension for Papio backend, which already supports QoS internally. Preliminary results demonstrate a good control over the throughput reservations. Contact: Yusuke Tanimura (yusuke.tanimura@aist.go.jp)

Improved Analysis and Trace Validation Using Metadata Snapshot

Ian F. Adams and Ethan L. Miller, University of California, Santa Cruz; Mark W. Storer, NetApp; Avani Wildani and Yangwook Kang, University of California, Santa Cruz

The fact that an I/O trace does not miss important activities is a crucial requirement for making true trace-based conclusions about the workload. Ian Adams presented an interesting approach for determining the coverage of a trace. Before the tracing starts, an initial file system metadata snapshot is taken. Immediately after the tracing is over, another snapshot, called a reality snapshot, is taken. By applying the trace records to the initial snapshot, one can obtain a so-called expected snapshot. The analysis of the differences between the expected and the reality snapshots allows identifying the coverage of the trace. The authors provide several examples of such an analysis that determines the periods of the logger failure, missing creates, renames, and permission changes. Contact: Ian F. Adams (iadams@soe.ucsc.edu)

An Efficient Data Deduplication Based on Tar-Format Awareness in Backup Applications

Baegjae Sung, Sejin Park, Youngsup Oh, Jeonghyeon Ma, Unsung Lee, and Chanik Park, Pohang University of Science and Technology (POSTECH)

Sejin Park presented an approach to improve the chunking algorithm for tar-files. It is known that typical tar-files consist of numerous concatenated sub-files. Traditional chunking algorithms, such as fixed chunking and content defined chunking (CDC), ignore sub-file boundaries, which degrades the deduplication ratio. The authors added to the Opendedup SDFS file system the ability to form chunks using the sub-file boundaries in tar files. Their experiments demonstrate that deduplication ratio for 20 Linux kernel sources in a single tar file increased from 2.5 for CDC to almost 8.5 for CDC with tar-aware chunking. Contact: Sejin Park (cipark@postech.ac.kr)

GreenDM: A Versatile Hybrid Drive for Energy and Performance

Zhichao Li, Ming Chen, and Erez Zadok, Stony Brook University

Zhichao Li and Ming Chen presented a design for a novel device mapper target—GreenDM. GreenDM rests on top of several block devices with varying performance and power consumption characteristics, e.g., SSDs and HDDs. Using a number of approaches to determine the hotness of the data, GreenDM transparently migrates the data between SSDs and HDDs to improve performance and reduce power consumption. Preliminary results demonstrate up to 330% performance improvements and up to 80% power savings. Contact: Zhichao Li (zhicli@cs.stonybrook.edu) and Ming Chen (mchen@cs.stonybrook.edu)

Using Hybrid Cloud and Mobile Platforms to Enhance Online Education

Rachel Chavez Sanchez and Ming Zhao, Florida International University

Moodle is a known open source educational system similar to BlackBoard. Currently it lacks the integration with virtualization technologies, where each student could, for example, have his or her own VM for the experiments. Rachel Chavez Sanchez presented vMoodle, an educational system that incorporates Virtual Machines (VMs) in Moodle. vMoodle supports Web-based and mobile application interfaces. For mobile application, the authors worked on developing intelligent caching algorithms to improve user experience when high-latency networks are employed. Another problem the researchers tried to tackle is the support of live VM migration from a private to public cloud. This can be useful in cases when the university, for example, does not have enough resources to run all VMs on its own hardware. Contact: Rachel Chavez Sanchez (rchav010@cs.fiu.edu)

Policy-Based Storage System for Heterogeneous Environments

Dai Qin, Ashvin Goel, and Angela Demke Brown, University of Toronto

Applications are often decoupled from storage even though these applications and file systems produce a variety of workloads. Most modern storage systems are not aware of application workloads and requirements and interact with the upper layers using a simple block interface. According to Dai Quin and his colleagues, solutions like ZFS and Btrfs that integrate storage management in a file system are not flexible enough for heterogeneous environments. Instead, the authors propose a modular framework that determines application semantics using previously developed introspection and hinting mechanisms, and adjust storage policies accordingly. Policies also allow handling hardware with different performance characteristics. Currently the work is focused on implementing a fast and consistent mapping layer for the virtual block device. In the future, the authors plan to develop a library of policies for different applications and devices. Contact: Dai Quin (mike@eecg.toronto.edu)

Keynote Address

Disruptive Innovation: Data Domain Experience

Kai Li, Princeton University
Summarized by Rik Farrow

Kai Li told the story of Data Domain, a tiny company he founded that set out to replace the tape libraries used in data centers. They wanted to reduce the data footprint and network bandwidth by an order of magnitude, and did. What once required 17 tape libraries, a huge row of systems, became three 3U rack-mounted systems, in an example Li cited.

Li first asserted that innovation in large companies is very difficult, but he had a much more disturbing message for academics later in his keynote. He also said that you must have customer-driven technical development, work with the best venture capital firms, raise more money than you need, and hire the best people you can, even if you miss hiring goals. As for hiring people, Li stated the goal was to have people who work well together, minimizing egos, and using the best ideas. Li also said that some people demonized VCs, but good VCs helped them avoid many detours, and also helped with software design reviews and business plans.

Li presented a very interesting graph that compared income growth to lines of code. In the early years of Data Domain (2001-2007), they were producing 100,000 lines of production quality code every year, while growing the engineering team from ten to one hundred over this period. Li encouraged startups to stay focused, to carefully pick what features you code for—that is your roadmap.

In the early days, they had to find companies willing to install their product instead of tape libraries. Tape libraries are expensive, and that helped them have high margins, as the Data Domain hardware costs were low. And even though storage customers are very conservative and slow to change, they succeeded by having a product that worked. Li disparaged both trade shows and analyst groups, like Gardner, as a way to create new markets. Data Domain was successful long before analysts ever noticed the company.

Li pointed out that large companies like EMC, NetApp, and HP hopped on the data deduplication bandwagon early, but discontinued their efforts soon after. Except for NetApp, these larger companies eventually acquired small companies with successful deduplication, just as EMC acquired Data Domain.

As for reasons why big companies often fail, Li suggested that engineers can lack motivation because they feel ignored by the company, including lack of incentives (stock options). Another reason is that the process used in big companies can be very wrong: checking with lead customers and research firms, and having many meetings structured around PowerPoint graphics. Li said, "Microsoft has reduced the productivity of millions," a statement greeted with enthusiastic applause. Another reason is that established companies are afraid of killing their own children, their cash cows, with new products that will compete with them.

Finally, Li put the focus on academic research. Deduplication was not developed in a university. He and others left their positions to focus on their research, saying you can't both research and innovate. If you want to do a startup, you cross over, rather than stand in "two canoes, research and startup innovation."

Someone from EMC asked how often can you go from academia to a startup with no prior experience. Li replied that he is not saying your prior research has nothing to do with success. It's just that the skill set for making a product successful is not

taught in universities. You must put yourself into the market environment, and work to make your product successful. Margo Seltzer pointed out that Michael Stonebraker was another model of how this can work. Li agreed while pointing out that Stonebraker's current project (VoltDB) is already VC funded. Margo replied that Stonebraker said it is easier to get VC funding than research funding. How do we get a supportive systems research program going? Li had no answer. Someone asked whether following technical trends was a good idea, and Li laughed and said that it was a good question. He pointed out that we are moving away from spindles to flash memory, using forms of cloud to minimize the cost of running private DCs. But moving to the cloud for large companies will not work because of the cost of network bandwidth.

Keith Smith (NetApp) wondered why large companies struggle with innovation, and Li replied that there is just not enough innovation juice in large companies, and that little innovation has happened at Data Domain since it was acquired. Someone from EMC said that he was a researcher now, and Li countered by saying that Apple killed their research lab when Steve Jobs came back, and Amazon, Cisco, and EMC don't have research labs. Li cannot find the destructive type of product developed mainly due to researchers, as they are not exposed to learning the market. Li did have a small research lab at Princeton, which did make important contributions, including deduping data before network transmission. Randal Burns (John Hopkins) suggested SBIR (Small Business Innovation Research, sbir.gov) as an example of an attempt to extract innovation where it occurs in research. Li replied that SBIR is good and getting better, and that if there were a way for SBIR efforts to interact with many customers and team up with professionals, that would be great. Tech people are trained not to listen to other people, to believe "my idea is better and we know more than you," and after years of doing that, they lose the ability to hear what people want.

During his keynote, Li kept hinting that he had more to say, but wouldn't because his talk was being recorded (the video is available free at USENIX.org). As it was, Li's speech was both disruptive and enlightening.

Deduplication

Summarized by Min Li (limin@cs.vt.edu)

Concurrent Deletion in a Distributed Content-Addressable Storage System with Global Deduplication

Przemyslaw Strzelczak, Elzbieta Adamczyk, Urszula Herman-Izycka, Jakub Sakowicz, Lukasz Slusarczyk, Jaroslaw Wrona, and Cezary Dubnicki, 9LivesData, LLC

Strzelczak presented a deletion algorithm for a distributed content-addressable storage (CAS) system with global deduplication. Data deletion with deduplication enabled all the time is motivated by the fact the otherwise the storage consumption would be increased significantly because successive backups are usually similar. Strzelczak explained that data deletion with deduplication enabled was challenging because deduplication resulted in several owners of chunks, dynamic system changes such as adding/deleting nodes, and failures. The requirements of deletion are continuous system availability, no read-only period, negligible impact on user operations, scalability, and fault tolerance. He then discussed a simplified data model in a CAS storage system followed by the challenges for deletion in CAS.

The data model for a CAS storage system has been trees built bottom up sharing deduplicated blocks. Challenges lie in the root set determination and block resurrection through deduplication. Their deletion algorithm is comprised of two phases: garbage collection and space reclamation. Each deletion run proceeds in three subphases. More specifically, to solve the problem that a retention root is written to block A after deletion starts yet A is deleted mistakenly, they proposed to allow the counter to be increased between the first and the second advance. To deal with the problem of block A becoming a duplicate after deletion start or being deleted wrongly, they use an undelete marker to preserve deduplicated blocks. Strzelczak went on to discuss how they extend the algorithm to support distributed CAS systems. The main difficulty is to decide consistently whether to preserve or remove all fragments of a block. The solution they proposed is to leverage redundancy of computation from good peers, which have good enough data state and counter validation. When mismatches are found, the deletion would be aborted.

In terms of implementation, Strzelczak explained that they implemented the algorithm with a commercial system, HYDRAstor, which is designed for backup and archival data. Their evaluation showed that the deletion reduces performance less than 30% while using 30% system resources under the default configuration. When given minimum system resources, the deletion impacts performance within 5%.

Neville Carvalho (EMC) asked what size of block and of identifier were used. Strzelczak answered the chunk size in HYDRAstor is 64 KB and the block address has 20 bytes. Mark Lillibridge (HP Lab) asked what happens if you put an undelete marker on a block that is later going to be deleted. Strzelczak replied that if a counter was positive, the system did not do anything, but otherwise it knew the block should be deleted.

File Recipe Compression in Data Deduplication Systems

Dirk Meister, André Brinkmann, and Tim Süß, Johannes Gutenberg University, Mainz

Meister introduced the concept of file recipes, which consists of lists of fingerprints of variable-sized chunks belonging to a file. He pointed out that file recipes occupy increasingly significant disk capacity because chunk data grow with post-deduplication space whereas file recipes grow with pre-deduplication space. To reduce the storage usage, he proposed compressing the file recipes by leveraging shortened code words rather than the

fingerprint in the file recipe with low overhead in terms of memory, I/O, storage, and limited impact on write and restore speeds. He mentioned several assumptions: fingerprinting-based data deduplication systems, full chunk index availability, backup workloads, and reverse lookup necessity.

Next, Meister discussed three techniques used in their file recipe compression system. First, based on the observation that few chunks exhibit a high number of references such as zero-chunks, Meister proposed optimizing the case by using a one-byte code word, eliminating the need to store and look up the fingerprint. Second, they adopted a chunk index page-based approach to assign a code word to each fingerprint. In particular, the code word is assigned by the page ID and a unique identifier in the page. Third, they utilized statistical mechanisms which generalize zero-chunk suppression and assign shorter code words to fingerprints based on statics of the chunk usages. Meister went on to discuss the evaluation result. They used a trace-based simulation of weekly full backup. The figures he presented illustrated that their technique shrinks file recipes by more than 90%. He also concluded that file recipe allows additional storage saving, and it calls for exploration in storage deduplication research.

Michael Condict (NetApp ATG) asked whether they conducted experiments to reduce the average size of deduplication chunks since the compression of file recipes opens up opportunities to enable smaller size of chunks. Meister replied no, because this was not the only metadata overhead; as the size of chunks is reduced, the size of the chunk index increases and, for performance purposes, it was not quite special. Akshat Aranya (NEC Labs) asked whether they have the lookup table stored on SSD, mapping the compressed code words to the actual hash. Meister answered no, they did not need extra indexes; the code word itself consists of a page ID and unique identifier in a page, and can be used as the lookup keys, which is a nice property of this approach. Akshat then said he would follow up the question offline.

Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication

Mark Lillibridge and Kave Eshghi, HP Labs; Deepavali Bhagwat, HP Storage

Mark Lillibridge started by pointing out that the restore speed in chunk-based deduplication system gets slower over time due to worsening chunk fragmentation. Because chunks of backups get scattered around the whole system, restoration suffers when it has to jump back and forth between different chunk groups of different ages. "Why not just defragment data periodically like we did for the disks?" Mark asked. He mentioned two reasons. One was that there usually did not exist a chunk layout that reduces the fragmentation for all the backups. The other was that rearranging chunks required expensive data movement.

To deal with the problem, they investigated three techniques: increasing the cache size, using a forward assembly area, and container capping. Next, he explained that they measure fragmentation by using the mean number of containers read per MB of backup restored since that is proportional to the extent of chunk fragmentation. They also measured the restore speed to be the inverse of mean containers read per MB of data restored, which allowed them to focus on the dominant cost, container reading, ignoring noise and other factors. He next described how a baseline restoration algorithm works and highlighted the effect of cache size on restoration speed. A graph illustrated how restore speed is inversely proportional to the measure of fragmentation and how larger cache size yielded faster restoration speed. Another finding was that the increasing fragmentation levels result in unacceptable restoration speeds in emergencies.

Mark explained the forward assembly area approach they designed, which leverages the accurate knowledge from the backup recipe to perform better caching and prefetching and reduce the memory required during restoration. The method contained two variants, M-byte slices and rolling. M-byte slices control the amount of data to be assembled at one time in the forward assembly area that can be sent out in a single piece; rolling utilizes a ring buffer to effectively use memory to ensure that each container is loaded at most once every M bytes. He also showed an animation explaining how this technique works. Mark presented a chart showing how rolling effectively improves the speed factor compared with fixed case and LRU. An interesting point he mentioned was that given a backup workload, there would be sweet spots for LRU. Next, he switched to capping techniques, which are used to exploit the tradeoff between deduplication and faster restore speed. The basic idea is to bound the containers read per MB ingested. Using an animation, he explained how it worked. They first divide the backup streams into segments, such as 20 MB fixed size, read a segment into I/O buffer, then check which of the chucks are stored and in which containers. Next they choose up to T old containers to use, and finally they compute the recipe section for the segment and append any new chunks to the open container. The evaluation results he mentioned illustrate that the capping technique provided a good tradeoff between deduplication efficiency and restoration speed.

One attendee asked about the impact of capping on ingestion time and backup speed. Mark answered that it was not much, and actually might be faster. He then suggested the attendee go to the poster session and have a more detailed discussion with him. Geoff Kuenning asked about the order of containers in the assembly area, and Mark replied that you could use a variant of an elevator algorithm. Fred Douglis (EMC) wondered whether by focusing on read performance you would have a really large look-ahead buffer for the recipe. Mark answered that there are various data structures that you can use in walking the recipe in linear time to create backpointers.

Work-in-Progress Reports (WiPs)

Summarized by Thanh Do (thanhdo@cs.wisc.edu)

A Deduplication Study for Host-Side Caches with Dynamic Workloads in Virtualized Data Center Environments

Jingxin Feng and Jiri Schindler, NetApp Inc.

Jiri Schindler said that it is unclear whether host-side caches are effective for dynamic workloads, e.g., virtual machine (VM) migration, in virtual desktop infrastructure (VDI). For such workloads, re-warming the caches after VM migration may be costly; the caches may contain many copies of the same content because each VM disk image is a separate entity. This work analyzes real dynamic VDI workload traces to assess the deduplication opportunity for large host-side caches. The study finds that deduplication can reduce the data footprint inside the caches by as much as 67%. As a result, deduplication enables caching larger data sets and improving cache hit rates, therefore alleviating load from networked storage systems during I/O intensive workloads.

IBIS: Interposed Big-Data I/O Scheduler

Yiqi Xu, Adrian Suarez, and Ming Zhao, Florida International University

Yiqi Xu started his presentation with the problem of I/O scheduling in current big-data systems, such as Hadoop MapReduce. Such systems do not expose management of shared storage I/O resources, leading to potential performance degradation under high I/O contention among applications. To solve that problem, he proposed a new I/O scheduler framework, called IBIS, which provides performance differentiation for competing applications. Implemented in the Hadoop framework, IBIS schedules I/Os based on application bandwidth demands at individual data nodes as well as across distributed data nodes. Preliminary results showed the benefit of IBIS. Someone from HP Labs asked whether the framework considered network contention. Yiqi answered that network contention was not a concern because IBIS exploited data locality (i.e., task was likely scheduled in the same node where data was stored).

Adaptive Resource Allocation in Tiered Storage Systems

Hui Wang and Peter Varman, Rice University

Peter Varman explained the tradeoff between utilization and fairness in tiered storage systems, which are composed of SSD and disk arrays, with a simple example. The example showed that fairly allocating weights among clients with different hit ratios leads to non-optimized system utilization. Peter argued that a better allocation scheme would lead to better system utilization. To maximize system utilization, he proposed that weights for clients should be dynamically computed, based on their hit ratios. He showed some simulation results to prove that the proposed method helps to improve system utilization.

Trace Analysis for Block-Level Caching in Cloud Computing Systems

Dulcardo Arteaga and Ming Zhao, Florida International University; Pim Van Riezen and Lennard Zwart, Cloud VPS

The goal of this work is to assess the efficiency of using SSD caches in cloud systems. To that end, various traces from real-world private and public cloud systems are analyzed in order to answer key questions about the proper size of SDD caches and the caching policies that work best. The analysis shows some preliminary but interesting answers. For instance, I/O patterns vary across workloads; write-back cache is best for write-intensive workloads. Someone asked when the trace would be available. The answer was taken offline.

Radio+Tuner: A Tunable Distributed Object Store

Dorian J. Perkins, Curtis Yu, and Harsha V. Madhyastha, University of California, Riverside

Dorian Perkins started his presentation with a dilemma: there are no one-size-fits-all storage systems. As a result, for system administrators to choose the "right" systems for their workloads is hard. Furthermore, as new workloads emerge, new systems need to be built. To address this challenge, Dorian proposed Radio+Tuner. While Radio offers flexible storage configuration, Tuner picks the most cost-effective configuration for Radio, given input specification about cluster hardware, application workload, and performance SLO. Finally, he showed initial results to prove the benefit of Radio+Tuner. Someone asked whether Dorian assumed the underlying storage system was a black box. Dorian clarified that he built the system from scratch, meaning no black-box assumptions here. Another person asked how many nodes Radio+Tuner could scale to. Dorian answered that in his current prototype, there were 12 nodes in the system; to scale to many more nodes would require a more accurate algorithm.

JackRabbit: Improved Agility in Elastic Distributed Storage

James Cipar, Lianghong Xu, Elie Krevat, Alexey Tumanov, and Nitin Gupta, Carnegie Mellon University; Michael A. Kozuch, Intel Labs; Gregory R. Ganger, Carnegie Mellon University

Building an elastic storage system that has high performance, good fault tolerance, flexibility to shrink to a small fraction of servers, and the ability to quickly resize the system footprint (termed "agility") with minimal data migration overhead is hard. Rabbit, an elastic distributed system, provides good agility but has poor write performance. JackRabbit improves Rabbit with new policies for data placement, workload distribution, and data migration. For instance, JackRabbit takes read requests away from low numbered servers, which are bottlenecks for writes, to improve write throughput. These new policies allow JackRabbit to shrink to a small number of nodes while still maintaining performance goals. Preliminary results show these policies as beneficial.

High Performance & Low Latency in Solid-State Drives Through Redundancy

Dimitris Skourtis, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz

SSDs provide many benefits such as fast random access, but they also have problems. For instance, garbage collection in the background can degrade performance, especially in the case of mixed workloads. This work proposes a new design based on redundancy that provides consistent performance and minimal latency for reads by physically isolating reads from writes. The idea is to have a cache layer sitting on top of two SSDs, each of which serves reads or writes. One major challenge is to keep data "in sync" across two drives. Initial results are promising.

SATA Port Multipliers Considered Harmful

Peng Li, University of Minnesota; James Hughes and John Plocher, FutureWei Technologies; David J. Lilja, University of Minnesota

This work studies the reliability of SATA port multipliers (PMs) by proposing a reproducible process for creating actual failures in the HDDs. The authors conducted two experiments, one with the SATA PMs and one without them. In all experiments, a fatal HDD error was emulated by removing the HDD's cover. Experimental results showed that without SATA PMs, HDD failure was independent; however, at least one combination of the SATA controllers, the SATA PMs, and the HDDs did not provide resiliency when a single HDD failed. Investigating why this occurred was left for future work. Someone made a comment asking for another way to emulate fatal errors without destroying the disk, perhaps by putting a bullet through it.

Beyond MTTDL: A Closed-Form RAID 6 Reliability Equation

Jon Elerath and Jiri Schindler, NetApp Inc.

Jiri Schindler argued that although simple, the original RAID reliability equation that expressed mean-time-to-data loss (MTTDL) is no longer accurate, because today RAID systems are much more complex, with many processes for proactive scanning and repair of media defects. Moreover, researchers now have a better understanding of HDD failure modes and non-constant time-to-failure distributions. As a result, Jiri proposed a new equation that is more accurate, easy to use, easy to understand, and could help system designers to explore a variety of design points quickly. The new equation takes into account many factors, such as HDD operational failures, their restorations, latent (sector) defects, and disk media scrubbing. The great news was that this new equation is available online for anyone who wants to try it out at http://raideqn.netapp.com/. Finally, Jiri presented some results showing that the new equation is more accurate than the original. Someone asked whether the new equation models "wetware" (i.e., the human factor). Jiri answered that the model actually covers the human factor.

Reverse Deduplication: Optimizing for Fast Restore

Zhike Zhang, Preeti Gupta, Avani Wildani, Ignacio Corderi, and Darrell D.E. Long, University of California, Santa Cruz

Deduplicated storage systems suffer from data fragmentation, as more and more data are added and more data chunks are shared. Due to the nature of existing deduplication algorithms, the most recent backup is the most fragmented, resulting in performance issues. This work proposes to invert the deduplication process in order to make restoring the most recent copy more efficient. Specifically, new data segments will be written contiguously, and older data segments that share chunks in the new segments will reference those chunks; however, because older backups will develop more and more holes, restoring them would be costly. Preliminary results show that retrieving the most recent backup in reverse deduplication is more efficient than in traditional deduplication.

Quality-of-Data Consistency Levels in HBase for GeoReplication

Álvaro García Recuero, Instituto Superior Técnico; Luís Veiga, INESC-ID Lisboa, Distributed Systems Group

HBase only supports eventual consistency for replication between the local site and remote sites; updates are replicated asynchronously between datacenters. Thus, ensuring a given level of quality of service for delivering data to remote master replicas is challenging. This work extends some of the main components of HBase to replace the eventual consistency model with an adaptive consistency one. It outlines the architecture of a quality-of-service layer proposed for HBase.

Something for Everyone

Summarized by Dorian Perkins (dperkins@cs.ucr.edu)

Shroud: Ensuring Private Access to Large-Scale Data in the Data Center

Jacob R. Lorch, Bryan Parno, and James Mickens, Microsoft Research; Mariana Raykova, IBM Research; Joshua Schiffman, AMD

Jacob Lorch addressed the question: How can we prevent the cloud from learning our private data? Even when encryption is used, cloud services can still learn up to 80% of the content in email. This approach is based on previous work on oblivious RAM (ORAM), a technique used to obfuscate a client's access patterns to data; however, the authors note that ORAM is far too slow in practice. For example, a map application serving a single map tile to one user can take up to one week. Shroud leverages parallelism to speed up this technique while preserving privacy, reducing I/O time, and providing fault tolerance.

Overall, Shroud aims to fetch data from the cloud without the service knowing which block a user actually wants to access. Shroud uses trusted, secure coprocessors (smart cards that cost approximately $4 each) throughout the datacenter as proxies to each storage node. Users convey requests over secure channels with these proxies, which then access the data in-parallel. The coprocessors then employ a binary tree ORAM selection

technique to randomize the access patterns to data. Each time a block needs to be accessed, an adversary can only know how far down the tree the block may be, but has no idea where it actually is; subsequent access must use a different path to access the block. When a block is found, all coprocessors must send their blocks to the same node, which then uses a technique called oblivious aggregation to compute a collective XOR efficiently and securely. Jacob said that Shroud was deployed on 139 machines at MSR using emulated smart cards (due to availability), and was tested using various workloads, including Facebook images and map tiles.

An attendee asked how Shroud scaled. Jacob said that performance increases linearly until around 10K coprocessors, where performance gains begin to taper off. Jacob noted that Shroud is still more about theory than practice, as performance is still very slow, taking about 45 seconds to serve a map tile, and around nine seconds for serving a tweet. The clear performance bottleneck is the low-bandwidth smartcards they use as coprocessors, which only have around 12 KB/s bandwidth. The authors leave as future work employing high-bandwidth tamper-resistant FPGAs as coprocessors to improve performance, and admitting the hard drive to the trusted computing base to allow the use of more efficient ORAM protocols.

Getting Real: Lessons in Transitioning Research Simulations into Hardware Systems

Mohit Saxena, Yiying Zhang, Michael M. Swift, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Mohit Saxena noted there has been much work on SSD design, and common evaluation methods focus on three techniques: modifying the SSD by replacing the FTL (generally limited to vendors); FPGA prototyping, which is flexible and fast, yet hard and time-consuming; and simulators/emulators, which replay block traces and implement device models, and are generally used by the research community. Commonly, simulators are used when designing or evaluating new solid state drive (SSD) designs; however, the problem is that simulators cannot model real hardware accurately as they do not capture the complete interaction with the operating system in their model (e.g., timing dependencies, request alignment, etc.). Mohit pointed out that in the past three years, most papers have used simulators to validate their SSD designs. Instead, Mohit suggests a better approach using the OpenSSD hardware platform with a Jasmine board to develop new SSD designs. Yet the OpenSSD platform was not without issues, so Mohit explained how his team spent time optimizing this board to improve its performance.

Mohit discussed their prototyping experience with two previous works, Solid-State Cache (SSC) and Nameless-Writes (NW-SSD), noting the challenges, solutions, and lessons learned from optimizing their hardware design and evaluation suite for flash storage devices. SSC aims to improve performance compared to using an SSD as block cache, while NW-SSD introduces new commands to build cheap and fast SSDs, by exposing the flash block layout and interacting directly with the OS when serving reads and writes. Mohit separated his prototyping experience into three sections: new forward commands, new device responses, and real hardware constraints for SSC and NW-SSD. Mohit summarized each of his points with lessons learned. When designing new forward commands, Mohit urges that you should always consider all layers of the OS, especially I/O schedulers merging and re-ordering operations, and also consider the complete SSD ecosystem, including encoding sub-types and accelerating new command queues. Designers of new device responses should again consider all OS layers, such as race conditions for callbacks in the device and file system, and handling of frequent benign errors in the storage device drivers. A prototyping lesson also learned here is simplicity and correctness; make the Kernel-FTL a simpler block layer OS interface and enforce correct erase-before-overwrite operations in the Device-FTL.

In their performance evaluation, they compared their two systems (SSC and NW-SSD) with a bare SSD using filebench. They validated the previous performance claims of SSC (168% better than common hybrid FTL) design by showing that it performs 52% better than a faster page-map FTL and that NW-SSD can substantially reduce the amount of device memory required with performance close to a page-map FTL. In conclusion, Mohit found that OpenSSD is a valuable tool for evaluating new SSD designs. Mohit shared his first high-performance open-source FTL at http://cs.wisc.edu/~msaxena/new/ftl.html.

To Zip or Not to Zip: Effective Resource Usage for Real-Time Compression

Danny Harnik, Ronen Kat, Oded Margalit, Dmitry Sotnikov, and Avishay Traeger, IBM Research—Haifa

Danny Harnik lightheartedly began his talk by asking, "To zip, or not to zip, that is question." Danny introduced his work with the motivating goal of reducing time, cost, rackspace, and cooling requirements. The challenge of this work is to add "seamless" compression to a storage system with little effect on performance. Danny noted paying the compression overhead is okay if you are going to gain something, but it is not always worth the effort. The authors' goal is to avoid compressing "incompressible" data, while also maximizing the compression ratio of their stored data. To tackle this problem, Danny noted that there is no established, accurate method for estimating compression ratio, outside of actually compressing the data. Other solutions included deducing from empirical application data or file extensions, but these are neither accurate nor always available.

Danny explained how the authors tackled this problem from a micro and macro scale. For macro, they considered a large multi-GB/TB volume in which the time to compress was on the order of hours, where estimates only took a short time (minutes) and they

could actually obtain accuracy guarantees. At this scale, they choose M random locations to test local compression ratios and compute an average compression ratio for these locations; however, he noted that in practice, this straightforward sampling method can have issues of compression locality. So they tweaked their method to evaluate the "compression contribution" of single bytes from regions throughout the file, and define the contribution of a byte as the compression ratio of its locality region (where locality depends on the specific compression method at hand). They then proved through statistical analysis for estimating averages that their method estimated the overall ratio with guaranteed accuracy (the actual parameters depend on the sample size but do not depend on the volume size). This macro-scale compression estimate is also useful as an evaluation and sizing tool. A version of this tool can by found by searching "IBM Comprestimator" or at this link: http://www-01.ibm.com/support/docview.wss?uid=ssg1S4001012.

Danny noted that at the micro-scale, they consider single write, KB-sized files, which take milliseconds to compress; since estimation has to be ultra-quick, they rely on heuristics. Danny pointed out that getting guarantees is impossible as the locality in this case amounts to the entire chunk. They considered two approaches: a prefix-based estimation and a heuristic indicator method. In the latter, they collect some basic indicators about the data and output a recommendation accordingly. For this method they employ a number of techniques to improve the time performance of the estimation. Danny discussed the performance evaluation of the two estimation methods on more than 300 GB (17,790 files) of mixed data types, showing that the heuristics approach wins out over the 1 KB prefix sampling, and both improve on the option of running a full compression on an 8 KB chunk. In a time versus compression tradeoff analysis, prefix compression has 74% CPU utilization with 2.2% capacity overhead, while the heuristics method has 65% CPU utilization at a nominally higher 2.3% capacity overhead.

In summary, Danny concluded that when most data is compressible use prefix estimation, when a significant percentage is incompressible use the heuristics method, and when most is incompressible, turn off compression and run macro-scale offline to detect a change. Michael Condit (NetApp) noted that other compression techniques are faster than the one studied in the paper, and this work depends on the compression algorithm's latency. Danny replied that the work generalizes to other methods as well, but may be less relevant to some. For example, Snappy is a compression algorithm that already uses prefix estimation.

Poster Session and Reception II

Summarized by Matias Bjorling (mabj@itu.dk)

Examining Scientific Data for Scalable Index Designs

Aleatha Parker-Wood, Brian A. Madden, Michael McThrow, and Darrell D.E. Long, University of California, Santa Cruz

Aleatha Parker-Wood argued that modern file systems with billions of files are no longer tractable for conducting searches of scientific data. The vast amount of data and ever larger metadata, describing scientific observations, has become unmanageable. They argue that databases optimized for sparse data, column-based compression, and high cardinality are a better choice as a file-system index database. They evaluated five architectures: row stores, column stores, key-value stores, document stores, and spatial trees and compared each in regard to sparseness, dimensions, cardinality, specialized access, and ad hoc queries. They found column and document stores to be efficient structures for the scientific metadata. Further investigations include novel indexing strategies, such as on-demand indexing on a per-column basis.

Reliability Analysis of Distributed RAID with Priority Rebuilding

Hiroaki Akutsu and Tomohiro Kawaguchi, Yokohama Research Laboratory, Hitachi, Ltd.

The storage capacity of hard drives has been increasing exponentially, leading to longer RAID rebuild times and increased risk of data loss. Distributed RAID is a technique to decrease the rebuild time. Because of the expanded rebuild range, more drives are prone to fault during rebuilding. Priority rebuilding is used to restore data with the lowest redundancy first. To estimate the redundancy reliability, Hiroaki Akutsu presented a reliability analysis of distributed RAIDs that they can use as a model. They found that distributed RAID reliability is roughly equal to that of a level-1 redundancy method (e.g., mirroring, RAID5); reliability becomes roughly constant, independent of the number of drives in a level-2 redundancy method (e.g., triplication, RAID6); and reliability increased due to the increase in the number of drives in the over level-3 redundancy method (e.g., triple parity RAID, high-redundancy erasure-coding).

Radio+Tuner: A Tunable Distributed Object Store

Dorian J. Perkins, Curtis Yu, and Harsha V. Madhyastha, University of California, Riverside

There are many storage systems today, each designed with its own specific workload and performance goals; however, no single distributed storage system design is cost-optimal for meeting performance goals of all workloads. Dorian Perkins presented Radio+Tuner, a tunable distributed object store (Radio) and its configuration engine (Tuner). Radio offers a simple GET/PUT interface, with three system components: NodeMetadataStore, DiskMetadataStore, and DataStore. Each component offers multiple implementations allowing for "mix-and-match" configurations, with the benefits that as new workloads emerge,

new implementations may be added to the system (instead of designing a new system). Tuner takes as input the workload's parameters and performance SLOs, as well as hardware and component implementation specifications, and simulates the operation of Radio to obtain a GET/PUT latency distribution. Tuner then outputs the lowest cost configuration that meets the workloads goals. Initial results show that Radio+Tuner is able to adapt to disparate workloads, and does so at up to 5x cost savings when using the Tuner-recommended configurations. Future work includes unifying Radio with prior solutions that consider consistency and availability requirements, and expanding Radio to handle multiple conflicting workloads on the same hardware.

JackRabbit: Improved Agility in Elastic Distributed Storage

James Cipar, Lianghong Xu, Elie Krevat, Alexey Tumanov, and Nitin Gupta, Carnegie Mellon University; Michael A. Kozuch, Intel Labs; Gregory R. Ganger, Carnegie Mellon University

Distributed storage is often expensive to scale and requires aggressive write periods when new nodes are added or removed. Recent research in elastic storage systems, such as Rabbit and Sierra, enable better elasticity by new data layouts and mechanisms, but both suffer from write degradation or poor agility. Lianghong Xu presented JackRabbit. It focuses on new policies, designed to maximize the agility of elastic storage, while accommodating both performance and fault tolerance. Evaluation shows that JackRabbit comes closer to the ideal machine hour elasticity (within 4%) and improves over state-of-the-art elastic storage systems by 6–120%.

High Performance & Low Latency in Solid-State Drives Through Redundancy

Dimitris Skourtis, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz

Dimitris Skourtis presented an approach to having both high performance and low latency in solid-state drives using redundancy. By separating read and write patterns, only one drive is being written at a time. Thus, the other drive is solely available for reads. After a variable amount of time, the disk responsibility is switched. The to-be-written data is cached and then flushed. The evaluation shows reads have consistently less variation and double throughput for 256 KB blocks. Future work includes quality of service for mixed workloads and evaluation under live workloads, such as databases and VMs.

A Deduplication Study for Host-Side Caches with Dynamic Workloads in Virtualized Data Center Environments

Jingxin Feng and Jiri Schindler, NetApp Inc.

Jiri Schindler presented a deduplication study of host-side caches in virtualized datacenter environments. Host-side caches can be rather large, and re-warming the cache for migrated virtual machines may take up to several hours. In virtual desktop infrastructure (VDI) deployments, a virtual machine is a separate entity, but the host-side cache might contain many copies of the same content, even though the network-attached shared storage system would only store a single instance. The goal of their study is to explore the effectiveness of deduplication for large host-side caches running dynamic VDI workloads. Their preliminary results show a disk space saving of 54% to 67% using deduplication and a larger saving if reads and writes are observed separately. They argue that the "deduplication degree" metric captures a useful concept for evaluating cache effectiveness for dynamic workloads. Future work includes analyzing similarity of VDI traffic, deduplication sensitivity to cache block size, and other aspects that can improve the host-side cache in VDI environments.

Summarized by Jorge E. Cabrera (jcabr020@cs.fiu.edu)

Adaptive Resource Allocation in Tiered Storage Systems

Hui Wang, Peter Varman, Rice University

Peter Varman addressed the challenge of providing both fairness and high utilization guarantees in multi-tiered storage systems. Their work is centered around dynamically computing a reservation and limit value for all clients based on their hit ratio. These values are used to guarantee fairness by providing a minimum allocation, and to provide remaining I/Os to other clients to obtain maximum system utilization. Evaluation results using a process-driven simulator show that their allocation model can potentially pull up utilization to maximum throughput or close to it depending on the reservation values of all the clients. Future work entails extending the allocation model to include relative shares.

Quality-of-Data Consistency Levels in HBase for GeoReplication

Álvaro García Recuero, Instituto Superior Técnico; Luís Veiga, INESC-ID Lisboa, Distributed Systems Group

A major challenge in cloud storage systems is providing quality-of-data consistency levels for data-replication mechanisms. Alvaro García Recuero presented a mechanism to extend the replication mechanisms of HBase, an open-source version of BigTable. Currently, HBase uses a best-effort delivery mechanism of data by using an eventual consistency mode. The proposed approach is to leverage the vector field consistency model into a framework that provides the HBase core with a QoD layer that allows it to prioritize specific client replicas to deliver replica updates with the agreed quality of data. Current evaluation is pending, and expected results promise a reduction in bandwidth usage and more control of the interval when replication occurs.

IBIS: Interposed Big-Data I/O Scheduler

Yiqi Xu, Adrian Suarez, and Ming Zhao, Florida International University

Yiqi Xu presented IBIS (Interposed Big-data I/O Scheduler), which tries to solve the scheduling problem that exists in big-data systems (e.g., Hadoop/MapReduce) because they do not expose management of shared storage I/O resources. As such, an application's performance may degrade in unpredictable

ways under I/O contention, even with fair sharing of computing resources. IBIS provides performance differentiation for the I/Os of competing applications in a shared MapReduce-type big-data system. IBIS is implemented in Hadoop by interposing HDFS I/Os and use an SFQ-based proportional-sharing algorithm. Experiments show that IBIS provides strong performance isolation for one application against another highly I/O-intensive application. IBIS also enforces good proportional sharing of the global bandwidth among competing parallel applications, by coordinating distributed IBIS schedulers to deal with the uneven distribution of local services in big-data systems.

Trace Analysis for Block-Level Caching in Cloud Computing Systems

Dulcardo Arteaga and Ming Zhao, Florida International University; Pim Van Riezen and Lennard Zwart, Cloud VPS

Client-side caching by using SSDs can potentially improve the performance of shared block-level storage systems that can suffer from scalability issues when the number of clients grows. Dulcardo Arteaga presented a trace analysis for this type of caching with the goal of analyzing the effective use of SSD devices as caches. Specifically, there are three factors that are studied: size of SSD device through working set size analysis, a comparison of three caching policy configurations, and dynamic and static allocation of shared caches among concurrent VM clients. The types of traces analyzed include both public and private cloud environments comprising Web servers, file servers, and VM clients. The types of caching policies used are write-back, write-through, and write-allocate. Some of the interesting results show that both public and private clouds have an average cache hit ratio of 74% and 78%, respectively, using write-back policy. Additionally, working set sizes can be accurately predicted 90%of the time.

Beyond MTTDL: A Closed-Form RAID 6 Reliability Equation

Jon Elerath and Jiri Schindler, NetApp Inc.

The complexity of RAID systems and new HDD technologies has risen to a level where old MTTDL models cannot be applied to obtain accurate results. New systems have improved designs that employ repair mechanisms that did not exist in older HDDs. Jiri Schindler presented a project based on developing a more accurate and reliable MTTDL equation model, specifically for RAID6 setups. The result of this research is a new closed-form RAID6 reliability equation that can better model data-loss events compared to the old MTTDL equation. This equation can yield estimations for HDD operational failures, latent defects, and disk media scrubbing. The equation was formulated by using a two-parameter Weibull distribution using parameters obtained from real-world data. The equation was verified against a Monte Carlo model simulation, and the results shows similar accuracy. Additionally, the new MTTDL equation can yield results in milliseconds, whereas a single MC simulation ran between 14 seconds and 18 hours. A Javascript implementation of the model is available for the public at http://raideqn.netapp.com. Evaluation results show that in comparison to the old model, the new equation shows more realistic results when it comes to predicting the occurrence of failures.

Reverse Deduplication: Optimizing for Fast Restore

Zhike Zhang, Preeti Gupta, Avani Wildani, Ignacio Corderi, and Darrell D.E. Long, University of California, Santa Cruz

Preeti Gupta explained that as the number of shared data chunks increases, the amount of data fragmentation increases and can lead to decreased performance in deduplicated storage systems. In particular, the most recent backup is the most fragmented of this data. The goal of this project is improve the performance access of the most recent backup in deduplicated backup systems. The proposed approach entails the inversion of the deduplication process. Instead of mapping new chunks to already existing chunks, each new data segment is written contiguously, and older data is mapped to the new chunks. Evaluation results show that they can significantly reduce fragmentation for the most recent data segments. Specifically, retrieval time can be 4 to 19 times faster. While this solution is great for the most recent backup, it does pose a tradeoff for accessing older backups, which develop portions of data that are no longer referenced.

Flash and SSDs

Summarized by Leonardo Marmol (marmol@cs.fiu.edu)

LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives

Kai Zhao, Rensselaer Polytechnic Institute; Wenzhe Zhao and Hongbin Sun, Xi'an Jiaotong University; Tong Zhang, Rensselaer Polytechnic Institute; Xiaodong Zhang, The Ohio State University; Nanning Zheng, Xi'an Jiaotong University

Current SSDs use Bose-Chaudhuri-Hocquengham (BCH) error correction mechanisms. However, as NAND flash technology becomes denser it also becomes less reliable, rendering BCH incapable of dealing with the several types of interference present in NAND. As an alternative, Kai Zhao proposed the use of low-density parity-check (LDPC) techniques and explores its potential and limitations. LDPC is known to provide stronger error correction capabilities, but the performance penalty associated with LDPC has made it impractical so far. Zhao addressed these limitations with a technique that combines the simplicity and high speed of hard-decision coding with the strong error correction capability of soft-decision coding.

At a high level, the idea is to use hard-decision at first and only apply a soft-decision in the presence of failures. By combining look-ahead memory sensing to reduce the total latency, fine-grained progressive sensing and decoding as needed, and smart data placement interleaving, Zhao et al. managed to provide a solution that significantly reduced the average response time delay while still providing high reliability for dense flash technologies.

The implementation was evaluated using the DiskSim simulator and six sets of traces of different workloads. The experimental work flow consisted of running a high number of program/erase cycles followed by a baking session to emulate the wear-out recovery. The baking time was determined using Arrhenius's Law. Next, random data is programmed into the chips, and these are baked once again to emulate one month retention time. Finally, the data is read and compared with the original data to get page error statistics. The results showed a comparison between the proposed techniques and the basic two-step sensing process. In general, the combined use of look-ahead, progressive sensing, and interleaving lead to a reduction of response time delay from more than 100% to less than 20%.

Joseph Tucek (HP Labs) asked how his solution would play with RAID systems with their own built-in error correction mechanisms. Zhao replied that having the upper layer doing error correction is an orthogonal solution that in most cases will not suffice. Peter Harllee (CMU) asked whether the error information was used to redefine new voltage thresholds. Zhao answered that it can only be done at the word granularity. On a related note, someone asked about the possibility of providing hints to the decoder to avoid interleaving pages of different qualities.

Extending the Lifetime of Flash-Based Storage Through Reducing Write Amplification from File Systems

Youyou Lu, Jiwu Shu, and Weimin Zheng, Tsinghua University

Youyou Lu explained how the increased density of flash memory also made it less tolerant to leakage and noise interference, taking a toll on the reliability and lifetime of flash memory. He pointed out that traditional file systems were developed assuming the use of hard disks and not flash, the reason for which common mechanisms such as journaling, metadata synchronization, and page-aligned update can induce extra write operations that further reduce the lifetime of flash. Lu proposed an object-based flash translation layer design (OFTL) that makes file systems no longer responsible for storage management and exports a byte-unit access interface to them. This decoupling allows the OFTL to lazily update metadata indexes and eliminates journals without losing any consistency properties by making use of the page metadata area. Further, OFTL makes it possible for coarse-grained block state maintenance to reduce free management overheads using units of erase blocks rather than file system blocks. Finally, the byte-unit interface allows OFTL to compact and better co-locate small updates, reducing the total number of updates and amortizing the cost of page writes across unaligned pages.

The system was evaluated with several workloads and traces and implemented as a Linux kernel module. For every workload, Lu et al. measured the write amplifications—defined as the total size or number of writes to the flash memory divided by the total size or number of writes issued from the application layer—across several file systems, including ext2, ext3, Btrfs and their OFTL implementation. The results showed that the OFTL-based system offers a write amplification reduction of 47% to 90% with synchronous workloads and 20 % to 64% with asynchronous workloads. Richard Spillane (Apple) asked why sequential workloads were causing so much write amplification in one of the experiments. Lu explained that data is duplicated, once for the journal and again for the actual write.

Understanding the Robustness of SSDs under Power Fault

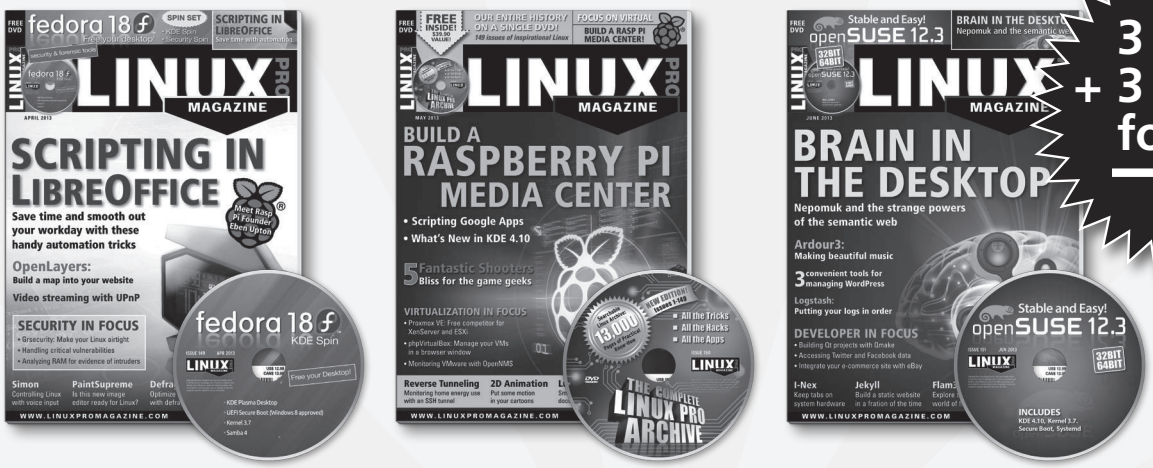
Mai Zheng, The Ohio State University; Joseph Tucek, HP Labs; Feng Qin, The Ohio State University; Mark Lillibridge, HP Labs

Mai Zheng talked about the wide adoption of SSDs due to their many good qualities; however, little has been said about the behavior of SSDs under adverse conditions. In particular, Zheng studied the behavior of SSDs under power failures. Among the potential types of failures a SSD can experience, Zheng listed bit corruption, metadata corruption, inconsistency in the internal state of the device, shorn and flying writes, and unserializable writes.

To test the resilience of SSDs under power failures, Zheng et al. created a testing framework made of four main components: scheduler, workers, switcher, and checker. Each test cycle consisted of three stages. Initially, the workers stress the SSD with many write operations, ideally making the device as vulnerable as possible by triggering wear leveling and garbage collection. Next, the device's power is cut off asynchronously by a circuit controlling the SSD's dedicated power source. Finally, the checker reads back the data and checks for potential failures. The data written to the device is carefully arranged and contains enough metadata to uniquely identify all types of errors and other possible interferences such as dynamic data compression by the SSD.

Fifteen SSDs were subjected to a series of controlled power failures, and 13 exhibited some form of failure. Unserializable writes were the most common type of failure, but all other types of failures were also found. One device exhibited metadata corruption after only eight injected faults, which caused 72 GB of data to be lost. Other devices were rendered undetectable by the host after 136 injected faults. Zheng was asked what types of errors he found when the devices were not stressed, which was not considered in the evaluation. John Badger (Quantum) asked about the devices that claimed to have some form of power failure protection. Zheng said three out of four failed. Fred Glover (Oracle) asked whether they tried cutting the AC power supply instead of DC. Zheng said they didn't, as automating and performing many power cycles by cutting the AC is not easy. Bill Bilofsky (Microsoft) asked whether they tried cutting off power for a short period of time and whether the error happened during powering off or powering up the device. Zheng said that a quick power restore was not part of the evaluation and the experimental setup did not provide enough insight to determine exactly when failures took place.

RISK-FREE TRIAL!



3 issues + 3 DVDs for only $3

PRACTICAL. PROFESSIONAL. ELEGANT.

Enjoy a rich blend of tutorials, reviews, international news, and practical solutions for the technical reader.

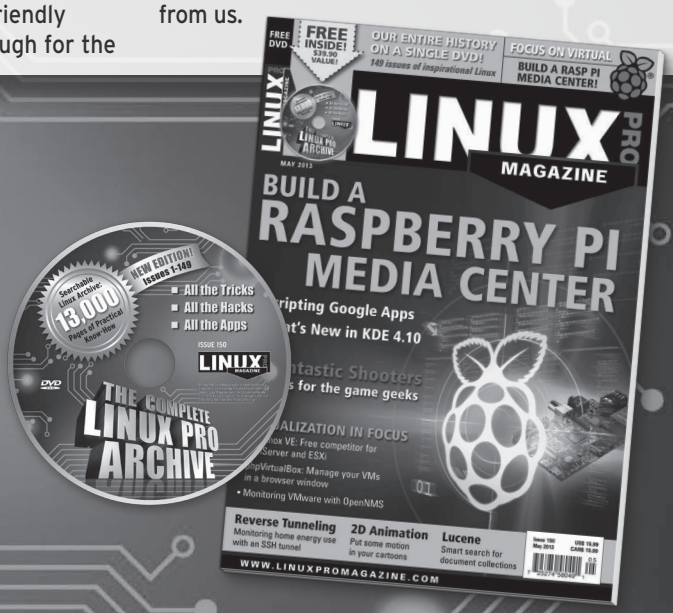
ORDER AT: shop.linuxnewmedia.com

Order Linux Pro Magazine issue 150

and receive our entire history on 1 searchable DVD!

For more than 12 years, Linux Magazine has specialized in smart, insightful articles on open source tools and real-world Linux. We've charted the history of Linux from an obscure hacker's system to a mainstream OS that is friendly enough for beginners and stable enough for the corporate desktop. Now all that knowledge is available in a single, searchable DVD archive. The Complete Linux Magazine Archive includes all the technical heavy lifting you've come to expect from us.

THE COMPLETE LINUX PRO ARCHIVE



usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES