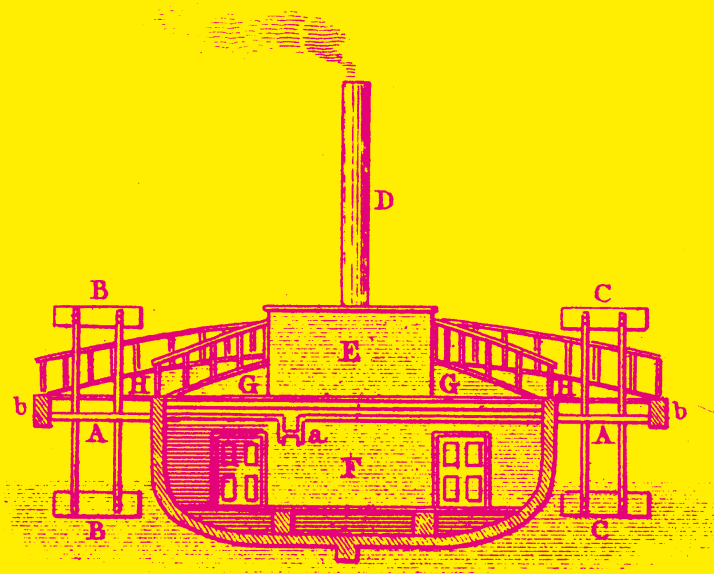




THE USENIX MAGAZINE



22

**OPINION**

Musings 2  
**RIK FARROW**

**SYSADMIN**

Data Corruption in the Storage Stack:  
 A Closer Look 6

**LAKSHMI BAIRAVASUNDARAM,  
 GARTH GOODSON, BIANCA SCHROEDER,  
 ANDREA ARPACI-DUSSEAU, AND REMZI  
 ARPACI-DUSSEAU**

Pergamum: Energy-efficient Archival Storage  
 with Disk Instead of Tape 15

**MARK W. STORER, KEVIN M. GREENAN,  
 ETHAN L. MILLER, AND KALADHAR VORUGANTI**

Don't Blame Disks for Every Storage Subsystem  
 Failure 22

**WEIHANG JIANG, CHONGFENG HU, YUANYUAN  
 ZHOU, AND ARKADY KANEVSKY**

TierStore: A Distributed File System for  
 Challenged Networks in Developing Regions 32

**MICHAEL DEMMER, BOWEI DU, AND  
 ERIC BREWER**

The Present and Future of SAN/NAS: Interview  
 with Dave Hitz and Brian Pawlowsky of  
 NetApp 39

**INTERVIEW BY MARGO SELTZER**

**PROGRAMMING**

Driving the Evolution of Software Languages  
 to a Concurrent Future 45

**ANDREW BROWNSWORD**

The Murky Issue of Changing Process  
 Identity: Revising "Setuid Demystified" 55

**DAN TSAFRIR, DILMA DA SILVA, AND  
 DAVID WAGNER**

**COLUMNS**

Practical Perl Tools: A Little Place for  
 Your Stuff 67

**DAVID N. BLANK-EDELMAN**

Pete's All Things Sun (PATS):  
 The State of ZFS 72

**PETER BAER GALVIN**

iVoyeur: Admin, Root Thyself. 77

**DAVID JOSEPHSEN**

/dev/random 83

**ROBERT G. FERRELL**

**STANDARDS**

Update on Standards: Undue Influence? 85

**NICK STOUGHTON**

**BOOK REVIEWS**

Book Reviews 88

**ELIZABETH ZWICKY ET AL.**

**USENIX NOTES**

Election Results 91

Notice of Annual Meeting 91

**CONFERENCES**

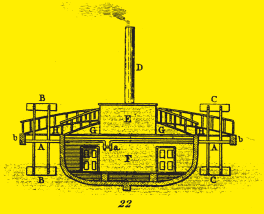
FAST 'o8 Reports 93

LSF 'o8 Reports 107

**USENIX**

The Advanced Computing  
 Systems Association

# USENIX Upcoming Events



---

## THE SIXTH INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS, AND SERVICES (MOBISYS 2008)

Jointly sponsored by ACM SIGMOBILE and USENIX  
JUNE 17–20, 2008, BRECKENRIDGE, CO, USA  
<http://www.sigmobile.org/mobisys/2008/>

---

## 2008 USENIX ANNUAL TECHNICAL CONFERENCE

JUNE 22–27, 2008, BOSTON, MA, USA  
<http://www.usenix.org/usenix08>

---

## FIRST USENIX WORKSHOP ON LARGE-SCALE COMPUTING (LASCO '08)

Co-located with USENIX '08  
JUNE 23, 2008, BOSTON, MA, USA  
<http://www.usenix.org/lasco08>

---

## XEN SUMMIT NORTH AMERICA 2008

Co-located with USENIX '08  
JUNE 23–24, 2008, BOSTON, MA, USA  
<http://xen.org/xensummit/>

---

## 2ND INTERNATIONAL CONFERENCE ON DISTRIBUTED EVENT-BASED SYSTEMS (DEBS 2008)

Organized in cooperation with USENIX, the IEEE and IEEE Computer Society, ACM SIGSOFT, and ACM SIGMOD  
JULY 2–4, 2008, ROME, ITALY  
<http://debs08.dis.uniroma1.it/>

---

## 2008 USENIX/ACCURATE ELECTRONIC VOTING TECHNOLOGY WORKSHOP (EVT '08)

Co-located with USENIX Security '08  
JULY 28–29, 2008, SAN JOSE, CA, USA  
<http://www.usenix.org/evt08>

---

## 2ND USENIX WORKSHOP ON OFFENSIVE TECHNOLOGIES (WOOT '08)

Co-located with USENIX Security '08  
JULY 28, 2008, SAN JOSE, CA, USA  
<http://www.usenix.org/woot08>  
Submissions due: June 1, 2008

---

## WORKSHOP ON CYBER SECURITY EXPERIMENTATION AND TEST (CSET '08)

Co-located with USENIX Security '08  
JULY 28, 2008, SAN JOSE, CA, USA  
<http://www.usenix.org/cset08>

---

## 17TH USENIX SECURITY SYMPOSIUM

JULY 28–AUGUST 1, 2008, SAN JOSE, CA, USA  
<http://www.usenix.org/sec08>

---

## 3RD USENIX WORKSHOP ON HOT TOPICS IN SECURITY (HOTSEC '08)

Co-located with USENIX Security '08  
JULY 29, 2008, SAN JOSE, CA, USA  
<http://www.usenix.org/hotsec08>

---

## THIRD WORKSHOP ON SECURITY METRICS (METRICON 3.0)

Co-located with USENIX Security '08  
JULY 29, 2008, SAN JOSE, CA, USA  
<http://www.securitymetrics.org/content/Wiki.jsp?page=Metricon3.0>

---

## 22ND LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE (LISA '08)

Sponsored by USENIX and SAGE  
NOVEMBER 9–14, 2008, SAN DIEGO, CA, USA  
<http://www.usenix.org/lisa08>

---

## SYMPOSIUM ON COMPUTER HUMAN INTERACTION FOR MANAGEMENT OF INFORMATION TECHNOLOGY (CHIMIT '08)

Sponsored by ACM in association with USENIX  
NOVEMBER 14–15, 2008, SAN DIEGO, CA, USA  
<http://www.chimit08.org>

---

## ACM/IFIP/USENIX 9TH INTERNATIONAL MIDDLEWARE CONFERENCE (MIDDLEWARE 2008)

DECEMBER 1–5, 2008, LEUVEN, BELGIUM  
<http://middleware2008.cs.kuleuven.be>

For a complete list of all USENIX & USENIX co-sponsored events, see <http://www.usenix.org/events>.

# contents



**VOL. 33, #3, JUNE 2008**

**EDITOR**  
Rik Farrow  
rik@usenix.org

**MANAGING EDITOR**  
Jane-Ellen Long  
jel@usenix.org

**COPY EDITOR**  
David Couzens  
proofshop@usenix.org

**PRODUCTION**  
Casey Henderson  
Jane-Ellen Long  
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  
  
<http://www.usenix.org>  
<http://www.sage.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 an annual subscription to ;login:. Subscriptions for nonmembers are \$120 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.

©2008 USENIX Association

USENIX is a registered trademark of the USENIX Association. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. USENIX acknowledges all trademarks herein. Where those designations appear in this publication and USENIX is aware of a trademark claim, the designations have been printed in caps or initial caps.

## OPINION

Musings 2  
**RIK FARROW**

## SYSADMIN

Data Corruption in the Storage Stack: A Closer Look 6  
**LAKSHMI BAIRAVASUNDARAM, GARTH GOODSON, BIANCA SCHROEDER, ANDREA ARPACI-DUSSEAU, AND REMZI ARPACI-DUSSEAU**

Pergamum: Energy-efficient Archival Storage with Disk Instead of Tape 15  
**MARK W. STORER, KEVIN M. GREENAN, ETHAN L. MILLER, AND KALADHAR VORUGANTI**

Don't Blame Disks for Every Storage Subsystem Failure 22  
**WEIHANG JIANG, CHONGFENG HU, YUANYUAN ZHOU, AND ARKADY KANEVSKY**

TierStore: A Distributed File System for Challenged Networks in Developing Regions 32  
**MICHAEL DEMMER, BOWEI DU, AND ERIC BREWER**

The Present and Future of SAN/NAS: Interview with Dave Hitz and Brian Pawlowsky of NetApp 39  
**INTERVIEW BY MARGO SELTZER**

## PROGRAMMING

Driving the Evolution of Software Languages to a Concurrent Future 45  
**ANDREW BROWNSWORD**

The Murky Issue of Changing Process Identity: Revising "Setuid Demystified" 55  
**DAN TSAFRIR, DILMA DA SILVA, AND DAVID WAGNER**

## COLUMNS

Practical Perl Tools: A Little Place for Your Stuff 67  
**DAVID N. BLANK-EDELMAN**

Pete's All Things Sun (PATS): The State of ZFS 72  
**PETER BAER GALVIN**

iVoyeur: Admin, Root Thyself. 77  
**DAVID JOSEPHSEN**

/dev/random 83  
**ROBERT G. FERRELL**

## STANDARDS

Update on Standards: Undue Influence? 85  
**NICK STOUGHTON**

## BOOK REVIEWS

Book Reviews 88  
**ELIZABETH ZWICKY ET AL.**

## USENIX NOTES

Election Results 91  
Notice of Annual Meeting 91

## CONFERENCES

FAST '08 Reports 93  
LSF '08 Reports 107

RIK FARROW

## musings

[rik@usenix.org](mailto:rik@usenix.org)



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

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

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

### Buggy Hardware

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

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

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

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

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

---

## Tomes

---

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

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

---

## Sluggish Memory

---

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

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

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

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

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

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

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

---

## In This Issue

---

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

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

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

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

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

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

LAKSHMI BAIRAVASUNDARAM,  
GARTH GOODSON, BIANCA SCHROEDER,  
ANDREA ARPACI-DUSSEAU, AND  
REMZI ARPACI-DUSSEAU

## data corruption in the storage stack: a closer look



Lakshmi N. Bairavasundaram is a PhD student in Computer Sciences at the University of Wisconsin, Madison, working with advisors Prof. Andrea Arpaci-Dusseau and Prof. Remzi Arpaci-Dusseau. He received his BE from Anna University, India, and his MS from the University of Wisconsin, Madison.

[laksh@cs.wisc.edu](mailto:laksh@cs.wisc.edu)



Garth Goodson is a researcher at NetApp, Inc., with interests in virtualization, distributed systems, and new memory technologies. He received his PhD in 2004 from Carnegie Mellon University under the supervision of Greg Ganger.

[Garth.Goodson@netapp.com](mailto:Garth.Goodson@netapp.com)



Bianca Schroeder is an Assistant Professor in the Department of Computer Science at the University of Toronto. Before coming to Toronto, Bianca completed her PhD and a two-year postdoc at Carnegie Mellon University. Her research focuses on computer systems and has earned her multiple best paper awards.

[bianca@cs.toronto.edu](mailto:bianca@cs.toronto.edu)



Andrea Arpaci-Dusseau is an associate professor of Computer Sciences at the University of Wisconsin, Madison. She received her BS from Carnegie Mellon University and her MS and PhD from the University of California, Berkeley, under advisor David Culler.

[dusseau@cs.wisc.edu](mailto:dusseau@cs.wisc.edu)



Remzi Arpaci-Dusseau is an associate professor of Computer Sciences at the University of Wisconsin, Madison. He received his BS from the University of Michigan and his Master's and PhD from the University of California, Berkeley, under advisor David Patterson.

[remzi@cs.wisc.edu](mailto:remzi@cs.wisc.edu)

**ONE OF THE BIGGEST CHALLENGES** IN designing storage systems is providing the reliability and availability that users expect. A serious threat to reliability is silent data corruption (i.e., corruption not detected by the disk drive). In order to develop suitable protection mechanisms against corruption, it is essential to understand its characteristics. In this article, we present the results from the first large-scale field study of data corruption. We analyze over 400,000 corruption instances recorded in production storage systems containing a total of 1.53 million disk drives, over a period of 41 months.

One primary cause of data loss is disk drive unreliability. It is well known that hard drives are mechanical, moving devices that can suffer from mechanical problems, leading to drive failure and *latent sector errors* (detected by the disk's ECC). Less well known, however, is that current hard drives and controllers consist of hundreds of thousands of lines of low-level firmware code. Bugs in this firmware code can cause a more insidious type of disk error: silent data corruption, where the data is silently corrupted with no indication from the drive that an error has occurred.

Silent data corruptions could lead to data loss more often than latent sector errors, since, unlike latent sector errors, they cannot be detected or repaired by the disk drive itself. Worse, basic protection schemes such as RAID may also be unable to detect these problems, thereby returning corrupt data.

The most common technique used in storage systems to detect data corruption is the addition of a higher-level checksum for each disk block, which is validated on each disk block read. However, checksums do not protect against all forms of corruption. Therefore, in addition to checksums, NetApp storage systems also use filesystem-level disk block identity information to detect previously undetectable corruptions.

In order to improve the handling of corruption errors, we need to develop a thorough understanding of data corruption characteristics. Although recent studies provide information on whole disk failures [4, 5, 7] and latent sector errors [1], very little is known about data corruption, its prevalence, and its characteristics. This article summarizes the re-



sults of our study of data corruption first published in the 2008 USENIX FAST conference [2].

---

## Detecting Data Corruption

---

The data we analyze is from tens of thousands of production and development NetApp storage systems from hundreds of customer sites. These storage systems are designed to detect and handle a wide range of disk-related errors, including silent data corruption. Data corruption may be caused by both hardware and software errors. Hardware bugs include bugs in the disk drive or the disk shelf firmware, bad memory, and adapter failures. Typically, it is not possible to identify the root cause of a corruption error. However, our storage system has several mechanisms in place to detect when data corruption occurs, to prevent propagation of corrupt data. We briefly describe two of those mechanisms.

---

### DATA INTEGRITY SEGMENT

---

In order to detect corruptions, the system stores extra information along with each disk block. For every 4KB file system block written, the storage controller writes a 64-byte data integrity segment along with the disk block.

One component of the data integrity segment is a checksum of the entire 4KB filesystem block. The checksum is validated by the RAID layer whenever the data is read. Once a corruption has been detected, the original block can usually be restored through RAID reconstruction. We refer to corruptions detected by RAID-level checksum validation as *checksum mismatches*.

A second component of the data integrity segment is the block identity information. The identity information refers to where the block resides within the file system (e.g., this block belongs to inode 5 at offset 100). This identity is cross-checked at file read time to ensure that the block being read belongs to the file being accessed. If, on file read, the identity does not match, the data is reconstructed from parity. We refer to corruptions that are not detected by checksums, but detected through filesystem identity validation, as *identity discrepancies*.

---

### DATA SCRUBBING

---

In order to proactively detect errors, the RAID layer periodically *scrubs* all disks. A data scrub issues read operations for each physical disk block, computes a checksum over its data, and compares the computed checksum to the checksum located in its data integrity segment. If the checksum comparison fails (i.e., a checksum mismatch), the data is reconstructed from other disks in the RAID group, after those checksums are also verified.

We refer to these cases of mismatch between data and parity as *parity inconsistencies*. Note that data scrubs are unable to validate the extra filesystem identity information stored in the data integrity segment, since this information only has meaning to the file system.

---

### CHECKSUM MISMATCHES

---

As just described, corruption events are classified into three classes: checksum mismatches, identity discrepancies, and parity inconsistencies. In this article we focus on checksum mismatches, since we find that they occur

with the highest frequency. Checksum mismatches can result from (i) data content corrupted by components within the data path, or (ii) a torn write, wherein only a portion of the data block is written successfully, or (iii) a misdirected write, wherein the data is written to either the wrong disk or the wrong location on disk, thus overwriting and corrupting data [3, 6].

Our study focuses on the characteristics of checksum mismatches, such as their frequency, the factors that affect the development of checksum mismatches, and the statistical properties of checksum mismatches. In our analysis we refer to a 4KB file system block with a checksum mismatch as a *checksum mismatch block*. We call a disk drive a *corrupt disk* if it has at least one checksum mismatch block.

---

## DATA COLLECTION

The data we collected covers a period of 41 months starting in January 2004 and includes tens of thousands of NetApp storage systems containing a total of 1.53 million disk drives. The data was collected by a built-in, low-overhead mechanism called AutoSupport. AutoSupport is included in every NetApp storage system and logs system events back to a central repository.

Our disk drive sample is not only large but also diverse. The disks belong to 14 different *disk families*. Each disk family refers to one particular disk drive product. Typically, disks in the same family only differ in the number of platters and/or heads. The drives come from 31 distinct *disk models*. A disk model is the combination of a disk family and a particular disk size. Finally, the drives cover two different *disk classes*: an enterprise class of Fibre Channel disks and a nearline class of SATA disks.

---

## Result Synopsis

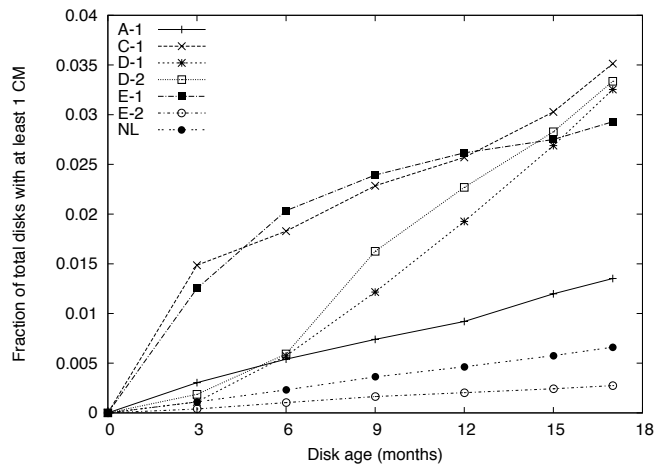
During the 41-month period covered by our data we observed a total of about 400,000 checksum mismatches. Of the total sample of 1.53 million disks, 3855 disks developed checksum mismatches: 3088 of the 358,000 SATA disks (0.86%) and 767 of the 1.17 million Fibre Channel disks (0.065%). This indicates that SATA disks may be more susceptible to corruption leading to checksum mismatches than Fibre Channel disks. On average, each disk developed 0.26 checksum mismatches. By considering only corrupt disks, the mean number of mismatches per disk is 104, the median is 3, and the mode (i.e., the most frequently observed value) is 1 mismatch per disk. The maximum number of mismatches observed for any single drive was 33,000.

---

## DISK CLASS, MODEL, AGE, AND SIZE

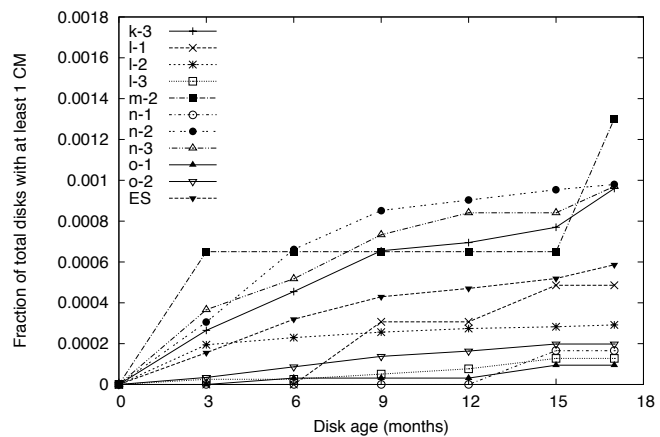
We start by examining the dependence of checksum mismatches on factors such as disk class, disk model, and disk age. A disk's age is its time in the field since its ship date.

Figures 1 and 2 shows the cumulative distribution function of the time in the field until the first checksum mismatch occurs for SATA and Fibre Channel disks, respectively.



**FIGURE 1: CUMULATIVE DISTRIBUTION FUNCTION OF THE TIME IN THE FIELD UNTIL THE FIRST CHECKSUM MISMATCH OCCURS FOR SATA DISKS**

**FIGURE 2: CUMULATIVE DISTRIBUTION FUNCTION OF THE TIME IN**



**THE FIELD UNTIL THE FIRST CHECKSUM MISMATCH OCCURS FOR FIBRE CHANNEL DISKS**

*Observation:* SATA disks have an order of magnitude higher probability of developing checksum mismatches than Fibre Channel disks.

We find that 0.66% of SATA disks develop at least one mismatch during the first 17 months in the field, whereas only 0.06% of Fibre Channel disks develop a mismatch during that time.

*Observation:* The probability of developing checksum mismatches varies significantly across different disk models within the same disk class.

We see that there is an order of magnitude difference for developing at least one checksum mismatch after 17 months between the two most extreme SATA disk models: 3.5% for one model vs. 0.27% for the other.

*Observation:* Age affects different disk models differently with respect to the probability of developing checksum mismatches.

On average, as SATA disks age, the probability of developing a checksum mismatch is fairly constant, with some variation across the models. As Fibre Channel disks age, the probability of developing the first checksum mismatch decreases after about 6–9 months and then stabilizes.

*Observation:* There is no clear indication that disk size affects the probability of developing checksum mismatches.

Since the impact of disk size on the fraction of disks that develop checksum mismatches is seen in only 7 out of 10 families, we conclude that disk size does not necessarily impact the probability of developing checksum mismatches.

---

### CHECKSUM MISMATCHES PER CORRUPT DISK

---

*Observation:* The number of checksum mismatches per corrupt disk varies greatly across disks. Most corrupt disks develop only a few mismatches each. However, a few disks develop a large number of mismatches.

A significant fraction of corrupt disks develop only one checksum mismatch. However, a small fraction of disks develop several thousand checksum mismatches (i.e., 1% of the corrupt disks produce more than half of all mismatches recorded in the data).

*Observation:* On average, corrupt Fibre Channel disks develop many more checksum mismatches than corrupt SATA disks.

Within 17 months, 50% of corrupt disks develop about 2 checksum mismatches for SATA disks but almost 10 for Fibre Channel disks. Given that very few Fibre Channel disks develop checksum mismatches in the first place, it might make sense to replace the Fibre Channel disk when the first mismatch is detected.

*Observation:* Checksum mismatches within the same disk are not independent.

We found that the conditional probability of developing further checksum mismatches, given that a disk has at least one mismatch, is higher than the probability of developing the first mismatch. We also found that one particular SATA disk model is particularly aberrant: Around 30% of its corrupt disks develop more than 1000 checksum mismatches.

---

### DEPENDENCE BETWEEN DISKS IN THE SAME SYSTEM

---

*Observation:* The probability of a disk developing a checksum mismatch is not independent of that of other disks in the same storage system.

Although most systems with checksum mismatches have only one corrupt disk, we do find a considerable number of instances where multiple disks develop checksum mismatches within the same storage system. In fact, one of the systems in the study that used SATA disks had 92 disks develop checksum mismatches. The probability of 92 disks developing errors independently is less than  $10^{-12}$ , much less than  $10^{-5}$ , the approximate fraction of systems represented by one system.

---

### SPATIAL LOCALITY

---

We measure spatial locality by examining whether each checksum mismatch block has another checksum mismatch block (*a neighbor*) within progressively larger regions (*locality radius*) around it on the same disk. For example, if in a disk, blocks numbered 100, 200, and 500 have checksum mismatches, then blocks 100 and 200 have one neighbor at a locality radius of 100, and all blocks (100, 200, and 500) have at least one neighbor at a locality radius of 300.

*Observation:* Checksum mismatches have very high spatial locality. Much of the observed locality is due to consecutive disk blocks developing corruption. Beyond consecutive blocks, the mismatches show very little spatial locality.

For more than 50% of the checksum mismatch blocks in SATA disks and more than 40% of the checksum mismatch blocks in Fibre Channel disks, the immediate neighboring block also has a checksum mismatch (on disks with between 2 and 10 mismatches). These percentages indicate very high spatial locality.

It is interesting to examine how many consecutive blocks have mismatches. We find that, among drives with at least 2 checksum mismatches, on average 3.4 consecutive blocks are affected. In some cases, the length of consecutive runs can be much higher. About 3% of drives with at least 2 mismatches see one or more runs of 100 consecutive blocks with mismatches, and 0.7% of drives with at least 2 mismatches see one or more runs of 1000 consecutive mismatches.

---

### TEMPORAL LOCALITY

*Observation:* Most checksum mismatches are detected within one minute of a previous detection of a mismatch.

*Observation:* Checksum mismatches also exhibit temporal locality over larger time windows and beyond the effect of detection time.

The first observation might not be surprising, since it could just be an artifact of the manner in which the detection takes place (by scrubbing). In order to remove the impact of detection time, we examined temporal locality over larger time windows. For each drive, we first determined the number of checksum mismatches experienced in each two-week time window that the drive was in the field and then computed the autocorrelation function (ACF) on the resulting time series. The ACF can be used to determine whether the number of mismatches in one two-week period of our time series is correlated with the number of mismatches observed in two-week periods later.

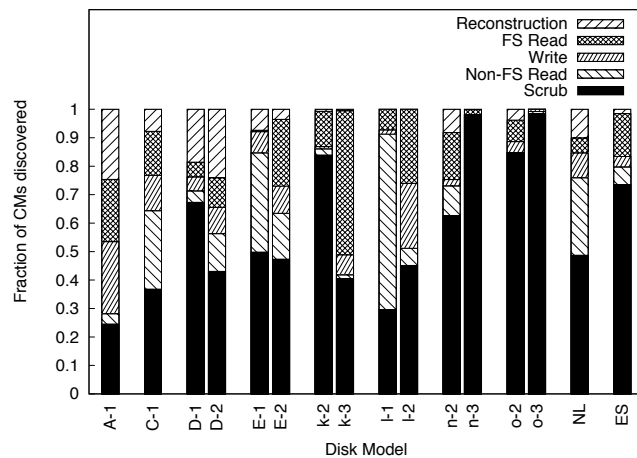
If checksum mismatches in different two-week periods were independent (no temporal locality on bi-weekly and larger time scales), the autocorrelation would be close to zero at all time lags. Instead, we observe strong autocorrelation even for large lags in the range of up to 10 months.

---

### DISCOVERY

The severity of a data corruption event depends on when it is discovered. If a checksum mismatch is encountered during RAID reconstruction, data loss can result if the system is not configured to handle simultaneous disk failures.

Figure 3 (on p. 12) shows the distribution of requests that detect checksum mismatches. There are five types of requests that discover checksum mismatches: (i) file system reads (*FS Read*); (ii) writes by the RAID layer (*Write*); (iii) reads for disk copy operations (*Non-FS Read*); (iv) reads for scrubbing (*Scrub*); and (v) reads for RAID reconstruction (*Reconstruction*).



**FIGURE 3: DISTRIBUTION OF REQUESTS THAT DETECT CHECKSUM MISMATCHES**

*Observation:* RAID reconstruction encounters a non-negligible number of checksum mismatches.

We see that, on average, data scrubbing discovers about 49% of the checksum mismatches in SATA disks and 73% of the checksum mismatches in Fibre Channel disks. Despite the use of data scrubbing, we find that RAID reconstruction discovers about 8% of the checksum mismatches in SATA disks. For some models more than 20% of checksum mismatches were detected during RAID reconstruction. This observation implies that (i) data scrubbing should be performed more aggressively and (ii) systems should consider protection against double disk failures.

### COMPARISON TO LATENT SECTOR ERRORS

When comparing checksum mismatches to latent sector errors we find some interesting similarities and differences:

- Frequency: The probability of developing checksum mismatches is about an order of magnitude smaller than that for latent sector errors.
- Disk model: For both error types, the development of errors depends on the disk model. Interestingly, the SATA disk model with the highest percentage of disks developing latent sector errors also had the lowest percentage of disks developing checksum mismatches.
- Disk class: For both error types, Fibre Channel disks are less likely to develop an error than SATA disks. Surprisingly, however, in both cases, once an error has developed, Fibre Channel disks develop a higher number of errors than SATA disks.
- Spatial locality: Both latent sector errors and checksum mismatches show high spatial locality. However, the locality radius is significantly larger for latent sector errors.

We also found a weak positive correlation between checksum mismatches and latent sector errors. The conditional probability of a latent sector error, given that a disk has checksum mismatch, is about 1.4 times higher than the unconditional probability of a latent sector error for SATA disks and about 2.2 times higher for Fibre Channel disks. We also verified the existence of a correlation between the two error types by performing a chi-square test for independence.

---

## Lessons Learned

---

We present some of the lessons learned from the analysis for corruption-proof storage system design.

- Albeit not as common as latent sector errors, data corruption does happen. For some drive models as many as 4% of drives develop checksum mismatches during the time examined. Even though rare, identity discrepancies and parity inconsistencies do occur. Therefore, the protection offered by checksums and block identity information is critical to protect against data corruption.
- A significant number (8% on average) of corruptions are detected during RAID reconstruction, creating the possibility of data loss. In this case, protection against double disk failures is necessary to prevent data loss.
- Although the probability of developing a corruption is lower for enterprise-class drives, once they develop a corruption, many more are likely to follow. Therefore, replacing an enterprise-class drive on the first detection of a corruption might make sense.
- Strong spatial locality suggests that redundant data structures should be stored at a distance from each other.
- The high degree of spatial and temporal locality may suggest that corruptions occur at the exact same time, perhaps as part of the same disk request. Thus, important or redundant data structures should be written as part of different write requests spaced over time.
- Strong spatial and temporal locality (over long time periods) suggests that it is worth investigating how the locality can be leveraged for smarter, targeted scrubbing (e.g., trigger a scrub before its next scheduled time) or selective scrubbing of an area of the drive that's likely to be affected.
- Failure prediction algorithms in systems should take into account the correlation of corruption with other errors such as latent sector errors.

---

## Conclusion

---

We have analyzed data corruption detected in 1.53 million disks used in production storage systems. During a 41-month period we observed more than 400,000 instances of checksum mismatches, 8% of which were discovered during RAID reconstruction, creating the possibility of real data loss.

We identified various characteristics of checksum mismatches, including: (i) the probability of developing the first checksum mismatch is almost an order of magnitude higher for SATA disks than for Fibre Channel disks; (ii) checksum mismatches are not independent occurrences—both within a disk and within different disks in the same storage system—and the number of mismatches per disk follows a heavy-tailed distribution; and (iii) checksum mismatches also show high spatial and temporal locality, encouraging system designers to develop schemes that spread redundant data with respect to both the on-disk location and the time written.

---

## REFERENCES

---

[1] L.N. Bairavasundaram, G.R. Goodson, S.Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS '07)*, San Diego, California, June 2007.

- [2] L.N. Bairavasundaram, G.R. Goodson, B. Schroeder, A.C. Arpaci-Dusseu, and R. Arpaci-Dusseu, "An Analysis of Data Corruption in the Storage Stack," in *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST '08)*, San Jose, California, Feb. 2008.
- [3] W. Bartlett and L. Spainhower, "Commercial Fault Tolerance: A Tale of Two Systems," *IEEE Transactions on Dependable and Secure Computing*, 1(1):87–96 (Jan. 2004).
- [4] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are Disks the Dominant Contributor for Storage Subsystem Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics," in *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST '08)*, San Jose, California, Feb. 2008.
- [5] E. Pinheiro, W.-D. Weber, and L.A. Barroso, "Failure Trends in a Large Disk Drive Population," in *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST '07)*, San Jose, California, Feb. 2007.
- [6] V. Prabhakaran, L.N. Bairavasundaram, N. Agrawal, H.S. Gunawi, A.C. Arpaci-Dusseu, and R.H. Arpaci-Dusseu, "IRON File Systems," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pp. 206–220, Brighton, United Kingdom, Oct. 2005.
- [7] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" in *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST '07)*, San Jose, California, Feb. 2007.

## Thanks to USENIX and SAGE Corporate Supporters

### USENIX Patrons

Google  
Microsoft Research  
NetApp

### USENIX Benefactors

Hewlett-Packard  
IBM  
*Linux Pro Magazine*  
VMware

### USENIX & SAGE Partners

Ajava Systems, Inc.  
DigiCert® SSL Certification  
FOTO SEARCH Stock Footage  
and Stock Photography  
Raytheon  
rTIN Aps  
Splunk  
Tellme Networks  
Zenoss

### USENIX Partners

Cambridge Computer  
Services, Inc.  
cPacket Networks  
EAGLE Software, Inc.  
GroundWork Open  
Source Solutions  
HypericInfosys  
Intel  
Interhack  
Oracle  
Ripe NCC  
Sendmail, Inc.  
Sun Microsystems, Inc.  
UUNET Technologies, Inc.

### SAGE Partner

MSB Associates



MARK W. STORER, KEVIN M. GREENAN,  
ETHAN L. MILLER, AND KALADHAR  
VORUGANTI

## Pergamum: energy-efficient archival storage with disk instead of tape



Mark W. Storer is a fourth-year graduate student at the University of California, Santa Cruz. His primary research is archival storage, in particular the security, design, and management of long-term storage. He plans to finish his PhD at the end of calendar year 2008.

*mstorer@cs.ucsc.edu*



Kevin Greenan is a graduate student at the University of California, Santa Cruz. His interests include system reliability, novel applications of erasure codes in systems, and power-managed systems.

*kmgreen@cs.ucsc.edu*



Dr. Ethan L. Miller is an associate professor of computer science at the University of California, Santa Cruz, where he is a member of the Storage Systems Research Center (SSRC). His current research projects, which are funded by the NSF, Department of Energy, and industry support for the SSRC, include long-term archival storage systems, scalable metadata and indexing, issues in petabyte-scale storage systems, reliability and security in file systems, and file systems for nonvolatile memory technologies.

*elm@cs.ucsc.edu*



Kaladhar Voruganti is a Technical Director in the Advanced Technology Group at NetApp. He got his PhD in Computing Science from the University of Alberta in Edmonton, Canada. Kaladhar likes to build systems and also write papers.

*kaladhar@netapp.com*

IS THE CART LEADING THE HORSE WHEN it comes to long-term digital storage? Few would disagree that there has been a tremendous shift toward writing our personal histories as digital data. The convenience of digital photos has lured people away from film, just as email has supplanted letters. Unfortunately, we have yet to demonstrate that we can reliably preserve digital data for more than a few years. Will future generations be able to browse the sentimental and historical artifacts we leave them the way we might flip through our grandparents' photo albums? Clearly, the long-term preservation of data requires a storage system that can evolve over time, while remaining cheap enough to allow the retention of anything that might be important. To fill this need, we have developed Pergamum, a distributed network of energy-efficient, hard drive-based storage appliances. Each of our devices, which we call a Pergamum tome, offers the low-latency access times of disks while being cheaper to buy and operate than either disk- or tape-based systems.

### The Archival Storage Problem

The state of archival storage, in the professional sector, is largely the same as in the private sector, although businesses are slowly starting to recognize the importance of archival storage. Further, they are starting to recognize it as a class of storage distinct from mere backups. This is partially due to legislation that has mandated requirements for the preservation, retrieval, and auditing of digital data. However, outside of legal requirements, data-mining techniques have proven to be a boon in shaping business strategy and have demonstrated the enormous value contained in archival data.

Paradoxically, the increasing value of archival data is driving the need for inexpensive, evolvable storage. The goal of cost-efficient, long-term storage is to enable the potentially indefinite retention of all data that might one day prove useful. With current systems, it is simply too expensive to store everything indefinitely (if any long-term persistence guarantees are available at all). Imagine parents implementing a seven-year retention policy on

sentimental data in order to reclaim storage space! With home movies and photography all being done digitally, is this what we are forcing them into? Archival storage therefore needs to be cheap to obtain (static costs), cheap to operate (operational costs), and easy to expand (evolvable). In particular, one of the biggest culprits in high storage costs is energy consumption. Some reports find that commonly used power supplies operate at only 65%–75% efficiency, representing one of the primary culprits of excess heat production and contributing to cooling demands that account for up to 60% of data-center energy usage [7].

Unfortunately, despite its increasing prominence, archival data is still often confused with backup data. The access pattern of archival data is dominated by writes, and data, once written, is rarely changed. Reads are also rare, but although a slight latency penalty is acceptable if it results in cost savings, archival storage must still be fairly accessible. Archival data can be thought of as cold data: You may not need it right away, but it is still useful; its value increases the easier it is to read, query, browse, and search over. In contrast, backup data is a safety net that you only resort to if something else has failed. Moreover, most backup data only needs to live long enough to be superseded by a newer write. Thus, whereas both backup and archival storage are concerned with data safety, the goal of the latter is to maintain both the persistence and the usability of data. This aspect of archival storage can be seen quite clearly in digital libraries. For example, the Digital Library Federation, a consortium of libraries, was formed not only to advance the preservation of digital collections but also to expand their accessibility [4].

As a result of this confusion, digital archives are often relegated to storage systems designed for backup data. Oftentimes, these systems utilize removable media that decouple the media from the access hardware. Although many of these systems seem cost-effective because the media are cheap, when you amortize the high costs of readers, silos, and robots over the number of media, you often discover that these systems are far from a bargain. In addition to hidden costs, decoupled media introduce other problems as well. They generally offer poor access times, and they introduce the need to either preserve complex chains of hardware or institute expensive and time-consuming migration strategies. Tape, the most common media in these decoupled systems, is further hindered by a sequential access pattern. The end result is that it can take on the order of minutes to handle random requests. This conspires against many archival storage operations—such as auditing, searching, consistency checking, and inter-media reliability operations—that rely on relatively fast random-access performance.

In contrast to the decoupled media and reader of tape, hard drive–based storage is an attractive alternative. By coupling the heads and the media, hard drives offer better performance and obviate the need for robotics, reducing physical movement and system complexity. Recent trends showing drive prices dropping relative to tape [8] reinforce the idea that disk-based systems may be feasible for archival storage. Even more promising, recent work on MAIDs (Massive Arrays of Idle Disks) has demonstrated that considerable energy-based cost savings can be realized, while still maintaining high levels of performance, by keeping hard drives spun down [3, 12].

Although inexpensive hard drives can help control static costs, they cannot, by themselves, fully address all the needs of archival storage. Long-lived data has a potentially indefinite lifetime, and that requires a system that can scale across time as well as capacity. Luckily, a number of recent innovations have opened the door to a new model of evolvable storage. High-performance, low-power CPUs and inexpensive, high-speed networks make it

possible to produce a self-contained, network-attached storage device [6] with reasonable performance and low power utilization. The Ethernet backplane helps simplify the long-term maintenance, as interfaces and protocols are standardized and have changed much more slowly than storage-specific interfaces. The long-term benefit of a network of intelligent storage devices is that the system can be largely agnostic to how the actual devices are implemented. For example, in fifty years, the devices might utilize holographic storage, but their admission to the group will still only be predicated on the ability to speak a given protocol and their ability to perform a set of well-defined tasks.

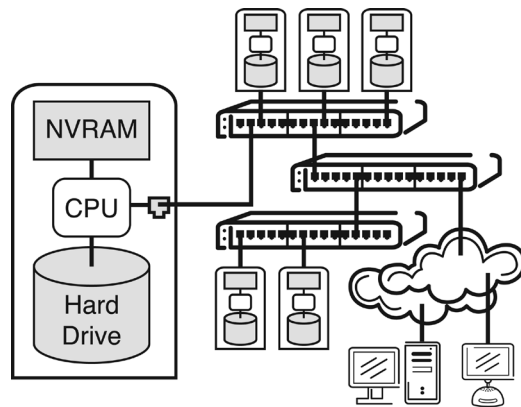
The system we have developed using this model, called Pergamum, consists of a distributed network of independent, intelligent storage appliances. Other distributed systems exist, but they either compromise a fully distributed design for easier management [5] or do not achieve the level of power savings needed in archival storage [13, 9]. Although each device in our system is fully self-sufficient and manages its own consistency checking and disk scrubbing, the devices cooperate in inter-device redundancy schemes so that, even if a unit fails, data can be rebuilt.

---

## The Design of Pergamum

---

The Pergamum tomes that make up our system are simple, intelligent storage devices. As Figure 1 shows, each unit is composed of four hardware components: a commodity hard drive for persistent, large-capacity storage; on-board flash memory for persistent, low-latency metadata storage; a low-power CPU; and a network port. The result is a reliable, low-power storage device that can be used as a building block for more advanced systems.



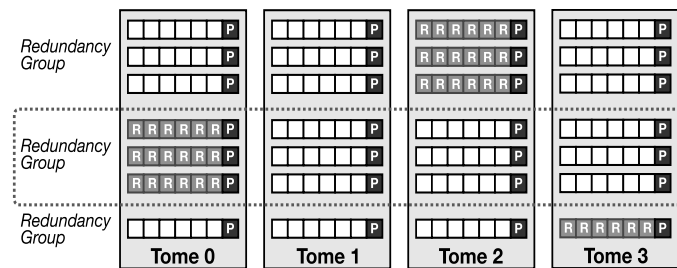
**FIGURE 1: HIGH-LEVEL SYSTEM DESIGN OF PERGAMUM. INDIVIDUAL PERGAMUM TOMES ARE CONNECTED BY A COMMODITY NETWORK BUILT FROM OFF-THE-SHELF SWITCHES.**

When fully active, each Pergamum tome consumes less than 13 watts, well within the capabilities of power over Ethernet. Of that, the disk itself is by far the largest energy consumer. This explains why MAID systems have been able to achieve considerable power savings by keeping idle disks spun down. Our design goes a step further and achieves even more cost savings by moving away from the power-hungry, centralized controllers found in most MAID systems. Each Pergamum tome consumes less than a single watt in its spun-down, idle mode! By pairing a 2–3 watt processor with each disk, we can gracefully scale the power consumed to the size of the system's load.

The form factor of each Pergamum tome can be quite compact. As the low-power processing boards are roughly the size of a pack of gum (or smaller),

the entire device would not be much larger than the drive itself. With power over Ethernet, each Pergamum tome is essentially a sealed device with a single connection. This, together with the lower air space requirements of idle drives, means that very high storage densities can be achieved. It also opens the door for novel rack configurations. Unlike a tape silo, there is no need to provide room for robots to operate.

In addition to our choice of numerous low-power processors, the theme of scaling the response to the size of the task can also be seen in our reliability model. As Figure 2 shows, Pergamum utilizes two levels of redundancy encoding: intra-disk and inter-disk. Individual segments are protected with redundant blocks on the same disk (those labeled with a P). Redundancy groups are protected by the shaded segments (labeled R), which contain erasure correcting codes for the other segments in the redundancy group. Note that segments used for redundancy still contain intradisk redundant blocks to protect them from latent sector errors. Recent work has highlighted the danger of latent sector errors on disks [2]. These are errors that often go undetected until they are read. In a traditional RAID system, at the first sign of any trouble, all the disks in the redundancy group would be spun up. This one-size-fits-all approach to data recovery works well, but it is very expensive. In our system, for many errors, the Pergamum tome can utilize its own intra-disk parity to recover from such errors without waking up a single other device. By using two levels of redundancy, Pergamum achieves higher reliability compared to traditional RAID setups and much higher power savings for the price of only a slight increase in storage overhead.



**FIGURE 2: THE TWO LEVELS OF REDUNDANCY IN PERGAMUM**

Although flash memory has received a fair amount of press lately, it will be some time before it is cheap enough to use as the primary storage medium for archival data. This is not, however, to say that it is not useful in long-term storage. As Figure 1 shows, each Pergamum tome contains a small amount of nonvolatile RAM. This is used as a persistent metadata store. It allows the Pergamum tome to handle many types of requests without spinning up the disk.

One of the most important types of metadata that we store in NVRAM is data signatures. Similar to a traditional hash value, such as SHA-1, data signatures allow us to confirm the correctness of data during a disk scrub or read request; the stored signatures can be compared to a signature calculated over the data being read. The signatures we use, however, are called algebraic signatures and they display a rather useful characteristic [10]. For many codes, the signatures of the data exhibit the same relationship as the data itself. In other words, if data blocks A and B generated parity block P (as in a traditional RAID system), then the signatures of A and B will generate the signature of P. Using these signatures, Pergamum is able to check the consistency not only of the data stored on each Pergamum tome but of the entire inter-disk redundancy group. Moreover, using trees of hash values, we greatly reduce the amount of signature data we need to exchange between nodes in order to confirm the integrity of the inter-disk redundancy

groups. We describe our approach in full detail in our FAST '08 paper on Pergamum [11].

---

## Results and Observations

---

One of the challenges raised by the use of low-power processors is the need for tight engineering and optimized code. Today, even laptop processors have the horsepower to make non-native, high-level scripting languages perform well, albeit at a huge power cost. As this project has spent its life in a research lab and not in an engineering department, almost all of the software running on the Pergamum tome has been implemented in Python, and it can only be considered proof-of-concept-level code. The results are nonetheless promising. For example, whereas data encoding on a Pergamum tome took almost ten times as long as on a laptop processor, the laptop processor consumed more than ten times the power. However, early system profiling indicates that native implementations and careful optimizations can reap great improvements in performance.

Although we are still early in development, our experiments suggest that our model is a viable approach to archival storage. Our two-level approach to reliability provides excellent protection against latent sector errors as well as full drive failure, and it does so while incurring only minimal storage overhead. Regarding performance, our initial optimizations suggest that we have yet to tap the full potential of our low-power processors, not to mention the possible level of parallelism inherent in our design. Finally, our cost analysis suggests that we can be very price-competitive with tape, while offering functionality that tape systems simply cannot provide. All of these results and a full explanation of our experimental methods are available in our FAST '08 paper on Pergamum [11].

---

## Where Do We Go from Here?

---

Pergamum demonstrates some of the features needed in an archival storage system, but work remains to turn it into a fully effective, evolving, long-term storage system. In addition to the engineering tasks associated with optimizing the Pergamum implementation for low-power CPUs, there are a number of important research areas to examine.

Storage management in Pergamum, and in archival storage in general, is an open area with a number of interesting problems. Management strategies play a large part in cost efficiency; many believe that management costs eclipse hardware costs [1]. The goal of our management research is to maintain the decentralized design of Pergamum, while making the addition and removal of drives as automated as possible. In the model we envision, at a frequency of no more than once a month a minimally trained administrator would be tasked with adding new devices to the network and removing failed devices. Once added, the devices would automatically find the existing nodes and either join their redundancy groups or join new redundancy groups created by the system.

In our current implementation, users interact with Pergamum by submitting requests to specific Pergamum tomes using a connection-oriented protocol. In future versions, the use of a simple, standardized put and get style protocol, such as that provided by HTTP, could allow storage to be more evolvable and permit the use of standard tools for storing and retrieving information. Further, techniques such as distributed searching that take into account data movement and migration could greatly simplify how users interact with the system.

Part of a storage administrator's role has traditionally been to decide how much storage overhead to accept in order to increase storage redundancy. Moving forward, this role will grow slightly more complicated as administrators increasingly consider energy costs as well. In order to make an informed decision, the interplay of redundancy, storage overhead, and power consumption must be better understood. Part of our work in this area is to develop data-protection strategies that are best suited to the unique demands and usage model of archival storage.

Although there is still a lot of work to do to turn Pergamum into a fully functioning, evolvable, archival storage system, our storage model is promising. In its current state, Pergamum uses low-power, network-attached disk appliances to provide reliable, cost-effective archival storage. Two levels of redundancy encoding, within disks and across disks, provide both reliability and cost savings, as data recovery techniques can be appropriately scaled to the size of the data-loss events. Finally, Pergamum achieves its cost-efficiency goals by controlling both static and operational costs. We keep fixed costs low through the use of standardized network interfaces and commodity hardware, allowing each Pergamum tome to be viewed as an essentially "disposable" appliance; a system operator can simply throw away faulty nodes. Operational costs are controlled by utilizing ultra-low-power CPUs, power-managed disks, and a myriad of new techniques such as local NVRAM for caching metadata and redundancy information to avoid disk spin-ups, intra-disk redundancy, and trees of algebraic signatures for distributed consistency checking.

---

## Historical Note

---

Our system is named after the Library of Pergamum, one of the most famous libraries of the ancient world. Located in modern-day Turkey, then a part of ancient Greece, the library was built by Eumenes II. Two distinctions make this an apt inspiration for our system. First, the Library of Pergamum is the home and namesake of parchment. At the time, manuscripts were written on papyrus, which was expensive, as it was produced only in Alexandria. Second, the library took great care in the layout of its shelves, as it recognized the importance that airflow played in the long-term persistence of its works.

---

## ACKNOWLEDGMENTS

---

We would like to thank our colleagues in the Storage Systems Research Center (SSRC), who provided valuable feedback on the ideas in this paper, helping us to refine them.

This research was supported by the Petascale Data Storage Institute under Department of Energy award DE-FC02-06ER25768 and by the industrial sponsors of the SSRC, including Los Alamos National Lab, Livermore National Lab, Sandia National Lab, Agami Systems, Data Domain, Digisense, Hewlett-Packard Laboratories, IBM Research, LSI Logic, Network Appliance, Seagate, Symantec, and Yahoo!.

---

## REFERENCES

---

[1] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: Running Circles around Storage Administration," in *Proceedings of the 2002 Conference on File and Storage Technologies (FAST '02)*, Monterey, CA, Jan. 2002.

- [2] L.N. Bairavasundaram, G.R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," in *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 2007.
- [3] D. Colarelli and D. Grunwald, "Massive Arrays of Idle Disks for Storage Archives," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC '02)*, Nov. 2002.
- [4] Digital Library Federation: <http://www.diglib.org> (accessed Mar. 2008).
- [5] S. Ghemawat, H. Gobiuff, and S.-T. Leung, "The Google File System," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, Oct. 2003.
- [6] G.A. Gibson and R. Van Meter, "Network Attached Storage Architecture," *Communications of the ACM*, 43(11):37–45, 2000.
- [7] Green Grid Consortium: <http://www.thegreengrid.org>, Feb. 2007.
- [8] W. C. Preston and G. Didio, "Disk at the Price of Tape? An In-depth Examination," Copan Systems white paper, 2004.
- [9] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building Distributed Enterprise Disk Arrays from Commodity Components," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 48–58, 2004.
- [10] T. Schwarz and E.L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," in *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS '06)*, Lisbon, Portugal, July 2006.
- [11] M.W. Storer, K.M. Greenan, E.L. Miller, and K. Voruganti, "Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-based Archival Storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, Feb. 2008.
- [12] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning, "PARAID: A Gear-shifting Power-aware RAID," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)*, Feb. 2007.
- [13] W.W. Wilcke, R.B. Garner, C. Fleiner, R.F. Freitas, R.A. Golding, J.S. Glider, D.R. Kenchammana-Hosekote, J.L. Hafner, K.M. Mohiuddin, K. Rao, R.A. Becker-Szendy, T.M. Wong, O.A. Zaki, M. Hernandez, K.R. Fernandez, H. Huels, H. Lenk, K. Smolin, M. Ries, C. Goettert, T. Picunko, B.J. Rubin, H. Kahn, and T. Loo, "IBM Intelligent Bricks Project—Petabytes and Beyond," *IBM Journal of Research and Development*, 50(2/3):181–197, 2006.

WEIHANG JIANG, CHONGFENG HU,  
YUANYUAN ZHOU, AND ARKADY KANEVSKY

## don't blame disks for every storage subsystem failure



Weihang Jiang is a PhD candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He is particularly interested in system mining that applies data mining techniques to improve system performance, dependability, and manageability.

wjiang3@uiuc.edu



Chongfeng Hu spent his college life at Peking University, China, and is currently a graduate student in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His interests include operating systems, software reliability, storage systems reliability, and data mining.

chu7@cs.uiuc.edu



Yuanyuan Zhou is an associate professor at the University of Illinois, Urbana-Champaign. Her research spans the areas of operating system, storage systems, software reliability, and power management.

yyzhou@uiuc.edu



Arkady Kanevsky is a senior research engineer at NetApp Advanced Technology Group. Arkady has done extensive research on RDMA technology, storage resiliency, scalable storage systems, and parallel and distributed computing. He received a PhD in Computer Science from the University of Illinois in 1987. He was a faculty member at Dartmouth College and Texas A&M University prior to joining the industry world. Arkady has written over 60 publications and is a chair of DAT Collaborative and MPI-RT standards.

arkady@netapp.com

Trademark notice: NetApp, the NetApp logo, Go further, faster, and RAID-DP are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

**DISKS ARE THE KEY COMPONENTS OF** storage systems. Researchers at CMU and NetApp had demonstrated a trend of increasing gap between the size of individual disks and disk access time [11], and hence the probability that a secondary failure happens during RAID reconstruction becomes too high for comfort. This led to RAID-6 [4] and RAID-DP [5]. Interestingly, even though disk reliability is critical to storage systems reliability, we found that disks themselves were not the component most likely to fail in storage systems.

In this work, we looked at other components in storage systems beyond disks to answer the following questions: Are disk failures the main source of storage system failures? Can enterprise disks help to build a more reliable storage system than SATA disks? Are disk failures independent? What about other storage component failures? Are techniques that went into RAID design, such as redundant interconnect between disks and storage controllers, really helpful in increasing the reliability of storage systems?

Reliability is a critically important issue for storage systems because storage failures not only can cause service downtime but can also lead to data loss. Building reliable storage systems becomes increasingly challenging as the complexity of modern storage systems grows to an unprecedented level. For example, the EMC Symmetrix DMX-4 can be configured with up to 2400 disks [6], the Google File System cluster is composed of 1000 storage nodes [7], and the NetApp FAS6000 series can support more than 1000 disks per node, with up to 24 nodes in a system [9].

To make things even worse, disks are not the only component in storage systems. To connect and access disks, modern storage systems also contain many other components, including shelf enclosures, cables and host adapters, and complex software protocol stacks. Failures in these components can lead to downtime and/or data loss of the storage system. Hence, in complex storage systems, component failures are very common and critical to storage system reliability.

Although we are interested in failures of a whole storage system, this study concentrates on the core part of it—the *storage subsystem*, which contains



disks and all components providing connectivity and usage of disks to the entire storage system.

We analyzed the NetApp AutoSupport logs collected from about 39,000 storage systems commercially deployed at various customer sites. The data set covers a period of 44 months and includes about 1,800,000 disks hosted in about 155,000 storage shelf enclosures. Our study reveals many interesting findings, providing useful guidelines for designing reliable storage systems. Some of our major findings include:

- Physical interconnect failures make up the largest part (27%–68%) of storage subsystem failures, and disk failures make up the second largest part (19%–56%). Choices of disk types, shelf enclosure models, and other components of storage subsystems contribute to the variability.
- Each individual storage subsystem failure type and storage subsystem failure as a whole exhibit strong self-correlations.
- Storage subsystems configured with redundant interconnects experience 30%–40% lower failure rates than those with a single interconnect.

Data on latent sector errors from the same AutoSupport Database was first analyzed by Bairavasundaram et al. [2], and data on data corruptions was further analyzed by Bairavasundaram et al. [3].

---

## Background

---

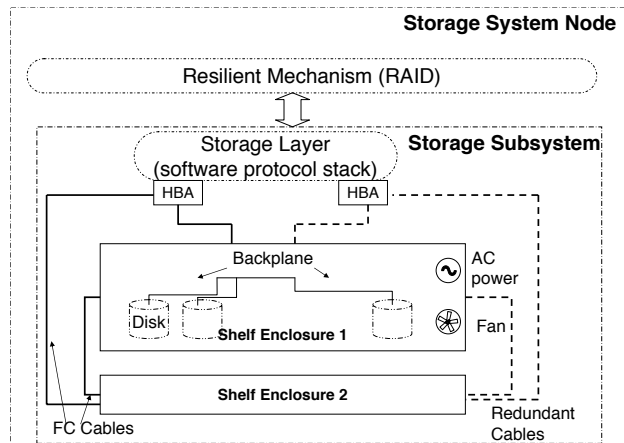
In this section, we detail the typical architecture of storage systems we study in NetApp, the definitions and terminology used in this article, and the source of the data studied in this work.

---

### STORAGE SYSTEM ARCHITECTURE

---

Figure 1 shows the typical architecture of a NetApp storage system node. A NetApp storage system can be composed of several storage system nodes.



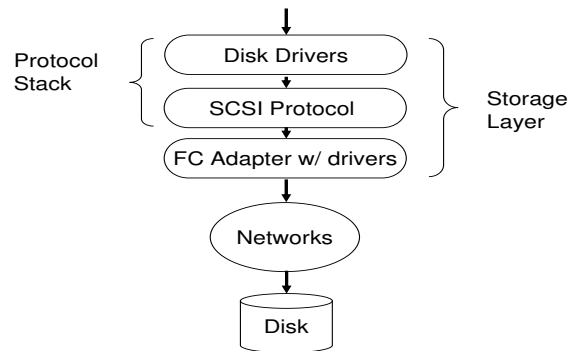
**FIGURE 1: STORAGE SYSTEM NODE ARCHITECTURE**

From the customer's perspective, a storage system is a virtual device that is attached to customers' systems and provides customers with the desired storage capacity with high reliability, good performance, and flexible management.

Looking from inside, we see that a storage system node is composed of storage subsystems, resiliency mechanisms, a storage head/controller, and other

higher-level system layers. The storage subsystem is the core part of a storage system node and provides connectivity and usage of disks to the entire storage system node. It contains various components, including disks, shelf enclosures, cables and host adapters, and complex software protocol stacks. Shelf enclosures provide a power supply, a cooling service, and a prewired backplane for the disks mounted in them. Cables initiated from host adapters connect one or multiple shelf enclosures to the network. Each shelf enclosure can be optionally connected to a secondary network for redundancy. In the Results section we will show the impact of this redundancy mechanism on failures of the storage subsystem.

Usually, on top of the storage subsystem, resiliency mechanisms, such as RAID, are used to tolerate failures in storage subsystems.



**FIGURE 2: I/O REQUEST PATH IN STORAGE SUBSYSTEM**

#### TERMINOLOGY

We use the followings terms in this article:

- *Disk family*: A particular disk product. The same product may be offered in different capacities. For example, “Seagate Cheetah 10k.7” is a disk family.
- *Disk model*: The combination of a disk family and a particular disk capacity. For example, “Seagate Cheetah 10k.7 300 GB” is a disk model. For disk family and disk model, we use the same naming convention as in Bairavasundaram et al. [2, 3]. (See also “Data Corruption in the Storage Stack,” p. 6 in this issue.)
- *Failure types*: Refers to the four types of storage subsystem failures—disk failure, physical interconnect failure, protocol failure, and performance failure.
- *Shelf enclosure model*: A particular shelf enclosure product. All shelf enclosure models studied in this work can host at most 14 disks.
- *Storage subsystem failure*: Refers to failures that prevent the storage subsystem from providing storage service to the whole storage system node. However, not all storage subsystem failures are experienced by customers, since some of the failures can be handled by resiliency mechanisms on top of storage subsystems (e.g., RAID) and other mechanisms at higher layers.
- *Storage system class*: Refers to the capability and usage of storage systems. There are four storage system classes studied in this work: nearline systems (mainly used as secondary storage), low-end, mid-range, and high-end (mainly used as primary storage).
- Other terms in the article are used as defined by SNIA [12].

---

## DEFINITION AND CLASSIFICATION OF STORAGE SUBSYSTEM FAILURES

---

Figure 2 shows the steps and components that are involved in fulfilling an I/O request in a storage subsystem. As shown in Figure 2, for the storage layer to fulfill an I/O request, the I/O request will first be processed and transformed by protocols and then delivered to disks through networks initiated by host adapters. *Storage subsystem failures* are the failures that break the I/O request path; they can be caused by hardware failures, software bugs, and protocol incompatibilities along the path.

To better understand storage subsystem failures, we categorize them into four types along the I/O request path:

- *Disk failure*: This type of failure is triggered by failure mechanisms of disks. Imperfect media, media scratches caused by loose particles, rotational vibration, and many other factors internal to a disk can lead to this type of failure. Sometimes the storage layer proactively fails disks based on statistics collected by on-disk health monitoring mechanisms (e.g., a disk has experienced too many sector errors [1]). These incidences are also counted as disk failures.
- *Physical interconnect failure*: This type of failure is triggered by errors in the networks connecting disks and storage heads. It can be caused by host adapter failures, broken cables, shelf enclosure power outages, shelf backplanes errors, and/or errors in shelf FC drivers. When physical interconnect failures happen, the affected disks appear to be missing from the system.
- *Protocol failure*: This type of failure is caused by incompatibility between protocols in disk drivers or shelf enclosures and storage heads and software bugs in the disk drivers. When this type of failure happens, disks are visible to the storage layer but I/O requests are not correctly responded to by disks.
- *Performance failure*: This type of failure happens when the storage layer detects that a disk cannot serve I/O requests in a timely manner while none of the previous three types of failures are detected. It is mainly caused by partial failures, such as unstable connectivity or when disks are heavily loaded with disk-level recovery (e.g., broken sector remapping).

The occurrences of these four types of failures are recorded in AutoSupport logs collected by NetApp.

---

## Results

---

---

### FREQUENCY OF STORAGE SUBSYSTEM FAILURES

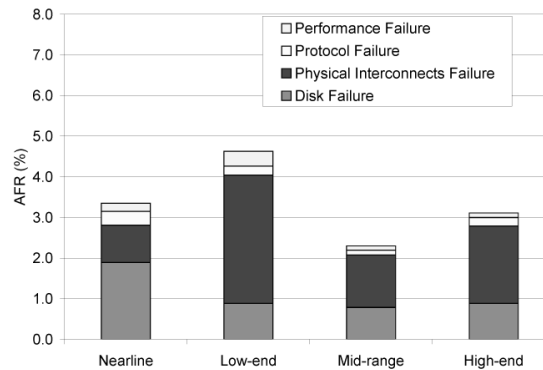
---

As we categorize storage subsystem failures into four failure types based on their root causes, a natural question is therefore, “What is the relative frequency of each failure type?” To answer this question, we study the NetApp AutoSupport logs collected from 39,000 storage systems.

Figure 3 presents the breakdown of the average failure rate (AFR) for storage subsystems based on failure types, for all four system classes studied in this work.

*Finding (1)*: Physical interconnect failures make up the largest part (27%–68%) of storage subsystem failures; disk failures make up the second largest part (20%–55%). Protocol failures and performance failures both make up noticeable fractions.

*Implications:* Disk failures are not always a dominant factor in storage subsystem failures, and a reliability study for storage subsystems cannot focus only on disk failures. Resilient mechanisms should target all failure types.



**FIGURE 3: AFR FOR STORAGE SUBSYSTEMS IN FOUR SYSTEM CLASSES AND THE BREAKDOWN BASED ON FAILURE TYPES**

As Figure 3 shows, across all system classes, disk failures do not always dominate storage subsystem failures. For example, in low-end storage systems, the AFR for storage subsystems is about 4.6%, whereas the AFR for disks is only 0.9%, about 20% of the overall AFR. However, physical interconnect failures account for a significant fraction of storage subsystem failures, ranging from 27% to 68%. The other two failure types, protocol failures and performance failures, contribute 5%–10% and 4%–8% of storage subsystem failures, respectively.

*Finding (2):* For disks, nearline storage systems show higher (1.9%) AFR than low-end storage systems (0.9%). But for the whole storage subsystem, nearline storage systems show lower (3.4%) AFR than low-end primary storage systems (4.6%).

*Implications:* Disk failure rate is not indicative of the storage subsystem failure rate.

Figure 3 also shows that nearline systems, which mostly use SATA disks, experience about 1.9% AFR for disks, whereas for low-end, mid-range, and high-end systems, which mostly use FC disks, the AFR for disks is under 0.9%. This observation is consistent with the common belief that enterprise disks (FC) are more reliable than nearline disks (SATA).

However, the AFR for storage subsystems does not follow the same trend. Storage subsystem AFR of nearline systems is about 3.4%, lower than that of low-end systems (4.6%). This indicates that other factors, such as shelf enclosure model and network configurations, strongly affect storage subsystem reliability. The impacts of these factors are examined next.

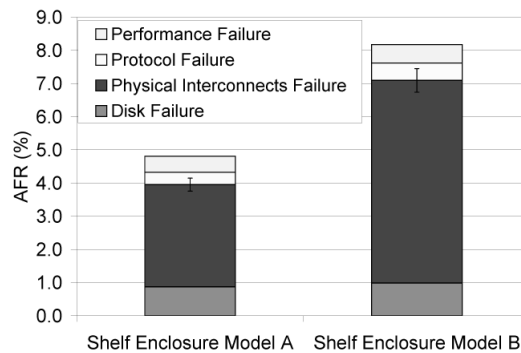
### IMPACT OF SYSTEM PARAMETERS ON STORAGE SUBSYSTEM FAILURES

As we have seen, storage subsystems of different system classes show different AFRs. Although these storage subsystems are architecturally similar, the characteristics of their components, such as shelves, and their redundancy mechanisms, such as multipathing, differ. We now explore the impact of these factors on storage subsystem failures.

#### SHELF ENCLOSURE MODEL

Shelf enclosures contain power supplies, cooling devices, and prewired backplanes that carry power and I/O bus signals to the disks mounted in

them. Different shelf enclosure models are different in design and have different mechanisms for providing these services; therefore, it is interesting to see how shelf enclosure model affects storage subsystem failures.



**FIGURE 4: AFR FOR STORAGE SUBSYSTEMS BY SHELF ENCLOSURE MODELS USING THE SAME DISK MODEL. THE ERROR BARS SHOW 99.9% CONFIDENCE INTERVALS FOR PHYSICAL INTERCONNECT FAILURES.**

*Finding (3):* The shelf enclosure model has a strong impact on storage subsystem failures.

*Implications:* To build a reliable storage subsystem, hardware components other than disks (e.g., shelf enclosure) should also be selected carefully.

Figure 4 shows AFR for storage subsystems when configured with different shelf enclosure models but the same disk models. As expected, shelf enclosure model primarily impacts physical interconnect failures, with little impact on other failure types.

To confirm this observation, we tested the statistical significance using a T-test [10]. As Figure 4 shows, the physical interconnect failures with different shelf enclosure models are quite different ( $3.08 \pm 0.20\%$  versus  $6.11 \pm 0.35\%$ ). A T-test shows that this is significant at the 99.9% confidence interval, indicating that the hypothesis that physical interconnect failures are impacted by shelf enclosure models is very strongly supported by the data.

#### **NETWORK REDUNDANCY MECHANISM**

As we have seen, physical interconnect failures contribute to a significant fraction (27%–68%) of storage subsystem failures. Since physical interconnect failures are mainly caused by network connectivity issues in storage subsystems, it is important to understand the impact of network redundancy mechanisms on storage subsystem failures.

For the mid-range and high-end systems studied in this work, FC drivers support a network redundancy mechanism, commonly called *active/passive multipathing*. This network redundancy mechanism connects shelves to two independent FC networks, redirecting I/O requests through the redundant FC network when one FC network experiences network component failures (e.g., broken cables).

To study the effect of this network redundancy mechanism, we look at the data collected from mid-range and high-end storage systems, and we group them based on whether the network redundancy mechanism is turned on. Owing to the space limitation, we show results only for the mid-range storage systems here. As we observed from our data set, about 1/3 of storage subsystems are utilizing the network redundancy mechanism, whereas the

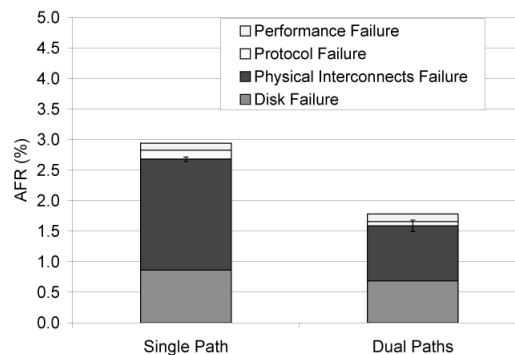
other 2/3 are not. We call these two groups of storage subsystems *dual path* systems and *single path* systems, respectively.

*Finding (4):* Storage subsystems configured with network redundancy mechanisms experience much lower (30%–40% lower) AFR than other systems. AFR for physical interconnects is reduced by 50%–60%.

*Implications:* Network redundancy mechanisms such as multipathing can greatly improve the reliability of storage subsystems.

Figure 5 shows the AFR for storage subsystems in mid-range systems. As expected, secondary path reduces physical interconnect failures by 50%–60% ( $1.82 \pm 0.04$  % versus  $0.91 \pm 0.09$  %), with little impact on other failure types. Since physical interconnect failure is just a subset of all storage subsystem failures, AFR for storage subsystems is reduced by 30%–40%. This indicates that multipathing is an exceptionally good redundancy mechanism that delivers reduction of failure rates as promised. As we applied a T-test on these results, we found that the observation is significant at the 99.9% confidence interval, indicating that the data strongly supports the hypothesis that physical interconnect failures are reduced by multipathing configuration.

However, the observation also tells us that there is still further potential in network redundancy mechanism designs. For example, given that the probability for one network to fail is about 2%, the idealized probability for two networks to both fail should be a few orders of magnitude lower (about 0.04%). But the AFR we observe is far from the ideal number.



**FIGURE 5: AFR FOR STORAGE SUBSYSTEMS BROKEN DOWN BY THE NUMBER OF PATHS. THE ERROR BARS SHOW 99.9% CONFIDENCE INTERVALS FOR PHYSICAL INTERCONNECT FAILURES.**

### CORRELATIONS BETWEEN FAILURES

In this subsection, we will study the statistical property of storage subsystem failures both from a shelf perspective and from a RAID group perspective.

Our analysis of the correlation between failures is composed of two steps:

1. Derive the theoretical failure probability model based on the assumption that failures are independent.
2. Evaluate the assumption by comparing the theoretical probability against empirical results.

Next, we describe the statistical method we use for deriving the theoretical failure probability model.

## STATISTICAL METHOD

We denote the probability for a shelf enclosure (including all mounted disks) to experience one failure during time  $T$  as  $P(1)$  and denote the probability for it to experience two failures during  $T$  as  $P(2)$ . The relationship between  $P(1)$  and  $P(2)$  is as follows:

For a complete proof, refer to our conference paper [8].

$$P(2) = \frac{1}{2} P(1)^2$$

Next, we will compare this theoretically derived model against the empirical results collected from NetApp AutoSupport logs.

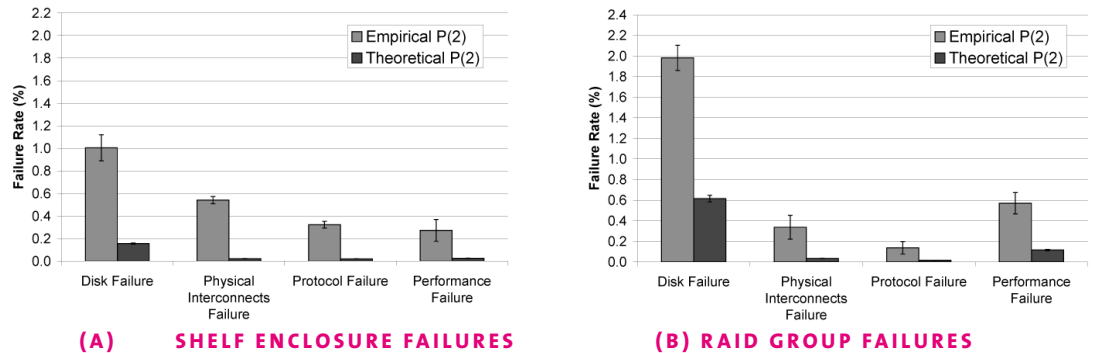
## CORRELATION RESULTS

To evaluate the theoretical relation between  $P(1)$  and  $P(2)$  shown in equation 1, we first calculate *empirical*  $P(1)$  and *empirical*  $P(2)$  from NetApp AutoSupport logs. Empirical  $P(1)$  is the percentage of shelves (RAID groups) that have experienced exactly one failure during time  $T$  (where  $T$  is set to one year), and empirical  $P(2)$  is the percentage that have experienced exactly two failures during time  $T$ . Only storage systems that have been in the field for one year or more are considered.

*Finding (5):* For each failure type, storage subsystem failures are not independent. After one failure, the probability of additional failures (of the same type) is higher.

*Implications:* The probability of storage subsystem failures depends on factors shared by all disks in the same shelf enclosures (or RAID groups).

Figure 6a shows the comparison between *empirical*  $P(2)$  and *theoretical*  $P(2)$ , which is calculated based on empirical  $P(1)$ . As we can see in the figure, empirical  $P(2)$  is higher than theoretical  $P(2)$ . More specifically, for disk failure, the observed empirical  $P(2)$  is higher than theoretical  $P(2)$  by a factor of 6. For other types of storage subsystem failures, the empirical probability is higher than the theoretical correspondences by a factor of 10–25. Furthermore, T-tests confirm that the theoretical  $P(2)$  and the empirical  $P(2)$  are statistically different with 99.5% confidence intervals.



**FIGURE 6: COMPARISON BETWEEN THEORETICAL MODEL [WITH  $P(2)$  CALCULATED FROM EQUATION 1] AND EMPIRICAL RESULTS; THE ERROR BARS SHOW 99.5%+ CONFIDENCE INTERVALS.**

These statistics provide a strong indication that when a shelf experiences a storage subsystem failure, the probability for it to have another storage subsystem failure increases. In other words, storage subsystem failures from the same shelves are not independent.

Figure 6b shows an even stronger trend for failures from the same RAID groups. Therefore, the same conclusion can be made for storage subsystem failures from the same RAID groups.

---

## Conclusion

---

In this article we have presented a study of NetApp storage subsystem failures, examining the contribution of different failure types, the effect of some factors on failures, and the statistical properties of failures. Our study is based on AutoSupport logs collected from 39,000 NetApp storage systems, which contain about 1,800,000 disks mounted in about 155,000 shelf enclosures. The studied data covers a period of 44 months. The findings of our study provide guidelines for designing more reliable storage systems and developing better resiliency mechanisms.

Although disks are the primary components of storage subsystems, and disk failures contribute 19%–56% of storage subsystem failures, other components such as physical interconnect and protocol stacks also account for significant percentages (27%–68% and 5%–10%, respectively) of storage subsystem failures. The results clearly show that the other storage subsystem components cannot be ignored when designing a reliable storage system.

One way to improve storage system reliability is to select more reliable components. As the data suggests, storage system reliability is highly dependent on shelf enclosure model. Another way to improve reliability is to employ redundancy mechanisms to tolerate component failures. One such mechanism studied in this work is multipathing, which can reduce AFR for storage systems by 30%–40% when the number of paths is increased from one to two.

We also found that the storage subsystem failure and individual storage subsystem failure types exhibit strong self-correlations. This finding motivates revisiting resiliency mechanisms, such as RAID, that assume independent failures.

A preliminary version of this article was published at FAST '08 [8]. Because of limited space, neither this article nor our FAST paper includes all our results. Readers who are interested in a complete set of results should refer to our NetApp technical report [13].

---

## ACKNOWLEDGMENTS

---

We wish to thank Rajesh Sundaram and Sandeep Shah for providing us with insights on storage failures. We are grateful to Frederick Ng, George Kong, Larry Lancaster, and Aziz Htite for offering help on understanding and gathering NetApp AutoSupport logs. We appreciate useful comments from members of the Advanced Development Group, including David Ford, Jiri Schindler, Dan Ellard, Keith Smith, James Lentini, Steve Byan, Sai Sussarla, and Shankar Pasupathy. Also, we would like to thank Lakshmi Bairavasundaram for his useful comments. Finally, we appreciate our shepherd, Andrea Arpaci-Dusseau, for her invaluable feedback and precious time, and the anonymous reviewers for their insightful comments for our conference paper [8]. This research has been funded by NetApp under the “Intelligent Log Mining” project at CS UIUC. Work of the first two authors was conducted in part as summer interns at NetApp.



## REFERENCES

- [1] Bruce Allen, “Monitoring Hard Disks with SMART,” *Linux Journal*, 2004(117):9 (2004).
- [2] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler, “An Analysis of Latent Sector Errors in Disk Drives,” *SIGMETRICS Perform. Eval. Rev.* 35(1):289–300 (2007).
- [3] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, “An Analysis of Data Corruption in the Storage Stack,” in *FAST '08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, San Jose, CA, February 2008.
- [4] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon, “Evenodd: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures,” *IEEE Transactions on Computing*, 44:192–202 (1995).
- [5] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar, “Row-diagonal Parity for Double Disk Failure Correction,” in *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pp. 1–14 (2004).
- [6] EMC Symmetrix DMX-4 Specification Sheet (July 2007): <http://www.emc.com/collateral/hardware/specification-sheet/c1166-dmx4-ss.pdf>.
- [7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The Google File System,” in *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, New York, pp. 29–43 (2003).
- [8] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky, “Are Disks the Dominant Contributor for Storage Failures?—A Comprehensive Study of Storage Subsystem Failure Characteristics,” in *FAST '08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, San Jose, CA, February 2008.
- [9] FAS6000 Series Technical Specifications: [http://www.netapp.com/products/filer/fas6000\\_tech\\_specs.html](http://www.netapp.com/products/filer/fas6000_tech_specs.html).
- [10] A.C. Rosander, *Elementary Principles of Statistics* (Princeton, NJ: Van Nostrand, 1951).
- [11] Bianca Schroeder and Garth A. Gibson, “Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?” in *FAST '07: Proceedings of the 5th USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA, 2007.
- [12] Storage Networking Industry Association Dictionary: <http://www.snia.org/education/dictionary/>.
- [13] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky, “Don’t Blame Disks for Every Storage Subsystem Failure—A Comprehensive Study of Storage Subsystem Failure Characteristics,” NetApp Research Paper, April 2008: <http://media.netapp.com/documents/dont-blame-disks-for-every-storage-subsystem-failure.pdf>.

MICHAEL DEMMER, BOWEI DU, AND  
ERIC BREWER

## TierStore: a distributed file system for challenged networks in developing regions



Michael Demmer is a PhD candidate at UC Berkeley. His research is on delay- and disruption-tolerant networking, distributed systems for unusual or challenged network environments, and application of technology in developing regions. He received his BS from Brown University.

*demmer@cs.berkeley.edu*



Bowei Du is a PhD candidate at UC Berkeley. His research is on distributed storage in delay-tolerant networks. He received his BS from Cornell University.

*bowei@cs.berkeley.edu*



Eric Brewer is a Professor of Computer Science at UC Berkeley who focuses on all aspects of Internet-based systems, including technology, strategy, and government. He leads the TIER research group on technology for developing regions, with projects in India, Ghana, and Uganda, and including communications, health, education, and e-government. He received an MS and PhD in EECS from the Massachusetts Institute of Technology and a BS in EECS from UC Berkeley and was recently elected to the National Academy of Engineering for leading the development of scalable servers.

*brewer@cs.berkeley.edu*

**TECHNOLOGY HAS A GREAT ROLE TO** play in developing regions, but we need approaches that can tolerate limited networking and power infrastructure. One promising model is to build applications around a file system interface that provides eventual consistency in these “challenged” network environments. Our resulting system, TierStore, hides much of the complexity of intermittency and simplifies the deployment of important applications such as email, Web caching, and wiki-based collaboration.

In many developing region settings throughout the world, there is an unmet need for robust information distribution applications. The limited communications infrastructure that exists in these environments means that simple information sharing systems can have a large impact. In fact, several projects have shown tangible results in the areas of health care, education, commerce, and productivity. As one example, data collection related to causes of child deaths in Tanzania led to a reallocation of resources and a 40% reduction in child mortality (from 16% to 9%) [1, 3].

However, the limited infrastructure also makes application deployment challenging. Wired networks are often either poor in quality or virtually nonexistent, cellular networks may be growing rapidly but remain a largely urban and costly phenomenon, and satellite networks provide good coverage but are prohibitively expensive. Many of these networking approaches further suffer from periodic outages owing to unreliable grid power. Thus any software system targeted toward these environments must deal with intermittent connectivity and potentially long-lasting network partitions and failures.

In response to the combination of application needs and the complexity of programming intermittency-tolerant applications, we have developed a distributed storage system called TierStore [4]. TierStore is a new approach to designing and deploying information distribution applications which aims to overcome the connectivity challenges in developing countries, while at the same time making it easy to port existing applications and develop new ones.

This work is part of the Technology and Infrastructure for Emerging Regions (TIER) [7] research effort at UC Berkeley. The aim of the TIER project is to address challenges in bringing the information

technology revolution to the masses of the developing regions of the world. Unfortunately, most projects that aim to do this today rely on technology that was developed for the affluent world, yet these imported technologies fail to address key challenges in cost, deployment, power consumption, and support for semi- and illiterate users. Instead, our approach is to explore the development of novel solutions to technical challenges that explicitly take the needs of developing countries into account.

---

## Background

---

In developing TierStore, we were inspired by several existing projects that deal with poor and intermittent connectivity. For example, the Wizzy Digital Courier system [9] distributes educational content among schools in South Africa by delaying dialup access until night time, when rates are cheaper. As another example, DakNet [6] provides email and Web connectivity by copying data to a USB drive or hard disk and then physically carrying the drive among locations that have no traditional network connectivity, sometimes via motorcycles. Finally, the TEK [8] disconnected Web search engine allows users to search the Web using SMTP as the underlying protocol, which can buffer communication across network outages.

These examples help underscore the value of information distribution applications in developing regions, but they all essentially started from scratch and thus use ad hoc solutions with little leverage from previous work. Instead, the goal of TierStore is to be a general-purpose framework that can abstract away most of the complications related to working with intermittent networks.

Asynchronous	Disconnection-tolerant	Synchronous
	offline WWW	
email	information portals	
voicemail	e-government services	VOIP
bulk data copy	survey/data collection	video chat
	medical records	

**TABLE 1: AN APPLICATION TAXONOMY WITH RESPECT TO INTERMITTENCY**

Table 1 gives a rough breakdown of example applications and their behavior with respect to intermittent networks. At the end points, “asynchronous” applications already work well in disconnected environments, whereas “synchronous” applications fundamentally require end-to-end connectivity and just cannot function during network outages. However, the large class of “disconnection-tolerant” applications can potentially work well in a disconnected fashion, yet their implementations are limited by the requirements of the underlying software platforms on which they are implemented. For example, many such services require Internet connectivity simply because they have been written as Web applications, whereas they could potentially function well even when disconnected. Our goal with the TierStore system is to make it easy to adapt these applications to work well in an intermittent environment.

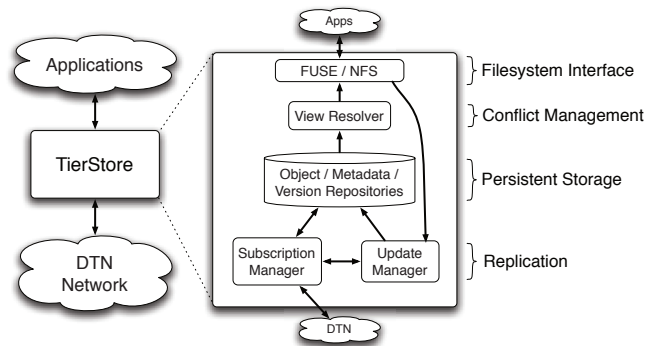
One step toward this goal comes in the form of Delay-Tolerant Networking (DTN) [2]. DTN is a new approach to networking in challenged environments that seeks to address the shortcomings of traditional Internet protocols in some scenarios. Specifically, there are many cases in which it is difficult to maintain the kind of reliable, low-latency network connection needed by TCP/IP-based protocols. In these cases DTN can route network messages across a variety of different transports such as peer-to-peer wire-

less connections, dialup links, or physically carried flash drives or PDAs. Furthermore, DTN is based around application-defined data objects called “bundles” (not packets or circuits) and can deal with outages by storing messages in the network core to wait for connectivity to be restored. DTN also offers new approaches to routing, quality of service, and reliability based on custody transfer which help to deal with many of the problems that exist in challenged network environments.

Yet applications need more than just a messaging service. Running while disconnected implies that applications need to have local storage to respond to user requests. Distributing information between instances mandates conventions for object naming and organization to ensure that multiple sites remain in sync with each other. Operations that modify the system state need to be logically ordered, and potentially conflicting operations need to be identified and resolved. Perhaps most importantly, adapting existing applications to DTN environments would require significant rewriting to use the DTN-specific messaging APIs. TierStore is aimed squarely at addressing these application needs while leveraging the existing advantages of the DTN framework.

## How TierStore Works

To address these needs, TierStore implements a replicated file system interface, and applications interact with the system using the standard POSIX APIs. This decision means that existing applications that are already written to use the file system for interprocess communication can be adapted with relatively few changes, while developers creating new applications can leverage their familiarity with the existing APIs and use a wide range of programming environments and languages to interact with the system. Figure 1 shows the TierStore system components.



**FIGURE 1: TIERSTORE SYSTEM COMPONENTS**

Unlike NFS or CIFS, there is no central server that stores file data. Instead, each TierStore node keeps a copy of the data and uses a lazy distribution protocol to forward updates among nodes using DTN bundles. This means that replicated file data is available for access even when the network is down, and local filesystem interactions need not consume valuable bandwidth.

### REPLICATION

File system modifications, such as writing some data to a file, are encoded as update messages. These updates are immediately applied to the local node so that applications see the effects of their operations in the file system. They

are also forwarded to the subscription manager that determines how to distribute the updates to other nodes.

To enable fine-grained data sharing, the files and directories in the TierStore file system are divided into non-overlapping subsets called publications. Publications define the units of replication between TierStore installations, and they are defined in application-specific ways. For example, an individual publication might be a user's mailbox, files from a particular Web site, or a set of data collection samples from a specific region. TierStore nodes then subscribe to a set of publications, indicating that they want to receive updates to the relevant files. Subscribing is the TierStore equivalent to mounting a portion of the file system, and therefore file data in a publication is replicated only to the set of subscribed nodes.

TierStore uses DTN for all internode communication, meaning that it can leverage its range of network transports and in-network message queuing features. Thus TierStore need not be concerned with the details of how updates are communicated, but instead it can queue an update message for transmission whenever the network becomes available, using whatever transmission mechanism is most appropriate for the environment.

To help distribute data efficiently over low-bandwidth links, each TierStore node is configured as part of a multicast-like distribution tree in the DTN network. Each publication can be thought of as a multicast group, so updates need only be transmitted once across a network link in the tree and are reforwarded down the tree, eventually reaching all subscribed nodes. In our early deployments, this distribution tree was manually configured, as the number of nodes was fairly small, but we are currently working on a new DTN multicast implementation to automate this process.

## HANDLING CONFLICTS

Alice	Bob
<i>Network is unavailable ...</i>	
\$ echo "red" > /tierstore/foo	\$ echo "blue" > /tierstore/foo
\$ ls /tierstore	\$ ls /tierstore
foo	foo
<i>Network is available ...</i>	
\$ ls /tierstore	\$ ls /tierstore
foo foo.#bob	foo foo.#alice
\$ cat foo	\$ cat foo
red	blue
\$ cat foo.#bob	\$ cat foo.#alice
blue	red

**FIGURE 2: DEFAULT CONFLICT HANDLER. DISCONNECTED USERS ALICE AND BOB MAKE EDITS TO THE SAME FILE. WHEN THEY RECONNECT, THE OTHER'S EDITS WILL BE VISIBLE AS A CONFLICT FILE IN THE FILE SYSTEM.**

Since TierStore nodes might be disconnected for long periods of time, they must be able to modify the filesystem state while disconnected, so applications need some way of handling concurrent updates. Yet long outages mean that traditional approaches such as file locking will not work well. Instead,

we allow arbitrary operations to occur and then detect (and possibly resolve) conflicts when nodes return into connectivity.

The first (and best) way to handle conflicts is to avoid them in the first place. TierStore only considers conflicts on a per-file basis, so updates to different files or directories are independent and do not conflict. Thus applications that partition their data into separate directories or use uniquely named files that are not updated at different parts of the network are thereby conflict-free. Many of the applications we have ported to TierStore naturally fall into this category.

When conflicts are unavoidable, applications can register custom handlers to resolve the situation. These handlers are able to look at conflicting versions of a file and arbitrarily rename, merge, or modify them to deal with the conflict. If there is no custom resolver, a default policy appends each conflicted filename with `.#X`, where X is the identity of the node that generated the conflict. This approach allows applications to see both versions of the conflicted file, similarly to how CVS allows multiple versions of a file to simultaneously exist on different branches. Applications can then resolve the conflict later at any point in the network.

However, one subtle aspect of this default policy is that file operations that occur at a particular node are presented unmodified to applications that are running at that node (i.e., without the `.#X` extension). This does mean that the displayed filesystem structure can vary at different locations in the network, but it also has the important side effect that nodes always “see” the files they have generated and modified locally, regardless of any conflicting updates that may have occurred at other locations (see Figure 2). This is an important decision that helps when porting unmodified applications, since their local file modifications do not suddenly disappear if another node makes a conflicting update to a file. It also means that application state remains self-consistent even in the face of conflicts and, most importantly, is sufficient to handle many applications without needing to write a custom conflict resolver.

---

## Using TierStore

---

We have adapted several commonly used applications for use with TierStore to validate our system: an IMAP email service, a shared offline Web cache, and a shared wiki collaboration system. These three applications represent a range of requirements. Email in IMAP folders is accessed by a single user, but the folders may be replicated to many different computers. A Web cache is shared by many users but is read-only. A wiki system is shared by many users and has the potential for many conflicting writes from users editing overlapping parts of the wiki.

To support a shared email service with the TierStore system, we used a stock IMAP server configured to store mail content in maildir format. The maildir storage format is ideal for use with TierStore because each mail message is stored as a separate file and the metadata associated with the message is encoded in the message file name. Mail folders are thus mapped to directories in the TierStore file system, and each user mailbox is placed in a separate publication. This allows each computer in the network to elect to replicate only some of the users’ mailboxes. To handle benign conflicts in state (e.g., flagging a message on one computer and marking it read on another), we wrote a conflict resolver that takes conflicting state flags and resolves them to be the union of the flags.

For push-based shared Web cache functionality, we took an existing offline-enabled Web cache, WWWOFFLE, and configured its cache directory to point to a TierStore shared directory. For a selected set of Web sites, we populate the cache directory with a Website crawl of commonly accessed reference Web sites. This model suits the needs of organizations that create local mirrors of online references such as Wikipedia. Using the TierStore system allows administrators to integrate Web content mirrors from many alternative sources into the same system. A bulk load of content via media such as DVDs can be supplemented by small incremental deltas pushed over conventional Internet access.

Finally, we have ported an existing piece of wiki software, PmWiki, to TierStore as well as our created own wiki software [5], which leverages TierStore as its storage back end. PmWiki stores its pages as individual files on the local file system. This matches the semantics of TierStore quite well, because conflicts in the file system map to edit conflicts at the page level; page-level conflicts are well understood by users of wikis, since they already occur because of delays between when a page is loaded in the browser and when the edit is saved to the server. Using the default conflict resolver, users of PmWiki will see edit conflicts as specially named pages on the wiki site. In our own wiki software, we implemented a resolver that performs a text-based merge of the conflicting pages.

---

## Next Steps

---

On the research front, we are continuing to push TierStore in two directions. The first is focused on the networking layer to develop a robust publish/subscribe-based distribution network that functions well in DTN environments. The next is focused on developing an easy-to-use SQL interface to support disconnection-tolerant Web applications that interface with a database instead of the file system.

We are also continuing to work on several TierStore deployments in developing countries. One example is supporting community radio stations in Guinea Bissau, a small West African country characterized by a large number of islands and poor infrastructure. For many of the residents, their main information source comes from small radio stations that produce and broadcast local content. TierStore is being used to help bridge the communication barriers between the different islands and distribute content from these stations throughout the country. We are also currently doing a pilot deployment in Senegal, where TierStore will be used to link student medical records between two schools and a local hospital.

In general, our initial results from working on TierStore are encouraging, and we hope to gain additional insights through more deployment experience.

---

## REFERENCES

---

- [1] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedeveschi, J. Pal, R. Patra, S. Surana, and K. Fall, "The Case for Technology in Developing Regions," *IEEE Computer* 38(6), 25–38 (June 2005).
- [2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," RFC 4838, April 2007.
- [3] D. de Savigny, H. Kasale, C. Mbuya, and G. Reid, in *Fixing Health Systems* (International Development Research Centre Books, Ottawa, 2004).

[4] M. Demmer, B. Du, and E. Brewer, "TierStore: A Distributed File System for Challenged Networks in Developing Regions," in *FAST '08: 6th USENIX Conference on File and Storage Technologies*, pp. 35–48 (February 2008).

[5] B. Du and E. Brewer, "DTWiki: A Disconnection and Intermittency Tolerant Wiki," in *17th Annual International WWW Conference* (April 2008).

[6] A.S. Pentland, R. Fletcher, and A. Hasson, "DakNet: Rethinking Connectivity in Developing Nations," *IEEE Computer* (January 2004).

[7] Technology and Infrastructure for Emerging Regions (TIER) Research Group: <http://tier.cs.berkeley.edu/>.

[8] W. Thies, J. Prevost, T. Mahtab, G. Cuevas, S. Shakhshir, A. Artola, B. Vo, Y. Litvak, S. Chan, S. Henderson, M. Halsey, L. Levison, and S. Amarasinghe, "Searching the World Wide Web in Low-connectivity Communities," in *Proceedings of the 11th International World Wide Web Conference, Global Community Track* (May 2002).

[9] Wizzy Digital Courier: <http://www.wizzy.org.za/>.

# *Save the Date!* 22nd LARGE INSTALLATION LISA'08 SYSTEM ADMINISTRATION CONFERENCE

November 9–14, 2008, San Diego, CA

- 6 days of training by experts in their fields
- 3-day technical program
- Keynote Address by Sean Dennehy and Don H. Burke, Intellipedia, U.S. Central Intelligence Agency
- Plenary Session by Bruce Schneier, Founder and CTO, BT Counterpane
- Invited talks by industry leaders
- Refereed Papers, Guru Is In Sessions, Workshops, and Work-in-Progress Reports
- Vendor Exhibition
- And more!

[www.usenix.org/lisa08/jlo](http://www.usenix.org/lisa08/jlo)



INTERVIEW BY MARGO SELTZER

## the present and future of SAN/NAS

### INTERVIEW WITH DAVE HITZ AND BRIAN PAWLOWSKY OF NETAPP

Dave Hitz is one of the founders of NetApp. At NetApp, he has been a programmer, an evangelist, and the VP of Engineering. Now he focuses on strategy and culture as the Chief Philosophy Officer, asking the timeless questions: Who are we? Where did we come from? Where are we going? (See [blogs.netapp.com/dave](http://blogs.netapp.com/dave).)

*Dave.Hitz@netapp.com*

Brian Pawlowski is Senior Vice President and Chief Technology Officer at NetApp. Since joining NetApp in 1994, he has been involved in the design of high-performance, highly reliable storage systems.

*Brian.Pawlowski@netapp.com*

Margo I. Seltzer is a Herchel Smith Professor of Computer Science and a Harvard College Professor in the Harvard School of Engineering and Applied Sciences. Her research interests include file systems, databases, and transaction processing systems. She is the author of several widely used software packages. Dr. Seltzer is also a founder and CTO of Sleepycat Software, the makers of Berkeley DB.

*margo@usenix.org*



DAVE HITZ



MARGO SELTZER INTERVIEWING BRIAN PAWLOWSKY

**MARGO:** MY FIRST QUESTION IS THAT people use this term “network storage” but I think different people use it to mean different things, so in order to lay some context I’d like you guys to tell me what you think network storage is all about.

Dave: I think we should start with the technical answer.

Brian: Storage that’s on a network?

Dave: There’s a whole bunch of different dimensions when you look at network storage. Brian gave the answer: it’s storage over a network. Yes, but does a Fibre Channel network count as a network or does network storage only include Ethernet? Sometimes people say network-attached storage, which almost always means Ethernet, but is that only file-based protocols or would iSCSI be a form of network-attached storage? And so you can get into really funny kinds of technical semantic arguments about whether a particular type of storage like iSCSI is a form of network-attached storage or not, so I’m not that interested in the vocabulary of it, but I think that there’s two dimensions that matter. The first dimension is, “Are you using Ethernet, or are you using some other form of networking like Fibre Channel?” That’s important dimension number one; and then the other interesting dimension is, “Is it block-based storage like Fibre Channel or iSCSI (basically, read a block, write a block, talk directly to the disk drive), or is it file-based storage like NFS or CIFS?”

**Margo:** So let’s look at each of those dimensions. Why does it matter whether you’re talking over an Ethernet or something else?

Dave: From a technical perspective, technical people tend to look at the difference between Fibre Channel and Ethernet and they say it’s not that big of a difference. What really matters is where I plug into the operating system, and plugging in at the block device layer is an important distinction as opposed to plugging into the file system layer.

Margo: So that goes back to your other dimension, and I guess the question is, “Are those dimensions really separable, then?”

Dave: The block file really is where you plug into the OS, and technical people almost always argue that that’s the much more important distinction. Business people tend to focus on Fibre Channel versus Ethernet, and the reason business people tend to focus on that is because they worry about things like capital expenditure. If they’ve spent millions of dollars on a Fibre Channel infrastructure and they’re about to buy more storage, they care a whole lot whether that new storage is going

to plug into the millions of dollars' worth of Fibre Channel infrastructure they already bought or whether they're going to plug it into their corporate Ethernet infrastructure, in which case they may need to beef that up.

Brian: So there is a historical artifact here that I think was interpreted as a technical truism: that essentially the evolution of block storage went into the Fibre Channel network and Fibre Channel SANS, which were much better than using run-of-the-mill Ethernet and TCP networking, which was used for low-grade file sharing along the lines of NFS or things that you see in the Microsoft Windows network. And there was this line between the two that was more an artifact of the evolution of the two technologies than a technical requirement. Where we are today is just a total blur, first with iSCSI going over TCP/IP, Fibre Channel protocols being put over Ethernet, and block protocols being tunneled through Fibre Channel networks, and Infini-Band just playing merrily between the two camps.

Dave: This has been something that evolved over time. Ten years ago it was pretty clear where Fibre Channel would make sense and where Ethernet would make sense. If you were looking at heavy-duty database business kind of apps you definitely wanted a Fibre Channel. If you were looking at more distributed users' home directories, you definitely wanted NAS, and it was pretty distinct.

Margo: Why? Is it again just—

Dave: Because Ethernet reliability was not as strong and because the applications had not yet been modified to support NAS. If you went and talked to Oracle they would give you a list of reasons why NFS was not a good solution for running your databases. So 10 or 15 years ago there really was a strong distinction.

Brian: And even if it ran over NFS, Oracle would say they wouldn't support Oracle over NFS, which was a deal breaker for a lot of customers even if they said, "But we just ran the application over NFS and it works fine."

Dave: Oracle hadn't chosen to train their problem-solving people on those technologies and so they couldn't really help you. What happened is that Ethernet got to be much, much faster and better. Then Oracle said, "You know, this NFS stuff can save people money." So if you look at it like a Venn diagram of what are all the problems you could solve with NFS and what are all the problems you could solve with SAN, 10 years ago they were disjoint sets. There was not really any overlap. Today for the vast majority of things you might consider using storage for, you could use either one. It's gone to a Venn diagram with 90% overlap.

So from a business perspective, what I tend to believe is whatever you're already doing is probably the cheapest thing to keep doing. From a technical perspective, if you've got the opportunity to come in and redesign a bunch of stuff from scratch—not always but 80 or 90 percent of the time Ethernet storage, either NAS or iSCSI, is almost always going to be easier to manage and lead to lower cost.

Margo: So we can take away from this that in some sense these decisions are no longer important. It used to be that when I wanted storage I went and I bought the best price-performing disk I could. And it's no longer a pure price-performance choice in storage, so what are those other characteristics that you started alluding to and what are the value adds that storage manufacturers are really going to have to compete on?

Dave: Let me start top down. It's humbling as a storage vendor to recognize that CIOs do not care about storage. CIOs have some list of business prob-

lems that they want to solve and in general each of those business problems links to a particular application (e.g., all the employees need to be able to send each other email). Okay, we've chosen Exchange and so the CIO's top-level concern then is, how do I run Exchange—or, if we are going to balance the books, how do I run Oracle's financials? The more that a storage vendor can talk to the CIO about how its storage makes some kind of difference for running that application, the better off you are as a storage vendor. So—your face is all scrunched up.

Margo: That makes it sound like your value-adds are all application-specific and I'm going to claim that there's got to be a set of common value add-ons that you can argue will help your Exchange server and will help your financial apps and will help something else and that you can't possibly run a business having to argue each individual application independently.

Dave: There are common technologies that can help a lot of different apps, but I'll tell you that when you get into actually working with someone doing an Exchange deployment versus an Oracle deployment, they care about fundamentally different things. Let me use Exchange as just a really specific example. One of the things that people have noticed in Exchange deployments is that the Exchange database tends to get corrupted. So in an Exchange world, Exchange administrators care a lot about, "How do I get back to the earlier version of the stuff I had that used to be good?" And snapshots are a beautiful tool for doing that and so you can get back to that earlier version—and the more automated the better, right?

Think about the challenge in the real data center: You've got an Exchange administrator who typically doesn't own his own server, and there's a server administrator who typically doesn't own the network to the storage, and there's a Fibre Channel or an Ethernet administrator; and then down the line somewhere further on there's a storage administrator. Often each of those people reports to a different director and sometimes a different VP. The poor Exchange guy is just trying to get his database back the way it used to be, right? If you can somehow work with that Exchange guy and say, "Look, here's a tool that lets you do all this stuff," and now your Exchange environment is back up and running again without having to have even talked to the storage guy—that's a whole different model.

In Oracle, on the other hand, one of the big challenges is that people are always running test and development environments. They're not so worried about whether the database is corrupt, but they say, "I've got this giant production database that I'm not allowed to touch but I'm doing some little tweak in the customization that I have for SAP, say, or Oracle financials and I wish I had a playpen I could work in." Snapshots, writeable snapshots, or clones are a great tool for that. You really do have to look—there's a bazillion apps but you look at the combination of the major apps—the Microsoft Suite, the Oracle, the SAP, and VMware as an emerging one that has common characteristics: test and development environment, sort of the typical UNIX home directory. You look at that set of apps and optimize for them based on a common set of underlying capabilities. How do you virtualize your storage more? How do you create snapshots? How do you do thin provisioning? How do you do clones? You've got lots of data here, so how do you get it to there? De-duplication—those are the kinds of building blocks.

Brian: I want to make a comment, because Dave just kind of glossed over a large part of our history. It wasn't that there weren't a lot of NFS servers out there; one of the key differentiators was basically the instantaneous copying of an entire file system at essentially zero cost. And that shattered the Exchange deployment preconceptions about the time required for backup

and the number of recovery points you could have in your Exchange environment: when it divoted on you, what you hope to recover, and how fast the recovery was. Snapshots just blew away the traditional methods of doing backup to tape or any other means. Fast-forwarding, we come to that experience from the late nineties when we started seeing vast incursions into Exchange deployments for our product: a lot of times our customers were coming to us kicking and screaming about many different applications before we were giving them the tools around it.

I think there was a recognition that it's not the primary copy of data that's what is most important and of most concern to people in an organization. It's the secondary copies of data—the recovery points, the archives, etc.—and the ability to leverage and reuse data that has to be managed, because of the cost of making those copies for different purposes but also because of their usefulness in terms of business continuity. The primary copy is what everybody was designing around and everything was optimizing for. But what came circling back to everybody was the cost and value of the secondary copies, around which our fundamental technology enables interesting processes and techniques, regardless of how you access the data. How do we do data management with snapshots? How do we do disaster recoveries? Secondary copy management applies to Fibre Channel SANs and to NAS and file access.

Margo: So what I can take away is that snapshots were a truly fundamental value add that helped you differentiate early and that continued to be leveraged to solve a bunch of different business problems.

Brian: Yes.

Margo: What have you done for me lately? So snapshots were a great idea but it's 2008 and what's the next piece of core technology?

Dave: There are a handful of different ones that we can work our way through. One that I think is interesting and people don't understand the ramifications of as much as they might is RAID 6 or RAID DP—the ability to allow any two disks to fail instead of just one. When I say RAID group, I mean one parity drive with however many different disk drives; and as the number of disks gets bigger, each disk drive itself gets bigger. Say you put 10, then your overhead's 10%. You put 20: your overhead is 5%. The more disks you put in there, the more data you have to read to reconstruct a bad one. In fact, if you look at the math, disks are getting so big these days that just looking at the bit failure rate (with the current size of disks, you build a standard RAID group of 7 disks), you would expect to see failures 1% of the time on your RAID reconstructions just as a result of the raw bit failure rate of the underlying disk drives.

So imagine that doubles again because, remember, if one drive fails you have to read all of the other drives. So it was already getting bad for regular disk drives; the real challenge is what about those cheap ATA drives? Wouldn't it be nice to be able to use SATA drives? These are both bigger and slower, so it's going to take longer and be less reliable.

Margo: It's the next generation; now we're going to replace arrays of moderately inexpensive disks with arrays of really super cheap disks that are unreliable and so therefore we're going to have to go to even bigger parity.

Dave: Absolutely. Look at EMC, the way that EMC enabled the transition from the DASD style of drives to the cheaper emerging, more commoditized drives of that era was through the invention of RAID 4. I do think that ATA or SATA drives enable the next generation of this transition.

Brian: I want to connect the two topics of snapshots and RAID DP. The strength of snapshots was basically the commoditization and making snapshots available for everyone at no cost essentially compared to all other solutions. The clever part about RAID DP—having double disk protection—was not a new invention. RAID 6 was certainly okay, but no one could ever enable it because they would regret that decision forevermore because of the performance. The really clever part about RAID DP is that it was enabled with no more than a 1% to 3% performance drop versus single-parity disk protection on all our storage systems, to the point where we ship it out by default in all our systems, from our low-end SMB product, to the S500, and up to our high-end systems.

Dave: Another zone of technology is the data-replication technology that NetApp has—it turns out that snapshots are a beautiful starting point for replicating data to a remote location. The reason for that is one of the biggest challenges of replicating data: If the data that you're copying is changing underneath you and you're moving the blocks, depending on what order the data changes and depending on what order the blocks move in, you may get corruption on the copy that's of a form that even FSK can't fix. You've got to be really careful, so in a lot of situations when you do bulk copies to a remote location, you may even have to quiesce the system. If you have snapshots as a foundation you can just copy all of the blocks in a snapshot and you know that a snapshot is static and those blocks are locked down so that changes don't go back on top of those blocks.

Margo: You said “just the same,” but for using snapshots there's a time delay.

Dave: Sorry, “just the same as it looked at some point in the past; just the same but with a delay.”

Brian: And with a well-defined consistency point.

Dave: Yes, with a well-defined consistency model. What a lot of customers started saying as they moved away from tapes for backup was, “I like that model, but on the remote machine it's probably made with cheap boatloads of ATA and probably a lot more drives per head, so the performance wouldn't be there necessarily for running an app but just kind of for reference.” They want to keep the snapshots for a lot longer: On your primary systems you only keep snapshots for a day or two or maybe a week, but on your remote system, what if you could keep snapshots for literally a year or multiple years? We started getting banks looking at this and saying tape just isn't scaling; disks are getting bigger and faster—faster than tapes are getting bigger and faster, especially the faster part. A lot of banks have a regulatory requirement to keep data around for seven years and so they started saying “Can we have seven years' worth of snapshots, please?”

Margo: And it seems that when you get into that model of, “Okay, I need my snapshots for seven years,” and they're going to be spinning, then I also have a disk lifetime problem and the disks don't necessarily last seven years, and so I also have a problem of refreshing my disk farm as it's running with these unreliable disks.

Dave: Sure. The capital lifetime of this equipment for most customers is three or four years. Some people keep it for an extra year, but really three to five years is typically the replacement cycle. So keeping a snapshot for seven years, that snapshot may not be living on the same system or the same disks, but that snapshot has the same bytes in the same file system organized structure. The snapshot may live for much, much longer than any of the physical components live.

Margo: As a customer, when I'm refreshing my vault, I assume I want to keep my vault spinning, so am I doing sort of a real-time online migration to my new vault and then sort of replacing incrementally, or am I really doing, "Okay, time to copy the vault"?

Read the complete interview transcript online at [www.usenix.org/publications/login/2008-06/netappinterview.pdf](http://www.usenix.org/publications/login/2008-06/netappinterview.pdf).

*Save the Date!*



## 8TH USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION

December 8–10, 2008, San Diego, CA

### The 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)

brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software. The OSDI Symposium emphasizes both innovative research and quantified or illuminating experience.

The following workshops will be co-located with OSDI '08:

Fourth Workshop on Hot Topics in System Dependability (HotDep '08),  
December 7

<http://www.usenix.org/hotdepo8>

First USENIX Workshop on the Analysis of System Logs (WASL '08),  
December 7

<http://www.usenix.org/waslo8>

Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysMLo8), December 11

<http://www.usenix.org/sysmlo8>

[www.usenix.org/osdio8/jlo](http://www.usenix.org/osdio8/jlo)

ANDREW BROWNSWORD

## driving the evolution of software languages to a concurrent future



Andrew Brownsword is Chief Architect at Electronic Arts BlackBox in Vancouver, Canada. He has been with the company since 1990 and has a BSc in Computing Science from the University of British Columbia.

[andrew@brownsword.ca](mailto:andrew@brownsword.ca)

Concurrent: existing, happening, or done at the same time.

**CONCURRENCY HAS BECOME A HOT** topic in the past few years because the concurrency available in hardware is increasing. Processor designers have reached a point where making sequential tasks take less time has become impractical or impossible—the physical limits encountered demand engineering tradeoffs. But writing software that takes advantage of concurrency is hard, and making that software perform as well on different CPU architectures is all but impossible. In this article, we will explore the reasons why this is currently true, with specific examples, and will consider how this evolution represents a changing paradigm that renders the traditional imperative programming model fragile and inefficient.

For the past 20+ years, hardware designers have been using concurrency in the form of pipelining, superscalar issue, and multi-transaction memory buses to improve the apparent performance of what appears to be sequential hardware. Since about the late 1990s most microprocessors have also included SIMD instructions, which are typically capable of 4 FLOPs (Floating Point Operations) per instruction. More recently, some processors support multiple threads per core (from two in Intel's Hyperthreading, up to eight in Sun's Niagara architecture), and now processors with two or four cores are becoming the norm. Michael McCool's April 2008 *;login:* article provides an overview of these techniques.

### Computational Efficiency

Most high-performance modern hardware these days is theoretically capable of about 10–20 FLOPS for each floating point value moved through the processor. In terms of latency, approximately 400–4800 FLOPS could theoretically be done in the time it takes to fetch a value directly from memory. These are theoretical peak computation rates, based on the four-way SIMD capabilities that most processors have now. Theoretical rates are not reached in practice, so what can we really expect to achieve and what do we see in practice?

How efficiently a processor executes is a function of the details of the processor's operation, its memory subsystem, and the software that is running. Typically all operations are scalar, because that is how most languages are defined. ALGOL set the pace, and most imperative languages since then have been embellishing the basic model. They embody the ideas of the sequential Von Neumann architecture and are notations for describing a sequence of scalar operations. These operations are usually dependent on the output of operations that came immediately before, or nearly so.

Also, typical code sequences are filled with decision points and loops, which appear as branch instructions that disrupt the efficient flow of instructions. Branch frequency and data dependencies in typical code are a frequently measured metric by hardware and compiler developers. In the mid 1990s, IBM found during the development of the PowerPC 604 that branches occurred, on average, once every 6 instructions. This makes it largely unproductive to have hardware dispatch more than about 3 or 4 instructions per clock cycle. To this day most hardware is aimed at 2- to 4-way dispatch, which is unsurprising, since software hasn't changed substantially. More recent tests on an in-order processor showed that most of the software for gaming platforms averaged about 10% of the potential instruction dispatch rate. And very little of that software was using the considerable SIMD capabilities of the hardware, leaving the realization at less than 3% of the processor's theoretical computational capability. Instead of the theoretical potential ~25 GFLOP, less than 1 GFLOP was realized. Out-of-order-execution processors will do somewhat better than this, but usually only by a factor of 2 or 3.

In recent years both the number of cores in one system and, on some cores, the number of threads executing on each core have increased beyond unity. I will ignore the distinction between hardware threads sharing a core and multiple cores for the rest of this article and will refer simply to "threads."

---

## Memory and Threads

---

Multiple threads must share the *system* in which they exist. The most important shared resource is main memory (RAM). In most systems there is a single large pool of RAM. The system may also have a GPU processor, and some architectures give the GPU its own pool of RAM. Some systems partition the CPU memory by attaching parts of it directly to each processor; this is called NUMA (Non-Uniform Memory Access). How each processor accesses data at a given memory address and how long it takes to do it depend on where that memory is physically located.

The sharing of memory is complicated by the fact that processors use on-chip caches to speed up memory access. When the cache isn't shared by all threads, the potential exists for a given memory location's actual current state to be sitting in one thread's cache when another one needs it. Most hardware implements "cache coherency," which is a mechanism that tracks where each memory location's actual state is and retrieves it from that location when requested. Having multiple threads reading the same location is dealt with by having multiple copies of the state in the various caches. Writing to the same location is problematic, however, because a single current state must be kept up to date and it is usually placed in the cache of the thread that most recently modified it. If more than one thread is continuously updating the same location at the same time, considerable intercache traffic may result.



From a developer's perspective the problem with multiple threads, at least in an ALGOL-like language, is that the program must be explicitly written to take advantage of them. Given a program written for a single-processor machine, all but one thread will sit idle unless the operating system has something for additional threads to do. Typically the OS has enough to keep a second thread at least partially busy, and some OS services are now internally moving computation to other threads (graphics, movie playing, animation, etc.) if those services are in use. If the hardware has more than two threads, however, then they are going to be doing little to improve your software's performance. As a result, if you are using 3% of one thread's potential and you have a four-core machine, for example, you are now only using <1% of the system potential. In theory you could speed up your software by a factor of over 100 or be doing 100 times as much computation on the same hardware.

---

## Programming for Concurrency

---

So how do you take advantage of this ability of hardware to operate concurrently? Broadly speaking, there are two kinds of available concurrency: instruction-level and thread-level.

Instruction-level concurrency has largely become the domain of the compiler, but that doesn't mean there is nothing you can do about it. The choice of language you use and how you use it has an enormous impact. As already mentioned, most current languages embody a fundamentally scalar and sequential programming model. Some compile to an intermediate form that also embodies a simple sequential machine, and the need for efficient JIT severely limits how the compiler can optimize. Fully natively compiled languages or those with more powerful intermediate forms may have compilers that can perform aggressive optimizations (such as auto-vectorization) in specific circumstances, but these techniques tend to be fragile and provide limited results. In C and C++, the language provides fairly low-level-detail control over the emitted instructions, and the compilers often support hardware-specific language extensions to access SIMD instructions and other unique hardware features. With careful, labor-intensive techniques, highly specialized platform-specific code can be written to dramatically improve performance. This kind of code optimization requires substantial expertise, detailed knowledge of the hardware platform, and a great deal of time, and it incurs substantial maintenance costs. On modern hardware it commonly delivers 4–20x performance improvements, raising the 3% utilization into the 10–60% range. Notice that this is as good as or better than the improvement you might expect by going from a single- to a many-threaded processor. Not all problems can be optimized in this fashion, but more are amenable to it than most developers seem to think. This kind of data-parallel solution also tends to be more amenable to being divided across multiple threads, making it easier to achieve thread-level concurrency. Unfortunately, the aforementioned problems make the process of experimentation and exploration to find efficient new data-parallel solutions very difficult and expensive.

Achieving a high level of instruction-level concurrency boils down to writing code that fits into a set of constraints. The precise definition of these constraints depends on the hardware, but there are some general principles that apply. Choose data structures and access patterns that are uniform and predictable, organize data so that access to it is spatially and temporally dense, choose data structure alignments and sizes, apply precisely the same sequence of operations to many data elements, minimize data dependencies

across parallel elements, define interfaces where each interaction specifies a large amount of work to be performed, and so forth.

Unfortunately, most programming languages do nothing to aid the programmer in these pursuits, if they allow these things to be under explicit programmer control at all. Even worse, code written in these languages contains too much detail (i.e., the semantic information available to the compiler is at a very primitive level) and this limits what the compilers are capable of and what they are permitted to do by the language rules. There are a few languages and alternative notations available (Michael McCool's own RapidMind being one proprietary example; others are StreamIt, Intel's Ct, etc.), but they are far from pervasive and thus far there is no widely available standard that integrates tightly enough with the standard environments (such as C/C++) to be adopted. Tight integration and, where possible, familiar syntax are essential for practical efficiency reasons, for programmer comfort, and for gradual adoption by an industry heavily invested in existing languages. Exactly what such a language should look like is open to debate, but my opinion is that burdening an existing grammar of already excessive complexity is not the correct solution.

---

## Thread-level Concurrency

---

Thread-level concurrency tends to receive more attention, largely because it is easier to see and understand. It is accomplished in most languages by calling a threading interface provided by the OS, including it in a support library, or as a language feature. These vary in their details and features. The detailed semantics and capabilities of how threads work between platforms and interfaces vary significantly. How they get scheduled to run, whether priorities can be set, how the threads interact based on these priorities, whether a given software thread will stay on a given hardware thread, how long it will run without being interrupted, and so on can show up latent bugs and result in noteworthy performance differences that are hard to diagnose.

Threads running concurrently will inevitably want to interact and will therefore need to access some piece of shared state. This generally isn't a problem until one or more of the threads want to modify the shared state. Then we run into a problem called the "lack of atomicity." Most operations are actually composed of several hardware operations. Incrementing an integer variable, for example, reads the value from memory into a register, performs the add operation, then stores the value back to memory. Even with hardware that has an instruction that appears to add a value directly to memory, it in reality implements this internally as a read-modify-write sequence. When each of these operations is a pipelined instruction, which is itself broken down into many stages, it should be clear that there are a lot of steps involved in doing even this most trivial of examples. When these steps are happening in a machine where there are additional threads sharing the memory location being modified, the possibility exists of conflicting updates. Exactly what happens to the value depends on the operation, its hardware implementation, and the precise timing of each incident of conflicting change. This means that even this trivial example can do something different every time it is executed, and even if timing were somehow stable, then executing on different (even nominally compatible) hardware may impact the outcome.

Many thread interfaces provide some basic "atomic operations." These functions use capabilities provided by the processor to ensure atomicity for this kind of simple read-modify-write operation. Usually the hardware provides

a base mechanism to use instead of the atomic operation itself. This is usually either a compare-and-set operation or a reserve-and-conditional-store operation. I'm not going to describe what these two things are here, but the salient point is that using them correctly and ensuring atomicity is significantly more expensive in terms of complexity, instructions, performance, and programmer knowledge than just a simple nonatomic operation. They also don't guarantee atomicity across a series of operations.

An important point to note about primitive hardware synchronization operations is that they cause the hardware to synchronize. This seemingly trivial point has unfortunate consequences. It usually means that at least part of the hardware must stop and wait for some set of operations to complete, and often this means requests on the memory bus. This interferes with the bandwidth across the bus, which is one of the system's key performance bottlenecks. The more aggressive the processor is about concurrent and out-of-order operations, the more there is to lose by forcing it to synchronize.

The problem with the lack of atomicity goes far deeper than the trivial increment example just cited. Consider this simple piece of C code:

```
int done; // assume this is an initialized shared variable
if (done == 0)
{
    // do "stuff" here that you want to happen once
    done = 1;
}
```

If you have two threads and they try to execute this code at the same time, then both *may* perform the test and do the "stuff" before `done` is set to 1. You may think that by putting the `done=1` before doing the "stuff" you can fix the problem, but all you will do is make it happen less often. The instant that `done` is read for the comparison, another thread may come along and read the same value so that it can make the same test succeed. Subtle changes in the hardware or OS can dramatically impact how often this code fails in practice. One solution to this example is to use an atomic operation that your threading interface provides, but this doesn't get you very far, because it will provide only a limited set of atomic operations and you can't restrict your programming to just using those!

---

## Synchronization

---

The standard solution to this is to use synchronization primitives. Each threading interface provides some set of "synchronization primitives." These are usually fairly similar among interfaces, but the semantics can differ in subtle but important ways. Each primitive also brings with it different performance implications. Using some lightweight synchronization, even if blocking or waiting doesn't happen, might consume only a few tens or hundreds of cycles, whereas others might consume thousands (possibly many, many thousands). And this is the cost if there is no contention between threads!

Most of these primitives are in the form of a lock/unlock pair of operations. In some cases it is called `enter/leave` instead of `lock/unlock` because the pair is defining a "critical section" of code that you enter with the `lock` (or `enter`) operation and that you exit with the `unlock` (or `leave`) operation. Only a single thread can be inside such a critical section at a time. Other threads attempting to enter are forced to wait until the one in the critical section leaves it (i.e., they are "mutually excluded," which is shortened to "mutex"). This ensures that the code in the critical section is executed atomi-

cally. Unfortunately, this also forces this part of the program to be effectively single-threaded, and spending too much time in critical sections reduces your performance to that of a single hardware thread (or less, since executing synchronization primitives has a cost).

```
int done; // assume this is an initialized shared variable
enter_critical_section();
if (done == 0)
{
    // do stuff here that you want to happen once
    done = 1;
}
leave_critical_section();
```

From this you might think that synchronization is only needed when a value is going to be modified. Unfortunately this isn't so.

```
int *p; // assume this is an initialized shared variable
if (p != NULL)
{
    if (*p == 0)
    {
        // do something wonderful
    }
}
```

Here we are testing `p` to be sure that it is valid before using it. Unfortunately, some other code might come along and modify it (to `NULL`, for example) before it is dereferenced, causing unexpected behavior, an outright crash, or an exception.

One idea to attempt a fix might be this:

```
int *p; // assume this is an initialized shared variable
int *mycopy = p;
if (mycopy != NULL)
{
    if (*mycopy == 0)
    {
        // do something wonderful
    }
}
```

However, this can be a disaster as well. The memory referenced by `p` is part of the shared state, so simply making a copy of the pointer doesn't solve the problem. For example, another thread could deallocate the memory referenced by `p` or otherwise repurpose it. In a garbage-collected language the object will not have been deallocated, but if `p` now references a different object than `mycopy` you may be relying on (or changing) stale data. In other situations this might be a valid and efficient strategy.

There are likely to be multiple places in the code that modify a particular piece of shared state, so we need to be able to lock them all to make them mutually exclusive. To allow this, synchronization primitives are almost always objects—mutex objects, critical section objects, etc. The lock/unlock operations become methods, and all the normal issues of creation and lifetime management come into play. A synchronization object is a piece of shared state that is used to arbitrate between threads.

```
mutex m; // assume this is initialized shared state
int *p; // assume this is an initialized shared variable
```

```

m.lock();
if (p != NULL)
{
    if (*p == 0)
    {
        // do something wonderful
    }
}
m.unlock();

```

One obvious problem with this is that it becomes easy to forget the unlock operation or to retain the lock for long periods of time. This is particularly an issue if the lock and unlock operations are enacted indirectly through a higher-level piece of code via an interface, or if expensive function calls to other subsystems are made while the lock is held.

To mitigate these problems some threading interfaces provide lexically scoped synchronization. For example, in C#:

```

object m;
int *p; // assume this is an initialized shared variable
lock (m)
{
    if (p != NULL)
    {
        if (*p == 0)
        {
            // do something wonderful
        }
    }
}

```

Synchronization primitives are shared resources, and in C++ it is appropriate to apply the resource acquisition through a construction paradigm, as in, for example:

```

class MutexLock
{
public:
    MutexLock (Mutex m) : mMutex(m) { mMutex.Lock(); }
    ~MutexLock () { mMutex.Unlock(); }
private:
    Mutex mMutex;
};

```

As a project grows in size and more code needs to operate concurrently, a program will come to have multiple synchronization objects. A new problem arises in code with multiple synchronization primitives: deadlock. Imagine two threads (#1 and #2), which are using two mutexes (A and B). Now consider what happens in this scenario: Thread #1 acquires mutex A, thread #2 acquires mutex B, thread #1 attempts to acquire mutex B (and blocks until #2 releases it), and finally thread #2 attempts to acquire mutex A (and blocks until #1 releases it). These two threads are now stopped, each waiting for the other to release its resource, which of course they cannot do, because they are stopped.

Deadlock is relatively simple to be aware of and to avoid in a simple piece of software. It becomes considerably more complex to avoid in larger pieces of software. One solution is to use a single mutex instead of two different ones. Some early threaded operating systems adopted this approach to pro-

tect their internal resources. The problem with this approach was, as previously described, that the program can rapidly degenerate to being effectively single-threaded. The more hardware threads you have, the more of a loss this is.

An alternative to sharing mutexes widely is to avoid shared state and to use no other systems from within critical sections. In large projects this can become difficult or impossible. Powerful and important design patterns such as delegates and iterators can lead to unexpected situations where deadlock is possible.

The term “thread safety” is often used to describe objects (or systems) that are designed to be used from multiple threads simultaneously. It is often not clear what is meant by an object being thread-safe. One approach is to simply make each of the methods in the object lock a mutex while it executes to ensure that the object’s state remains consistent during the call. That does not ensure that the object’s state remains consistent between calls. For example, in C++ with STL:

```
vector<int> a;           // assume this is shared
int len = a.size();     // assume this is a synchronized op
int last_value = a[len-1]; // ... and so is this one
```

This code can fail even though all the operations on `a` are individually thread-safe. If another thread removes an element from `a` after the count is retrieved, then this thread will index past the end. If another thread adds an element to `a`, then it won’t be the last value that is retrieved.

A naive solution to this problem is to provide a lock/unlock operation as part of the object’s interface. The user of the object holds the lock while working with it. Unfortunately, working on objects in isolation is rare—most interesting code uses multiple objects to accomplish something interesting. If more than one of those objects uses this approach, you may now find yourself in a potential deadlock situation. One example of this arises when iterating across containers. What happens if another thread changes the container while it is being iterated? The .NET framework’s containers, for example, throw an exception if this happens. The code doing the iteration must ensure that this cannot happen. The obvious solution is to lock a mutex for the duration of the iteration, but heavily used or large containers or slow per-element operations can make this prohibitively expensive. Furthermore, if the operation involves calls to other code (such as methods on the objects from the container), then deadlock can become an issue. A common example is moving objects from one container to another where both containers are using this locking strategy.

---

## Amdahl’s Law

---

It is useful to understand Amdahl’s Law as it applies to concurrency. It actually applies to the principle of optimization in general, but here I’ll just point out its implication with respect to synchronization. The time to execute a program is equal to the sum of the time required by the sequential and parallel parts of the program:  $T = S + P$ . Optimizations to the sequential part can reduce  $S$ , and optimizations to the parallel part can reduce  $P$ . The obvious optimization to the parallel part is to increase the number of processors. Twice as many processors might reduce  $P$  by 2; an infinite number might reduce it to virtually nil. If the split between  $S$  and  $P$  cannot be changed, however, then the maximum speedup possible by parallel optimizations is  $1/S$  (i.e.,  $P$  has gone to zero). If the sequential part of the program is 50% of the execution time, adding an infinite number of processors will only double

its performance! It should therefore be clear that minimizing the amount of time spent in sequential parts of the program (i.e., critical sections or holding mutex locks) is essential to performance scaling.

One important point about the preceding paragraph is whether the split between S and P can be changed. Changing this split is a powerful approach, akin to improving your algorithm instead of your implementation. It can be accomplished in three basic ways. The obvious, although difficult, one is to replace some of your sequential work with parallel equivalents. The second is simply to *not* do some of the sequential work, which is not always an option but sometimes worth considering. And the third is to do more parallel work. The parallel work is what will scale with increasingly concurrent hardware, so do more of it.

---

## Conclusions

---

The point of this discussion of threaded programming is not to enumerate all the potential pitfalls of threaded programming. Nor do I claim that there aren't solutions to each of these individual problems, even if they require careful design and implementation and rigorous testing. What I am trying to convey is that there is a minefield of nasty, subtle, intractable problems and that the programming model embodied by the common programming languages does nothing to help the programmer deal with it. These languages were conceived for programming computers that no longer exist.

So what, with respect to concurrency, might a language and compiler take care of that could make life easier for the programmer? Here's a sampling (none of which I'm going to explain), just to give you an idea: structure and field alignment, accounting for cache line size and associativity, accounting for cache read/write policies, using cache prefetch instructions, accounting for automatic hardware prefetching, dealing with speculative hardware execution (particularly stores), leveraging DMA hardware, using a context switch versus spin-lock synchronization, dealing with variables co-inhabiting cache lines, loading variables into different register sets based on capabilities versus cost of inter-set moves versus usage, dealing with different register sizes, handling capability and precision differences among data types, organizing vectors of structures based on algorithmic usage and memory system in-flight transaction capabilities, vector access patterns based on algorithms and hardware load/store/permute capabilities, compare and branch versus bitwise math, branching versus selection, dealing with register load/store alignment restrictions, choosing SoA versus AoS in-memory and in-register organizations, hoisting computations from conditionals to fill pipeline stalls, software pipelining of loops, organizing and fusing loops based on register set size, selecting thread affinities, latching shared values for atomicity, object synchronization, trade-off between instruction and thread-level parallelism, and leveraging hardware messaging and synchronization capabilities. Any one of these things can result in doubling the performance of your algorithm and, although they may not combine linearly, you typically have to get them all right to get close to maximum performance. And many of them will break your code if you get them wrong. Some of them will simply make your code nonportable and fragile.

There are alternative categories of languages, and some of these offer programming models with strong advantages in concurrent programming. Functional languages (e.g., Haskell, Lisp, APL, ML, Erlang) offer some powerful advantages, but most bring with them various disadvantages and none is mainstream. Array programming languages (e.g., APL, MATLAB) are powerfully expressive of data parallel concepts but have limitations and haven't

reached their full potential in terms of optimization—and they typically eschew the object-oriented paradigms that are now deeply (and rightfully) entrenched in modern software development. Less traditional programming models and languages exist that embody powerful concurrent concepts such as message-passing objects networks (e.g., Microsoft’s Robotics Studio), but they are far from standard, are not portable, and typically cannot be integrated into existing environments. Highly specialized languages and tools such as StreamIt, RapidMind, and cT also suffer from the same issues. A few very successful specialized languages exist, particularly in the domain of graphics: HLSL, GLSL, and Cg have given graphics software and hardware developers tremendous power and flexibility. They have integrated well into existing software systems, and being part of the C language family makes them readily accessible to the C/C++/Java/C# communities. The wide adoption of these shading languages gives some indication of and hope about what is possible. All of these alternatives point in directions in which we can take our programming languages and models in the future.

---

#### SUGGESTED READING

---

Video of Herb Sutter’s talk “Machine Architecture: Things Your Programming Language Never Told You” at NWCPP on September 19, 2007: <http://video.google.com/videoplay?docid=-4714369049736584770>.

Slides for Sutter’s talk: <http://www.nwcpp.org/Meetings/2007/09.html>.

Michael D. McCool, “Achieving High Performance by Targeting Multiple Hardware Mechanisms for Parallelism,” *login*, April 2008.

Transactional memory: <http://research.microsoft.com/~simonpj/papers/stm/index.htm>.

Language taxonomy: <http://channel9.msdn.com/Showpost.aspx?postid=326762>.

Nested data parallelism: <http://research.microsoft.com/~simonpj/papers/ndp/NdpSlides.pdf>.

Erlang: [http://www.sics.se/~joe/talks/ll2\\_2002.pdf](http://www.sics.se/~joe/talks/ll2_2002.pdf).

P. Grogono and B. Shearing, “A Modular Programming Language Based on Message Passing”: <http://users.encs.concordia.ca/~grogono/Erasmus/E01.pdf>.



DAN TSAFRIR, DILMA DA SILVA, AND  
DAVID WAGNER

## the murky issue of changing process identity: revising “setuid demystified”



Dan Tsafirir is a postdoctoral researcher at the IBM T.J. Watson Research Center in New York. He is a member of the advanced operating systems group and is interested in various aspects of operating systems.

*dants@us.ibm.com*



Dilma da Silva is a researcher at the IBM T.J. Watson Research Center in New York. She manages the Advanced Operating Systems group. Prior to joining IBM, she was an Assistant Professor at University of Sao Paulo, Brazil. Her research in operating systems addresses the need for scalable and customizable system software.

*dilmasilva@us.ibm.com*



David Wagner is a professor in the computer science division at the University of California at Berkeley. He studies computer security, cryptography, and electronic voting.

*daw@cs.berkeley.edu*

**DROPPING UNNEEDED PROCESS PRIVILEGES** promotes security but is notoriously error-prone because of confusing `set*id` system calls with unclear semantics and subtle portability issues. To make things worse, existing recipes to accomplish the task are lacking, related manuals can be misleading, and the associated kernel subsystem might contain bugs. We therefore proclaim the system as untrustworthy when it comes to the subject matter, and we suggest a defensive, easy-to-use solution that addresses all concerns.

Whenever you run a program, it assumes your identity and you lend it all your power: Whatever you're allowed to do, it too is allowed. This includes deleting your files, killing your other programs, changing your password, and retrieving your mail, for example. Occasionally, you need to write programs that enhance the power of others. Consider, for example, a Mahjongg game that maintains a high-score file. Of course, making the file writable by all is not a very good idea if you want to ensure that no one cheats, so Mahjongg must somehow convey to players the ability to update the file in a controlled manner. In UNIX systems this is done as follows: When a game ends, if the score is high enough, Mahjongg temporarily assumes the identity of the file's owner, makes the appropriate modifications, and switches back to the identity of the original player.

Many standard utilities work this way, including `passwd` and `chsh` (which update `/etc/passwd`), `xterm` (which updates `utmp` usage information), `su` (which changes user), `sudo` (which acts as root), and `X` (which accesses interactive devices). The common feature of these tools is that they know their real identity is of a nonprivileged user, but they have the ability to assume a privileged identity when required. (Note that “privileged” doesn't necessarily mean root; it merely means some other identity that has the power to do what the real user can't.) Such executables are collectively referred as “setuid programs,” because (1) they must be explicitly associated with a “setuid bit” (through the `chmod` command) and (2) they pull off the identity juggling trick through the use of `set*id` system calls (`setuid(2)`, `setreuid(2)`, and all their friends).

There's another, often overlooked, type of program that can do identity juggling but does *not* have an associated setuid bit. These start off as root pro-

cesses and use `setuid` system calls to change their identity to that of an ordinary nonprivileged user. Examples include the login program, the cron daemon (which runs user tasks at a specified time), daemons providing service to remote users by assuming their identity (`sshd`, `telnetd`, `nfs`, etc.), and various mail server components.

Both types of programs share a similar philosophy: To reduce the chances of their extra powers being abused, they attempt to obey the principle of least privilege, which states that “every program and every user of the system should operate using the least set of privileges necessary to complete the job” [16]. For `setuid` programs this translates to:

1. minimizing the number and duration of the time periods at which the program temporarily assumes the privileged identity, to reduce the negative effect that programming mistakes might have (e.g., mistakenly removing a file as root can have far greater negative implications than doing it when the nonprivileged identity is in effect), and
2. permanently giving up the ability to assume the privileged identity as soon as it’s no longer needed, so that if an attacker gains control (e.g., through a buffer overflow vulnerability), the attacker can’t exploit those privileges.

The principle of least privilege is a simple and sensible rule. But when it comes to identity-changing programs (in the immortal words of *The Essex* [7] or anybody who ever tried to lose weight [14]) it’s easier said than done. Here are a few quotes that may explain why it’s at least as hard as doing a diet: Chen et al. said that “for historical reasons, the `uid`-setting system calls are poorly designed, insufficiently documented, and widely misunderstood” and that the associated manuals “are often incomplete or even wrong” [2]. Dean and Hu observed that “the `setuid` family of system calls is its own rat’s nest; on different UNIX and UNIX-like systems, system calls of the same name and arguments can have different semantics, including the possibility of silent failures” [3]. Terek and Dik concluded that “many years after the inception of `setuid` programs, how to write them is still not well understood by the majority of people who write them” [17]. All these deficiencies have made the `setuid` mechanism the source of many security vulnerabilities.

It has been more than 30 years since Dennis Ritchie introduced the `setuid` mechanism [15] and more than 20 years since people started publishing papers about how to correctly write `setuid` programs [1]. The fact that this article has something new to say serves as an unfortunate testament that the topic is not yet resolved. Our goal in this paper is to provide the equivalent of a magical diet pill that effortlessly makes you slim (or at least lays the foundation for this magic). Specifically, we design and implement an intuitive change-identity algorithm that abstracts away the many pitfalls, confusing details, operating-system-specific behavior, and portability issues. We build on and extend the algorithm proposed by Chen et al. [2], which neglected to factor in the role that supplementary groups play in forming an identity. Our code is publicly available [18]. It was extensively tested on Linux 2.6.22, FreeBSD 7.0-STABLE, OpenSolaris, and AIX 5.3. We warn that, given the history of subtle pitfalls in the `setuid` syscalls, it may be prudent for developers to avoid relying upon our algorithm until it has been subject to careful review by others.

---

## User Identity vs. Process Identity

---

Before attempting to securely switch identities, we need to define what the term “identity” means. In this context, we found it productive to make a dis-

inction between two types of identities: that of a user and that of a process. The user's credentials include the user ID (uid), the user's primary group (gid), and an additional array of supplementary groups (sups). Collectively, they determine which system resources the user can access. In particular, a zero uid is associated with the superuser (root) who can access all resources. We define the `ucred_t` type to represent a user by aggregating these three fields, as follows:

```
typedef struct supplementary_groups {
    gid_t *list; // sorted ascending, no duplicates
    int    size; // number of entries in 'list'
} sups_t;

typedef struct user_credentials {
    uid_t uid;
    gid_t gid;
    sups_t sups;
} ucred_t;
```

Things are a bit more complicated when it comes to the corresponding process credentials. Each process has three user IDs: real (ruid), effective (euid), and saved (suid). The real uid identifies the “owner” of the process, which is typically the executable's invoker. The effective uid represents the identity in effect, namely, the one used by the OS (operating system) for most access decisions. The saved uid stores some previous user ID, so that it can be restored (copied to the euid) at some later time with the help of `set*uid` system calls. Similarly, a process has three group IDs: rgid, egid, and sgid. We define the `pcred_t` type to encapsulate the credentials of a process:

```
typedef struct user_ids { uid_t r, e, s; } uids_t;
typedef struct group_ids { gid_t r, e, s; } gids_t;

typedef struct process_credentials {
    uids_t uids; // uids.r = ruid, uids.e = euid, uids.s = suid
    gids_t gids; // gids.r = rgid, gids.e = egid, gids.s = sgid
    sups_t sups;
} pcred_t;
```

Supplementary groups can be queried with the help of the `getgroups` system call. The ruid, euid, rgid, and egid of a process can be retrieved with `getuid`, `geteuid`, `getgid`, and `getegid`, respectively. The ways to find out the values of suid and sgid are OS-specific.

In Linux, each process also has an fsuid and an fsgid, which are used for access control to the file system. Normally, these are equal to the euid and egid, respectively, unless they are explicitly changed [11]. As this rarely used feature is Linux-specific, it is not included in the aforementioned data structures. To ensure correctness, our algorithm never manipulates the fsuid or fsgid, ensuring that (if programs rely only upon our interface for manipulating privileges) the fsuid and fsgid will always match the euid and egid.

The benefit of differentiating between user and process identities is that the former is more convenient to work with, easier to understand, better captures the perception of programmers regarding identity, and typically is all that is needed for programmers to specify what kind of an identity they require. In other words, the notions of real, effective, and saved IDs are not important in their own right; rather, they are simply the technical means by which identity change is made possible. Note, however, that “user” isn't an abstraction that is represented by any kernel primitive: The kernel doesn't deal with users; it deals with processes. It is therefore the job of our algorithm to internally use `pcred_t` and provide the appropriate mappings.

---

## Rules of Identity Juggling

---

### IDENTITY PROPAGATION AND SPLIT PERSONALITIES

---

The second thing one has to consider when attempting to correctly switch identities is the manner by which processes initially get their identity. When a user *rik* logs in, the login program forks a process *P* and sets things up such that (1) *P*'s three uids hold *rik*'s uid, (2) *P*'s three gids hold *rik*'s primary group, and (3) *P*'s supplementary array is populated with the gids of the groups to which *rik* belongs. The process credentials are then inherited across fork. They are also inherited across `exec`, unless the corresponding executable *E* has its `setuid` bit set, in which case the effective and saved uids are set to be that of *E*'s owner (but the real uid remains unchanged). Likewise, if *E* is `setgid`, then the saved and effective groups of the new process are assigned with *E*'s group.

Conversely, the supplementary array is *always* inherited as is, even if *E*'s `setuid`/`setgid` bits are set. Notice that this can lead to a bizarre situation where *E* is running with a split personality: The effective user and group are of *E*'s owner, whereas the supplementary groups are of *E*'s invoker. This isn't necessarily bad (and in fact constitutes the typical case), but it's important to understand that this is what goes on.

---

### USER ID JUGGLING

---

Since access control is based on the effective user ID, a process gains privilege by assigning a privileged user ID to its `eu`id, and drops privilege by removing it. To drop privilege temporarily, a process removes the privileged user ID from its `eu`id but stores it in its saved ID; later, the process may restore privilege by copying this value back to the `eu`id. To drop privilege permanently, a process removes the privileged user ID from all three uids. Thereafter, the process can never restore privilege.

Roughly speaking, there typically exists some technical way for a process to copy the value from one of its three uids to another, and thus perform the uid juggling as was just described. If the process is nonroot (`uid`  $\neq$  0), then that's all it can do (juggle back and forth between the real and saved uids). Root, however, can assume any identity.

---

### PRIMARY GROUP JUGGLING

---

The rules of changing gids are identical, with the exception that `egid=0` doesn't convey any special privileges: Only if `eu`id=0 can the process set arbitrary gids.

---

### SUPPLEMENTARY GROUPS JUGGLING

---

The rules for changing supplementary groups are much simpler: If a process has `eu`id=0, it can change them however it likes through the `setgroups` system call. Otherwise, the process is forbidden from using `setgroups` and is stuck with the current setting. The implications for `setuid` programs are interesting. If the `setuid` program drops privileges (assuming the identity of its invoker), then the supplementary groups will already be set appropriately. However, until that happens, the program will have a split personality. A `setuid`-root program can set the supplementary groups to match its privileged identity, if it chooses. However, nonroot `setuid` programs cannot: They will suffer from a split personality for as long as they maintain their privileged

identity, and there's simply no way around it. As a result, nonroot `setuid` programs might run with extra privileges that their creators did not anticipate.

---

### MESSINESS OF SETUID SYSTEM CALLS

---

Several standard `set*id` system calls allow programmers to manipulate the real, effective, and saved IDs, in various ways. To demonstrate their problematic semantics, we focus on only `setuid(2)` through an example of a vulnerability found in a mainstream program. Googling the word “`setuid`” with “vulnerability” or “bug” immediately brings up many examples that are suitable for this purpose. But to also demonstrate the prevalence of the problem, we attempted to find a new vulnerability. Indeed, the first program we examined contained one.

Exim is a popular mail server that is used by default in many systems [5]. Figure 1 shows the function `exim` uses to drop privileges permanently, taken from the latest version available at the time of this writing [6]. It implicitly assumes that calling `setuid` will update all three uids, so that all privileges are permanently relinquished. This assumption indeed holds for some OSes (e.g., FreeBSD). But if the effective ID is nonzero (which may be the case according to the associated documentation) then the assumption doesn't hold for Linux, Solaris, and AIX, as the semantics of `setuid` under these circumstances dictate that only the `eid` will be updated, leaving the `ruid` and `suid` unchanged. Consequently, if `exim` is compromised, the attacker can restore `exim`'s special privileges and, for example, obtain uncontrolled access to all mail in the system.

Although this particular vulnerability isn't nearly as dangerous as some previously discovered `setuid` bugs, it does successfully highlight the problematic system call behavior, which differs not only between OSes but also according to the current identity.

```
/*
 * This function sets a new uid and gid permanently, optionally calling
 * initgroups() to set auxiliary groups. There are some special cases when
 * running Exim in unprivileged modes. In these situations the effective
 * uid will not be root; [...]
 */
void exim_setuid(uid_t uid, gid_t gid, BOOL igflag, uschar *msg)
{
    uid_t eid = geteuid();
    gid_t egid = getegid();

    if (eid == root_uid || eid != uid || egid != gid || igflag) {
        if (igflag) {
            /* do some supplementary groups handling here */ ...
        }

        if (setgid(gid) < 0 || setuid(uid) < 0) {
            /* PANIC! */ ...
        }
    }
}
```

**FIGURE 1: EXIM'S CODE TO PERMANENTLY CHANGE IDENTITY CONTAINS A VULNERABILITY.**

---

## Safely Dropping Privileges

---

Equipped with a good understanding of the subject, we go on to develop an algorithm to safely drop privileges permanently. We do so in a top-down manner, making use of the `ucrd_t` and `pcrd_t` types previously defined. Figure 2 (facing page) shows the algorithm. Its input parameter specifies the target identity; the algorithm guarantees to permanently switch to the target identity or clearly indicate failure. The algorithm works by first changing the supplementary groups, then changing the `gids` and changing the `uids` (in that order), and, finally, checking that the current identity matches the target identity.

---

### ERROR HANDLING

---

There are two ways to indicate failure, depending on how the macros `DO_CHK` and `DO_SYS` are defined:

```
#ifndef LIVING_ON_THE_EDGE
# define DO_SYS(call)  if( (call) == -1 ) return -1  /* do system call  */
# define DO_CHK(expr) if( ! (expr) ) return -1  /* do boolean check */
#else
# define DO_SYS(call)  if( (call) == -1 ) abort()    /* do system call  */
# define DO_CHK(expr) if( ! (expr) ) abort()    /* do boolean check */
#endif
```

But although reporting failure through return values is possible, we advise against it, as it might leave the identity in an inconsistent state. Thus, when an identity change fails in the middle, programmers should either abort or really know what they're doing.

---

### INPUT CHECK

---

The `ucrd_is_sane` function checks the validity of the input parameter. It is implemented as follows:

```
long nm = sysconf(_SC_NGROUPS_MAX);
return (nm >= 0) && (nm >= uc->supsize) && (uc->supsize >= 0) &&
    uc->uid != (uid_t)-1 &&
    uc->gid != (gid_t)-1;
```

The maximal size of the supplementary groups may differ between systems, but it can be queried in a standard way. We also check that the user and group IDs aren't `-1`, because this has special meaning for several `set*id` system calls ("ignore").

---

### VERIFICATION

---

The first chunk of code in Figure 2 is responsible for setting the supplementary groups to `uc->supsize`, the three `gids` to `g`, and the three `uids` to `u`. Setting the `uids` last is important, because afterward the process might lose its privilege to change its groups. Setting supplementary groups before primary groups is also important, for reasons to become clear later on. The remainder of the function verifies that all of these operations successfully changed our credentials to the desired identity. This policy is required in order to prevent mistakes in the face of the poorly designed `set*id` interface (e.g., this policy would have prevented the `exim` vulnerability), to protect against possible

```

int drop_privileges_permanently(const ucred_t *uc /*target identity*/)
{
    uid_t u = uc->uid;
    gid_t g = uc->gid;
    pcred_t pc;

    DO_CHK( ucred_is_sane(uc) );
    DO_SYS( set_sups( &uc->sups ) );
    DO_SYS( set_gids( g/*real*/, g/*effective*/, g/*saved*/) );
    DO_SYS( set_uids( u/*real*/, u/*effective*/, u/*saved*/) );

    DO_SYS( get_pcred( &pc ) );
    DO_CHK( eql_sups ( &pc.sups , &uc->sups ) );
    DO_CHK( g == pc.gids.r && g == pc.gids.e && g == pc.gids.s );
    DO_CHK( u == pc.uids.r && u == pc.uids.e && u == pc.uids.s );
    free( pc.sups.list );

#ifdef __linux__
    DO_SYS( get_fs_ids( &u, &g ) );
    DO_CHK( u == uc->uid && g == uc->gid );
#endif

    return 0; /* success */
}

```

**FIGURE 2: PERMANENTLY SWITCHING IDENTITY AND VERIFYING THE CORRECTNESS OF THE SWITCH.**

related kernel bugs [2] or noncompliant behavior (see below) and to defend against possible future kernel changes. These reasons, combined with the fact that having the correct identity is crucial in terms of security, provide good motivation for our untrusting approach.

### QUERYING PROCESS IDENTITY

The `get_pcred` function we implement fills the memory pointed to by the `pcred_t` pointer it gets. We get the `ruid`, `rgid`, `euid`, and `egid` with the help of the standard system calls `getuid`, `getgid`, `geteuid`, and `getegid`, respectively. Unfortunately, there's no standard way to retrieve saved IDs, so we use whatever facility the OS makes available, as shown in Figure 3 on the next page. The `getresuid` and `getresgid` nonstandard system calls are the easiest to use and the most popular among OSes. AIX's `getuidx` and `getgidx` also have easy semantics, whereas with Solaris the programmer must resort to using Solaris's `/proc` interface [10].

The supplementary groups are retrieved with the help of the standard `getgroups` system call. To allow for easy comparison of supplementary arrays, we normalize the array by sorting it and by removing duplicate entries, if any exist. The array is `malloced`, and it should therefore be freed later on.

### LINUX FILESYSTEM IDS

In Linux, the `fsuid` is supposed to mirror the `euid`, as long as `setfsuid` isn't explicitly used [11], and the same goes for `fsgid` and `egid`. However, there has been at least one kernel bug that violated this invariant [2]. Therefore, in accordance with our defensive approach, the algorithm in Figure 2 explicitly

```

int get_saved_ids(uid_t *suid, gid_t *sgid)
{
    #if defined(__linux__)    || defined(__HPUX__)    || \
        defined(__FreeBSD__) || defined(__OpenBSD__) || defined(__DragonFly__)
        uid_t ruid, euid;
        gid_t rgid, egid;
        DO_SYS( getresuid(&ruid, &euid, suid) );
        DO_SYS( getresgid(&rgid, &egid, sgid) );

    #elif defined(_AIX)
        DO_SYS( *suid = getuidx(ID_SAVED) );
        DO_SYS( *sgid = getgidx(ID_SAVED) );

    #elif defined(__sun__) || defined(__sun)
        prcred_t p; /* prcred_t is defined by Solaris */
        int fd;
        DO_SYS( fd = open( "/proc/self/cred", O_RDONLY ) );
        DO_CHK( read(fd, &p, sizeof(p)) == sizeof(p) );
        DO_SYS( close(fd) );
        *suid = p.pr_suid;
        *sgid = p.pr_sgid;

    #else
    # error "need to implement, notably: __NetBSD__, __APPLE__, __CYGWIN__"
    #endif
    return 0;
}

```

**FIGURE 3: GETTING THE SAVED UID AND GID IS AN OS-DEPENDENT OPERATION.**

verifies that the fs-invariant indeed holds. As there is no `getfsuid` or `getfsgid`, our implementation of `get_fs_ids` is the C equivalent of

```

grep Uid /proc/self/status | awk '{print $5}' # prints fsuid
grep Gid /proc/self/status | awk '{print $5}' # prints fsgid

```

## SETTING UIDS AND GIDS

The POSIX-standard interfaces for setting IDs are tricky, OS-dependent, and offer no way to directly set the saved IDs. Consequently, *nonstandard* interfaces are preferable, if they offer superior semantics. This is the design principle underlying our implementation of `set_uids` and `set_gids`. The implementation is similar in spirit to the code in Figure 3, but it is complicated by the fact that nonprivileged processes are sometimes not allowed to use the preferable interface, in which case we fall back on whatever is available.

Specifically, all OSes that support `getresuid` (see Figure 3) also support `setresuid` and `setresgid`. These offer the clearest and most consistent semantics and can be used by privileged and nonprivileged processes alike. (Of course the usual restrictions for nonprivileged processes still apply, namely, each of the three parameters must be equal to one of the three IDs of the process.) In Solaris, only root can use the `/proc` interface for setting IDs [10], so with nonroot processes we naively use `seteuid` and `setreuid` (and their gid counterparts) and hope for the best: The verification part in Figure 2 will catch any discrepancies. In AIX, `setuidx` and `setgidx` are the clearest and most expressive, and they can be used by both root and nonroot processes [13]. However, AIX is very restrictive: a nonroot process can



only change its effective IDs, so dropping privileges permanently is impossible for nonroot processes; also, root processes are allowed to set `euid`, `euid/ruid`, or `euid/ruid/suid`, but only to the same value.

#### SUPPLEMENTARY GROUPS CAVEATS

Recall that nonroot processes are not allowed to call `setgroups`. Therefore, to avoid unnecessary failure, `setgroups` is only invoked if the current and target supplementary sets are unequal, as shown in Figure 4. (Disregard the FreeBSD chunk of code for the moment.) Additionally, recall that after setting the supplementary groups in Figure 2, we verify that this succeeded by querying the current set of supplementary groups and checking that it matches the desired value. In both cases the current and target supplementary sets must be compared. But, unfortunately, this isn't as easy as one would expect.

```
int set_sups(const sups_t *target_sups)
{
    sups_t targetsups = *target_sups;

#ifdef __FreeBSD__
    gid_t arr[ targetsups.size + 1 ];
    memcpy(arr+1, targetsups.list, targetsups.size * sizeof(gid_t) );
    targetsups.size = targetsups.size + 1;
    targetsups.list = arr;
    targetsups.list[0] = getegid();
#endif

    if( geteuid() == 0 ) { // allowed to setgroups, let's not take any chances
        DO_SYS( setgroups(targetsups.size, targetsups.list) );
    }
    else {
        sups_t cursups;
        DO_SYS( get_sups( &cursups) );
        if( ! eql_sups( &cursups, &targetsups ) // this will probably fail... :(
            DO_SYS( setgroups(targetsups.size, targetsups.list) );
        free( cursups.list );
    }

    return 0;
}
```

**FIGURE 4: SETTING SUPPLEMENTARY GROUPS, WHILE TRYING TO AVOID FAILURE OF NONROOT PROCESSES, AND ACCOMMODATING NONCOMPLIANT BEHAVIOR OF FREEBSD.**

The POSIX standard specifies that “it is implementation-defined whether `getgroups` also returns the effective group ID in the grouplist array” [9]. This seemingly harmless statement means that if the `egid` is in fact found in the list returned by `getgroups`, there's no way to tell whether this group is actually a member of the supplementary group list. In particular, there is no reliable, portable way to get the current list of supplementary groups. As a result, our code for comparing the current and target supplementary sets (see `eql_sups` in Figure 5, which is used in Figure 2 and Figure 4) assumes that they match even if the current supplementary set contains the `egid` and the target supplementary set doesn't. This isn't completely safe, but it's the best we can do, and it's certainly better than not comparing at all.

```

bool eql_sups(const sups_t *cursups, const sups_t *targetsups)
{
    int    i, j, n = targetsups->size;
    int    diff = cursups->size - targetsups->size;
    gid_t  egid = getegid();

    if( diff > 1 || diff < 0 ) return false;

    for(i=0, j=0; i < n; i++, j++)
        if( cursups->list[j] != targetsups->list[i] ) {
            if( cursups->list[j] == egid ) i--; // skipping j
            else return false;
        }

    // If reached here, we're sure i==targetsups->size. Now, either
    // j==cursups->size (skipped the egid or it wasn't there), or we didn't
    // get to the egid yet because it's the last entry in cursups
    return j == cursups->size ||
        (j+1 == cursups->size && cursups->list[j] == egid);
}

```

**FIGURE 5: WHEN COMPARING THE CURRENT SUPPLEMENTARY ARRAY TO THE TARGET ARRAY, WE IGNORE THE EGID IF IT'S INCLUDED IN THE FORMER.**

---

#### NONCOMPLIANT FREEBSD BEHAVIOR

---

Kernel designers might be tempted to internally represent the egid as just another entry in the supplementary array, as this can somewhat simplify the checking of file permissions. Indeed, instead of separately comparing the file's group against (1) the egid of the process and (2) its supplementary array, only the latter check is required. The aforementioned POSIX rule that allows `getgroups` to also return the egid reflects this fact. But POSIX also explicitly states that “set[\*]gid function[s] shall not affect the supplementary group list in any way” [12]. And, likewise, `setgroups` shouldn't affect the egid. So such a design decision, if made, must be implemented with care.

The FreeBSD kernel has taken this decision and designated the first entry of the supplementary array to the egid of the process. But the implementers weren't careful enough, or didn't care about POSIX semantics [4]. When trying to understand why the verification code in Figure 2 sometimes fails in FreeBSD, we realized that the kernel ignores the aforementioned POSIX rules and makes no attempt to mask the internal connection between egid and the supplementary array. Thus, when changing the array through `setgroups`, the egid becomes whatever happens to be the first entry of the array. Likewise, when setting the egid (e.g., through `setegid`), the first entry of the array changes accordingly, in clear violation of POSIX. The code in the beginning of Figure 4 accommodates this noncompliant behavior. Additionally, whenever we need to set the egid, we always make sure to do it after setting the supplementary groups, not before (see Figure 2).

---

#### TEMPORARILY DROPPING AND RESTORING PRIVILEGES

---

Our implementation also includes functions to temporarily drop privileges and to restore them. They are similar to Figure 2 in that they accept a “target identity” `ucred_t` argument, they treat supplementary groups identically, and they verify that the required change has indeed occurred. When dropping privileges temporarily, we change only the `euid/egid` if we can help it (namely, if the values before the change are present in the real or saved IDs,

which means restoration of privileges will be possible). Otherwise we attempt to copy the current values to the saved IDs before making the change. (Unfortunately, this will fail on AIX for nonroot processes.) The algorithm that restores privileges performs operations in the reverse order: first restoring uids, and only then restoring groups; saved and real IDs are unaffected.

---

### CAUTION!

Identity is typically shared among threads of the same application. Consequently, our code is not safe in the presence of any kind of multithreading: Concurrent threads should be suspended, or else they run the risk of executing with an inconsistent identity. Likewise, signals should be blocked or else the corresponding handlers might suffer from the same deficiency.

The algorithms described in this article do not take into account any capabilities system the OS might have (e.g., “POSIX capabilities” in Linux [8]). Capabilities systems, if used, should be handled separately.

---

### Conclusion

Correctly changing identity is an elusive, OS-dependent, error-prone, and laborious task. We therefore feel that it is unreasonable and counterproductive to require every programmer to invent his or her own algorithm to do so, or to expect programmers to become experts on these pitfalls. We suggest that the interests of the community would be better served by a unified solution for managing process privileges, and we propose the approach outlined in this article as one possible basis for such a solution. Our code is publicly available [18]. We welcome suggestions, bug reports, and extensions.

---

### REFERENCES

- [1] M. Bishop, “How to Write a Setuid Program,” *login* 12(1) (Jan./Feb. 1987).
- [2] H. Chen, D. Wagner, and D. Dean, “Setuid Demystified,” in *11th USENIX Security Symp.*, pp. 171–190 (Aug. 2002).
- [3] D. Dean and A.J. Hu, “Fixing Races for Fun and Profit: How to Use Access(2),” in *13th USENIX Security Symp.*, pp. 195–206 (Aug. 2004).
- [4] R. Ermilov, R. Watson, and B. Evans, [CFR] `ucred.cr_gid`, thread from the FreeBSD-current mailing list: <http://www.mail-archive.com/freebsd-current@freebsd.org/msg28642.html> (June 2001) (accessed March 2008).
- [5] Exim Internet mailer: <http://www.exim.org/> (accessed March 2008).
- [6] Exim-4.69/src/exim.c, source code of exim 4.69: <ftp://ftp.exim.org/pub/exim/exim4/exim-4.69.tar.gz> (accessed March 2008).
- [7] W. Linton and L. Huff, “Easier Said Than Done,” performed by The Essex (July 1963): <http://www.youtube.com/watch?v=tgJ1ssTjtnA> (accessed March 2008).
- [8] Man capabilities(7)—Linux man page—overview of Linux capabilities: <http://linux.die.net/man/7/capabilities> (accessed Mar 2008).
- [9] Man getgroups(2)—the Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 edition: <http://www.opengroup.org/online-pubs/000095399/functions/getgroups.html> (accessed March 2008).

- [10] Man proc(4)—Solaris 10 reference manual collection: <http://docs.sun.com/app/docs/doc/816-5174/proc-4?l=en&a=view> (accessed March 2008).
- [11] Man setfsuid(2)—Linux man page: <http://linux.die.net/man/2/setfsuid> (accessed March 2008).
- [12] Man setgid(2)—the Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 edition: <http://www.opengroup.org/onlinepubs/000095399/functions/setgid.html> (accessed Jan. 2008).
- [13] Man setuidx—AIX Technical Reference: Base Operating System and Extensions, Volume 2: <http://publib.boulder.ibm.com/infocenter/systems/topic/com.ibm.aix.basetechref/doc/basetrf2/setuid.htm> (accessed March 2008).
- [14] Nerd Gurl, “Why Can’t I Ever Achieve My Goals?” Yahoo! Answers (Jan. 2008): <http://answers.yahoo.com/question/index?qid=20080101143342AAQ1jbO> (accessed March 2008).
- [15] D.M. Ritchie, Protection of Data File Contents, Patent No. 4135240 (July 1973): <http://www.google.com/patents?vid=USPAT4135240> (accessed March 2008).
- [16] J.H. Saltzer and M.D. Schroeder, “The Protection of Information in Computer Systems, *Proc. of the IEEE* 63(9), 1278–1308 (Sept. 1975).
- [17] C. Torek and C.H. Dik, Setuid mess (Sept. 1995): [http://yarchive.net/comp/setuid\\_mess.html](http://yarchive.net/comp/setuid_mess.html) (accessed March 2008).
- [18] D. Tsafir, D. Da Silva, and D. Wagner, “Change Process Identity”: <http://www.research.ibm.com/change-process-identity> or <http://code.google.com/p/change-process-identity>.

DAVID N. BLANK-EDELMAN

## practical Perl tools: a little place for your stuff



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 *Perl for System Administration*. He has spent the past 22+ 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.

[dnb@ccs.neu.edu](mailto:dnb@ccs.neu.edu)

Actually this is just a place for my stuff, ya know? That's all, a little place for my stuff. That's all I want, that's all you need in life, is a little place for your stuff, ya know? I can see it on your table, everybody's got a little place for their stuff. This is my stuff, that's your stuff, that'll be his stuff over there. That's all you need in life, a little place for your stuff.

—George Carlin

**GIVEN THAT THIS IS THE FILESYSTEM** and storage issue, let's take a look both at some of the most widely used ways of keeping your stuff using Perl and some of the less conventional methods you may have missed. We'll start with the most specific kind of stuff and move toward the most general.

### Storing Your Perl Data Structures

There are times when your program has worked hard to create such a really useful data structure in memory that you want it to persist beyond the life of that current program run. Maybe your program runs once a week and wants to add that week's data to the previous runs. Maybe you have a job that is meant to run a long time before producing an answer, and (as they would say on *The Sopranos* if it took place in a datacenter), "You wouldn't want anything to happen to that data, say, right in the middle of the run, now would you, palseie?" You owe it to yourself to periodically dump important data structures and other context to disk so it is possible to resume the computation should power, net, or the little hamster that runs around in the machine doing the actual work cut out.

The most conventional approach to this problem is to use the `Storable` module that ships with Perl. It's really easy to use. To write a data structure to a file you can use:

```
use Storable;
my %datastructure = ...
# some convoluted data structure
nstore (\%datastructure, 'persistfile')
    or die "Can't write to persistfile\n";
```

Then, in another program (or another routine of the current program), you can use:

```
$dsref = retrieve('persistfile');
```

and you'll get back a reference to the data structure you stored. If you wanted to work with just a hash like the original data structure we stored instead of

a reference to it (and you don't mind the hit to copy the data), you can dereference it as per usual like this:

```
%datastructure = %{$dsref};
```

One last note about this code. I tend to use `nstore()`, as shown here, versus the default `store()` method, because it keeps the data in “network order” format. `Storable` writes its data in a binary format. Using the network order storage function ensures that this data can be read on different machines with different byte orders. It is a little less efficient to store in this format, but retrieving is still as fast and you gain more portability for your data.

That last note offers a good segue to a less common approach. If you are concerned about keeping your data around in some binary format because you are a Long Now kind of person ([www.longnow.org](http://www.longnow.org)) and you don't trust you'll have anything with which to read it back when Perl is at version 8.5, then a text-based data serialization module may be more your cup of tea. You could use something like `Data::Dumper` (or, better yet, the more obscure `Data::Streamer`) to write text to a file, but I'd assert you will be better off finding the most standard standard you can find and writing that format. Two such similar standards that fit the bill nicely are `YAML` ([www.yaml.org](http://www.yaml.org)) and `JSON` ([www.json.org](http://www.json.org)). The former has deeper Perl roots; the latter is well known because of its ties to Javascript and the AJAX world. The `YAML` folks also note that `JSON` is essentially a subset of `YAML` and so any decent `YAML` parser worth its salt should also be able to parse `JSON`.

In previous columns I've demonstrated the use of `YAML`, so let's take a quick look at `JSON`.

Taking a data structure and turning it into `JSON` is easy:

```
use JSON;
my %datastructure = ... # some convoluted data structure

my $json = JSON->new();
my $encoded = $json->encode(\%datastructure);
# or, for prettier but less space-efficient results:
# my $encoded = $json->pretty->encode(\%datastructure);
```

If we had a data structure like this:

```
%datastructure = ( 'Fred' => 1, 'Barney' => 2, 'Wilma' => 3 )
```

then `$encoded` would contain:

```
{
  "Wilma" : 3,
  "Barney" : 2,
  "Fred" : 1
}
```

which isn't all that impressive until you start playing with more complex data structures. I recently had a case where I needed to parse the output of a Java program that spat out `JSON` and I was able to write code like this:

```
use JSON;

# get the results of the zmGetUserFolders command in JSON format
open my $ZMMBOX, zmGetUserFolders($from_user) . "|"
  or die "Can't run zmGetUserFolders(): $!";
my $json_data = join( " ", <$ZMMBOX> );
close $ZMMBOX;

# parse it into a Perl data structure
my $folders = from_json($json_data);
```

```
# extract the hrefs that contain the folders we care about
my @sharedfolders
  = grep { exists $_->{ownerId} and $_->{view} eq 'appointment' }
    @{$folders->{children}};
```

This code ran the command, parsed the JSON output, and then walked the list of folders returned, looking for those with the right owner and view type.

---

## Storing Key–Value Pairs

---

A slightly more general and hence more widely used scheme for data storage involves a simpler key–value model. For instance, Username is associated with “dnb,” “FirstName” with “David,” and “LastName” with “Blank-Edelman.” It isn’t a particularly sophisticated idea, but it is the thing that makes Perl’s hashes (and Snobol4’s tables, which led to associative arrays in awk to give the predecessor languages their proper due respect) so useful.

The easiest and most conventional way to store data like this is to use Perl’s `tie()` functionality to call an external database library such as `gdbm` or `BerkeleyDB` (my preference). Two past columns discussed `tie()` in all of its glory, so let’s make do with a really small example sans explication:

```
use BerkeleyDB; # this module has lots of firepower, see the doc for details
tie my %tiedhash, 'BerkeleyDB::Hash', -Filename => 'data.db'
  or die "Can't open data.db: $!\n";

$tiedhash{uid}      = 'dnb';
$tiedhash{FirstName} = 'David';
$tiedhash{LastName} = 'Blank-Edelman';

untie %tiedhash;
```

Next time you tie to that database, you can retrieve the values you want for that key using standard Perl hash syntax.

Let’s see two less conventional ways to deal with key–value pair storage. The first is to use a very cool module you may not have seen before. `DBM::Deep` is an almost entirely pure Perl module that implements an entire database engine. This engine stores its data in a portable format, actually implements ACID transactions, and is pretty darn fast even with very large datasets.

(Note that the module used to be entirely written in Perl until a recent revision started to depend on the `FileHandle::Fmode` module. If you look at the CPAN bug reports you’ll find a pure-Perl replacement for that dependency if this is important to you.)

`DBM::Deep` can be used via `tie()` just like in our `BerkeleyDB` example, but then you’ll lose an additional piece of magic: multi-level array and hash support. Yup, we’re essentially combining the previous sections of this article with the current one because we can now write:

```
use DBM::Deep;

my $dbdeep = DBM::Deep->new('data.db');

$dbdeep->{uid}      = 'dnb';
$dbdeep->{name}     = { 'FirstName' => 'David',
                      'LastName'  => 'Blank-Edelman'};
```

Then in another program (or another part of the same program during a different program run), you can write:

```
print $dbdeep->{name}->{LastName}
```

and it will retrieve and print my last name. Pretty nifty!

The second, less conventional approach I want to mention but not really delve into is less of a storage technique and more of an optimization technique. If you have data that you only need to keep for a short amount of time (e.g., Web sessions) or have accessible in memory just while it is actively used, you should take a look at the various caching frameworks available. Cache::Cache is the most heavily used, but there are others. The CHI framework in particular seems to be up and coming and worth considering. Many of these frameworks will automatically serialize more complex data structures for you when you attempt to store that data similar to what Storable will do for you.

With all of these frameworks, you basically set up the kind of cache you want to use (in memory, stored as files, in shared memory, using a separate custom daemon such as memcached, etc.) and what flavor of cache you want (e.g., should the cache keep itself under a certain size?). Once you've picked your cache "backend" you then have the opportunity to add things to the cache using some sort of `set()` operator. This set operator usually lets you specify the amount of time that data should persist. Expired data will be flushed from the cache either automatically or at your command. To use the cache itself, you tend to write code that looks like this:

```
is item in cache?
  yes: retrieve and use data
  no: work harder to get data (pull from database, make a new one, etc),
     store the data in the cache and then return the new value
do stuff
purge the cache when the main work is done
```

Modules such as those in Cache::Cache and CHI handle all of the behind-the-scenes work of maintaining the cache for you, and everyone wins.

---

## Storing Stuff in SQL Databases

---

If you mention SQL and databases in the same sentence, Perl programmers will reflexively just say "DBI." The "DataBase Independent interface for Perl" ([dbi.perl.org](http://dbi.perl.org)) is one of the great gifts Perl, via Tim Bunce, has given the world.

It is basically an API that allows you to write database-agnostic code that will work independently of whatever database engine you are using today. It is a tremendous relief to be able to write code that will work unchanged with Oracle, MS-SQL, MySQL, Postgres, etc. Simple DBI code looks like this:

```
use DBI;

my $uid = $ARGV[0];

my $dbh = DBI->connect(
    "DBI:mysql:database=usenix;host=localhost", 'user', 'password')
    or die "Couldn't connect to database: " . DBI->errstr;

my $sth = $dbh->prepare('SELECT * FROM users WHERE uid = ?')
    or die "Couldn't prepare statement: " . $dbh->errstr;
$sth->execute($uid)
    or die "Couldn't execute statement: " . $sth->errstr;

my @results = ();
while (@results = $sth->fetchrow_array()){
    print join ('\n',@results);
}
```



```
$sth->finish;  
$dbh->disconnect;
```

We connect to a specific database by providing the database engine name, database name, server host name, and authentication information. This returns a database handle through which we'll communicate with that database. We then pre-parse the SQL statement we're going to send using `prepare()`. (Although this last step is not strictly necessary, it will prove useful for better performance when our code gets more sophisticated.) This yields a statement handle. The statement handle will give us a means to run that query with `execute()` and return the result via `fetchrow_array()`. `fetchrow_array()` returns the results of that query to an array, one result row at a time. Once done, we close both our statement and database handles and we're done.

There's much more we could look at around DBI (several published books' worth, to be exact) but we're going to leave that behind so we can look at a lesser-known twist on this approach: `DBD::SQLite`.

All DBI-compatible database engines have a database-dependent driver called a DBD written to use them. The particular DBD I'd like to call your attention to is called `DBD::SQLite`. It not only provides the driver for the SQLite database engine ([www.sqlite.org](http://www.sqlite.org)) but it also builds the actual runtime libraries it needs so you don't have to install anything extra to start using it. SQLite is a really wonderful lightweight SQL database engine that does not require a server to run. In some ways it is like the BerkeleyDB libs mentioned earlier, except that one can actually throw a fairly decent subset of SQL at it and it will do the right thing.

What does this mean to you? You can write reasonably portable Perl code using DBI and SQL without a server. Your database will live in a single file on your machine. This is wonderful for prototyping code or creating small projects that don't need the power a full database server would bring (and the concomitant hassles of setting it up).

Code that uses `DBD::SQLite` looks like any other DBI code. Instead of:

```
my $dbh = DBI->connect(  
    "DBI:mysql:database=usenix;host=localhost", 'user', 'password')  
    or die "Couldn't connect to database: " . DBI->errstr;
```

you write:

```
my $dbh = DBI->connect("dbi:SQLite:dbname=datafile.sql3", "", "")  
    or die "Couldn't connect to database: " . DBI->errstr;
```

and everything proceeds as normal from there.

And with that tip, I'm running out of, err, space. Take care, and I'll see you next time.

PETER BAER GALVIN

## Pete's all things Sun (PATS): the state of ZFS



Peter Baer Galvin ([www.galvin.info](http://www.galvin.info)) is the Chief Technologist for Corporate Technologies, a premier systems integrator and VAR ([www.cptech.com](http://www.cptech.com)). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is coauthor of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide.

[pbg@cptech.com](mailto:pbg@cptech.com)

WE ARE IN THE MIDST OF A FILE SYSTEM revolution, and it is called ZFS. File system revolutions do not happen very often, so when they do, excitement ensues—maybe not as much excitement as during a political revolution, but file system revolutions are certainly exciting for geeks. What are the signs that we are in a revolution? By my definition, a revolution starts when the peasants (we sysadmins) are unhappy with the status quo, some group comes up with a better idea, and the idea spreads beyond that group and takes on a life of its own. Of course, in a successful revolution the new idea actually takes hold and does improve the peasant's lot.

`login`: has had two previous articles about ZFS. The first, by Tom Haynes, provided an overview of ZFS in the context of building a home file server (`login`; vol. 31, no. 3). In the second, Dawidek and McKusick (`login`; vol. 32, no. 3) discuss ZFS's fundamental features, as well as the porting of ZFS to FreeBSD. This month I won't repeat those efforts, but, rather, continue on from that ZFS coverage to complete the list of ZFS features, discuss field experiences and the ZFS adoption status, and try to see into the future of ZFS. The revolution started in November 2005 when ZFS was made available for download. Now let's check in with the revolution and see how it is progressing.

### The Current Feature List

This detailed summary of all of the current ZFS features can serve as a checklist to determine whether ZFS can do what is needed in a given environment. The following feature list is accurate as of April 2008. All of the features are included in the current commercial Solaris release (Update 4, also known as 11/07).

- Disks or slices are allocated to storage “pools” in RAID 0, 1, 0+1, 1+0, 5 (RAID Z), and 6 (RAID Z2) formats. (Note that RAID Z and Z2 are optimized over the standard RAID levels to remove the RAID 5 “write hole.”)
- File systems live within a pool and grow and shrink automatically within that pool as needed.

- File systems can contain other file systems. (Think of ZFS file systems as being more like directories, with many new attributes.)
- File system attributes include compressed, NFS exported, iSCSI exported, owned by a container (a “dataset”), mount point, and user-definable.
- Copy-on-write allocation, data, and meta-data are always consistent on disk; no “fsck” is needed.
- There is end-to-end data and meta-data integrity checking via a Merkel tree structure; important blocks are automatically “dittoed,” giving data protection far beyond other solutions.
- The system is “self-healing”: If corrupt data or meta-data is found and a noncorrupt copy exists, the corrupt version is replaced with the good version.
- Highly efficient snapshots and clones (read-write snapshots) can be made.
- One can roll back a file system to a given snapshot and promote a clone to replace its parent file system.
- There are quotas to limit the size of a file system and reservations to guarantee space to a file system.
- One can make full and incremental backups and restores to a file or between two systems (replication) via send and receive commands.
- There is support for multiple block sizes, pipelined I/O, dynamic striping, and intelligent prefetch for performance.
- Fast re-silvering (re-mirroring) is allowed.
- ACLs are in NFS V4/NTFS style.
- Adaptive “endian-ness” allows import and export of ZFS pools between varying-architecture systems; new data writes are in the native format of the current system.
- Requestable pool integrity checks (scrubs) to search for corruption in the background can be made.
- Configuration data is stored with the data (e.g., disks know what RAID set they were a part of).
- The system can make use of hot spares, shareable between pools, with automatic RAID rebuild upon disk failure detection
- ZFS is implemented in two major commands (with lots of subcommands).
- Very, very large data structures (up to 128 bits) are allowed, with no arbitrary limits (files per directory, file systems, file size, disk per pool, snapshots, clones, and so on).
- ZFS is open source and free.
- It has been ported to FreeBSD, FUSE, and Mac OS X Leopard (read-only).

There are many articles about how to use ZFS and take advantage of these features, which, again, I won't repeat here [1].

---

## ZFS Status

---

File system revolutions, as opposed to political revolutions, happen much more slowly and tend to be bloodless (although losing files can be very painful). A file system gradually gains trust as direct and shared experiences gradually build into an “it works” or “it loses files” general consensus. At this point in the life of ZFS it has passed that test for many people. The testing performed during its development and continuing every day is rather awe-inspiring, as described in Bill Moore's blog [2]. Reading through the posts at the ZFS forum [3] suggests that ZFS is being used a lot and at many sites, mostly very successfully. There is quite a lot of discussion of current and future features, as well as a few “something bad happened” discussions. Those

posts, while revealing occasional problems, show in summary that ZFS is rock-solid, especially for such a new, innovative, core piece of software.

The next step in adopting new technology is support by other software products, such as backup/restore tools, clustering, and even general-purpose applications such as databases. Other vendors' products might work fine, but without a stamp of approval, commercial sites are very unlikely to use the new file system and risk being off of the support matrix. At first, of course there was zero non-Sun support for ZFS, but that situation has improved greatly. All major backup products support ZFS, and it is also now supported by Veritas and Sun cluster. Most applications are independent of the underlying file system, but those that do care, such as Oracle, are generally supporting ZFS.

Before a new technology can be put into top-priority environments (such as production OLTP database servers), it must perform as well as or better than the technology it is replacing. Performance tuning is usually a never-ending effort (or at least not ending until the product life ends). ZFS is no exception, and it is exceptionally young compared to the other production file systems such as UFS and Veritas Storage Foundation (the VXVM volume manager and VXFS file system). The only performance question more controversial than "Which is faster?" is "How do you prove which is faster?" The debate in general is continuous and unsolvable. There are certainly claims that ZFS is very fast, and faster than other file systems for certain operations. There are also counter claims that ZFS is slower at other operations. The StorageMojo blog has been following the debate and is a good site to watch. One posting [4] is especially interesting, showing ZFS compared with hardware RAID.

In my opinion, ZFS is a fundamentally fast volume manager/file system. It gets many aspects of storage very right. However, it cannot in software make up for one feature of hardware RAID: NVRAM cache. Nonvolatile cache allows writes to memory to take the temporary place of writes to disk. Because memory is much faster than disk, NVRAM is a great performance win. So, for example, using a Sun server containing local disk as a NAS device will have worse random write performance than a good NAS appliance that contains NVRAM. One solution to this performance delta is to use hardware RAID arrays that include NVRAM to provide individual LUNs to a system, and then use ZFS to create RAID sets and manage those LUNs as if they were individual disks. The NVRAM provides a performance boost, while all of the ZFS features remain available. Of course, in cases where random write performance is not critical (say, media servers and backup disk pools) NVRAM is not needed and ZFS is fine using local disks.

Aside from these performance challenges, ZFS is doing well at many sites. It is mostly being used in development, testing, and utility environments but is making its way into production. As more improvements are made to the feature set and more field experience drives acceptance, ZFS use should greatly increase.

---

## The Future Feature List and the Future of ZFS

---

In spite of the massive list of ZFS features, there are still features that are desirable but not yet included in ZFS.

Probably the most important and useful would be the use of ZFS as the root file system, which would enable all of the above features for system administration. Imagine creating an instant snapshot of "/" and installing a patch in "/" and rolling the system back to that snapshot if the patch did not have the desired effect. Or imagine creating a snapshot every minute of the day

to allow easy detection of changed files and restoration to the file's previous state. Once ZFS can be used as a root file system, zones will also be able to use ZFS for their root file systems. (Actually they already can have a ZFS root, but such a system cannot be upgraded to the next release of Solaris, as the upgrade code does not understand ZFS.) Fortunately, bootable ZFS has been added to OpenSolaris and should make its way into the commercial Solaris release in the future. It can be used currently via the various non-commercial Solaris distributions [5].

Native CIFS support is in OpenSolaris as well, so expect CIFS exporting as a future feature—no Samba (or other dancing) required.

Encryption is complicated to implement for a file system, mostly because of the key management. There is currently a ZFS encryption project underway for OpenSolaris [6], and alpha test code has already been released.

Removing disks (aside from hot spares) from a pool is an obvious need. Also missing is the ability to expand the size of a pool by adding individual disks. Currently, a set of disks can be added, for example as a RAIDZ set concatenated to a RAIDZ pool, and ZFS will cleverly stripe data across the two RAIDZ sets to maximize performance. However, adding a single disk to a ZFS pool simply has the disk concatenated to that pool, leaving for example a RAIDZ-plus-a-concatenated-disk pool rather than the much more desirable RAIDZ-expanded-to-include-the-new-disk pool.

The current quota system is a per-filesystem rather than a per-user one, which has pros and cons. There do not seem to be any plans to implement per-user quotas as well.

Scrubbing is currently done as a low-priority I/O task, but even lower-overhead user-definable scrubbing rates (in which a pool is gradually scrubbed over a period of time) are already planned for Solaris.

The ZFS intent log (ZIL), the place where ZFS stores changes that are to be applied to a ZFS pool, currently resides within the disks of that pool. OpenSolaris already includes the ability to put that log somewhere else, helping to improve write and especially random write performance. A natural next step would be to use a device dedicated to the ZIL. This could be an NVRAM device (and at least one company makes a PCI-based NVRAM card for Solaris) or a flash-based device that has been optimized for writes.

Another performance improvement could come from Brendan Gregg (of DTraceToolkit fame). He has added support in OpenSolaris for a level-2 adaptive replacement cache (L2ARC). This allows buffers to be evicted out of DRAM into a storage medium that fits between DRAM and the disk in terms of capacity and performance. The L2ARC and flash-based solid state drives (SSDs) seem to be a natural fit, and this is certainly an area to watch over the next 12 to 18 months.

ZFS integration with Mac OS X has been underway for quite a while, and read/write ZFS is available for testing [7].

For performance, many database sites use direct I/O, which bypasses the buffer cache and file locking, essentially telling the operating system and file system to get out of the way of database I/O. This feature does not exist within ZFS, and database performance on ZFS is currently a work in progress. For the latest information on ZFS performance see the ZFS Best Practices Guide [8].

Certainly the future is wide open for innovation around and integration of ZFS. As one example, have a look at the Service Management Facility (SMF)

services being written by Tim Foster to automate snapshots and backups of ZFS file systems [9].

---

## Next Time

---

Hopefully this ZFS status check has cleared up some questions and given guidance as to whether ZFS is now or will be in the future the right file system for your systems. The revolution seems to be well on its way and the Bastille is starting to fall. ZFS rising in its place seems inevitable and desirable.

In the next PATS column I'll discuss something that is basic, important, but frequently overlooked or done ad hoc: system problem analysis. What steps are the right ones to analyze a system that is having a problem, be it reliability or performance? My hard-learned cookbook may be a useful addition to your own techniques.

---

## REFERENCES

---

- [1] Recommended articles about ZFS: <http://opensolaris.org/os/community/zfs/intro/>; <http://www.samag.com/documents/s=9950/sam0602j/0602j.htm>; <http://www.samag.com/documents/s=9979/sam0603h/0603h.htm>; [http://www.sun.com/bigadmin/features/articles/zfs\\_overview.jsp](http://www.sun.com/bigadmin/features/articles/zfs_overview.jsp); [http://www.sun.com/bigadmin/features/articles/zfs\\_part1.scalable.jsp](http://www.sun.com/bigadmin/features/articles/zfs_part1.scalable.jsp).
- [2] Bill Moore's ZFS blog: <http://blogs.sun.com/bill/category/ZFS>.
- [3] ZFS Forum: <http://www.opensolaris.org/jive/forum.jspa?forumID=80>.
- [4] StorageMojo blog entry comparing hardware RAID to ZFS performance: <http://storagemojo.com/?p=441>.
- [5] OpenSolaris downloads: <http://opensolaris.org/os/downloads/>.
- [6] Alpha ZFS encryption support: <http://www.opensolaris.org/os/project/zfs-crypto/>.
- [7] ZFS for Mac OS X: <http://trac.macosforge.org/projects/zfs/wiki/>.
- [8] Latest on ZFS performance: [http://www.solarisinternals.com/wiki/index.php/ZFS\\_Best\\_Practices\\_Guide](http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide).
- [9] Automating snapshots: [http://blogs.sun.com/timf/entry/zfs\\_automatic\\_for\\_the\\_people](http://blogs.sun.com/timf/entry/zfs_automatic_for_the_people).

DAVID JOSEPHSEN

## iVoyeur: Admin, root thyself.



David Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and 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](mailto:dave-usenix@skeptech.org)

**DID YOU HAPPEN TO SEE THE LATEST** Die Hard movie? The one where the interwebs are broken? Well if you did, it probably annoyed you quite a bit. It's a pretty typical Hollywood blockbuster take on computers and the nerds who love them, and they pretty typically get it all horribly wrong. I'm kind of atypical when it comes to these sorts of films: I actually rather like them. I think I find something endearing in Hollywood's belief in magic [1]. But when I see them, I try to do so alone or in the company of nerds; otherwise someone I'm with will invariably ask, à la Homer Simpson [2], "Computers can do that?!" (or some variation thereof).

I imagine it's the same pain periodically felt by paleontologists who've been dragged to *Jurassic Park*, or um . . . pagans at Harry Potter? Anyway, I don't like to disappoint folks, and since the answer to the Homer question is almost always, "Well, not really," it's better to just avoid the situation when I can. But much as I hate to harsh on their newfound interest in computer security, I can't help but chuckle to myself at how disappointed they'd be if they knew the truth about the security capabilities of today's computers.

I'm not talking about Windows being vulnerable to the sploit of the week, or even theoretical design issues such as mandatory access control. I'm talking about simple functionality that everyone outside of our community probably assumes is there and would be surprised to find out is not. For example, if you asked a random movie producer whether he or she thought a computer kept a record of all the changes made to any given file on the file system for the past week, I think you'd find that most of them would give an emphatic yes.

How many times was `/etc/foobar` changed, by whom, and when? This is a problem I think most people would assume has been solved by now. But in reality, this type of auditing information is surprisingly difficult to come by. Indeed, very good books [3] have been written on the subject of teasing this type of stuff from a file system offline and after an attack. To pull it off in real time you need to audit changes to every file in the file system. The audit records need to include who, what, and when, and they need to be captured and written in a way that is difficult to bypass or modify after the

fact. Add to that UNIX's rather murky definition of a "file," and this isn't a solved problem. There are several solutions, and they're all far from perfect.

So since that example ties in so well with the filesystem theme of this issue, I'd like to take a look at some ways to monitor changes to the file system, including a method you may not have considered, namely using kernel instrumentation such as DTrace or SystemTap to audit kernel vfs read/write calls. I've become rather fond of the method lately for several reasons, and I hope it will prove useful to you.

By far the most popular way to do this sort of thing normally is with a filesystem integrity checker such as Tripwire [4] or Samhain [5]. These programs are polling engines; they usually run as a daemon and periodically wake up to recursively check the file system against a database of hashes. In practice this works fairly well. They have a reasonable overhead once the hash database is created, they capture changes to file metadata such as permissions, ownership, and modification dates, and they are pretty good at staying out of the way.

I've used Samhain in my production environments for a few years now, and I don't hate it. It has some rudimentary rootkit detection capabilities on Linux and Open/Free BSD via the `/dev/kmem` file, can hide itself from script kiddies, and does a good job of finding and notifying you of changes to the file system. Although it wants badly for you to use its client/server model, it will play nicely with your existing tools such as syslog, Splunk, databases, and SEC/Logsurfer, if you ask it politely.

The biggest thing I don't like about the filesystem integrity checkers has got to be that they can't tell you who changed the file. Ideally I'd like to know the pid and uid of the thingy that changed a given file. Since integrity checkers simply wake up once every so often and compare files against MD5 hashes in a database, they only know that a file has changed and not who changed it. The question of "who" is what you might call a fundamental piece of information.

A less important nit is that the change notifications are delayed by the length of the polling interval. Depending on your situation, it could take some time before you know what files have changed, which can be frustrating when you're dealing with an intrusion in real time. The integrity checkers can obviously only notify you of changes to "normal" files; changes to special files such as sockets and block devices cannot be detected this way. Finally, the integrity checkers are somewhat high in the stack, so it's possible that they could be bypassed for certain types of events. For example, they won't be able to notify you of read events if you have "noatime" set in `fstab` (because they won't be able to see a difference in access times in the file metadata).

One way to solve some of these problems, including the polling interval delay, is to use the kernel's `inotify` subsystem. The `inotify` subsystem provides user-space programs with notifications of file change events. It's used primarily by content-indexing tools such as Beagle [6], but there's no reason it couldn't be used to log changes to files systemwide. There are several user-space implementations, including some shell tools called "inotifytools" [7]. These include a program called "inotifywait" that basically blocks on `inotify` events for a given directory or set of directories and provides event details to `STDOUT`. I haven't used `inotifytools` to recursively monitor `/`, so I don't know how much overhead it might incur, but from my limited experience it seems pretty scalable. It's also a bit closer to the kernel, so it's more difficult to fool. Unfortunately `inotify` doesn't solve the "who" problem. The pid/uid of the



changing process is not one of the pieces of information passed by the kernel to user space. Bummer.

A somewhat more indirect approach might be to use tty snooping. Solutions of this type simply listen in on input from the ttys of the machine, thereby logging the actions of users. There are all sorts of implementations here; most of them are shell replacements or patches to existing shells such as `bofh-bash` and `ttysnoop` [8], but some are more elegant, kernel-space tools such as `Sebek` [9]. These tty sniffers work very nicely when a user can't simply launch another shell to bypass them. They solve the "who" problem, giving you granular detail of what changed and sometimes even the content of the change, depending on how the file was edited. These can induce some overhead, however, and, since they tend to be user-centric, they might be bypassed by non-interactive programs or system processes.

Finally, just about every system has a kernel-space auditing subsystem: SELinux and the kernel audit subsystem for Linux, BSM auditing for Solaris et al. These are used to great effect by folks who know them well, and they are probably the closest thing to the "right" answer, but they aren't necessarily focused on file accesses and can generate metric tons of auditing information. They can also be difficult to use and maintain and rarely play nicely with centralized tools such as OSSIM or Syslog.

So let's take a look at the kernel probes approach I've been playing with lately. I should disclaim that the DTrace folks have explicitly warned against the use of DTrace for security auditing [10] because DTrace might drop events if the system becomes overloaded. This is pretty much a deal breaker for DTrace in this context at the moment, but I have a feeling DTrace will eventually be a useful solution here. So for now I'll focus on the SystemTap script in Figure 1.

If you aren't familiar with SystemTap [11], it is comparable to DTrace but only runs on Linux. There are already healthy religions built up around both tools, and I'll probably get flamed for that last sentence, so I'll leave it at that and let you work out the differences for yourself. SystemTap scripts are written in an awk-like language, parsed into C by an interpreter, compiled into a kernel module, and finally loaded into a running kernel. Once loaded, the module can trace system calls and broker information between kernel and user space. It's a fascinating and useful tool, but it requires some understanding of the kernel internals, or at least a good handle on C and a willingness to dig around at the kernel headers to use.

SystemTap requires that `CONFIG_DEBUG_INFO`, `CONFIG_KPROBES`, and optionally `CONFIG_RELAY` and `CONFIG_DEBUG_FS` be enabled in the kernel. It also assumes some Red Hat-style symlinks to the running kernel source, so check the README if you're installing it on a box that's not Red Hat. One of the more interesting features of SystemTap is the ability to inject blocks of C directly into the system tap script. In the script in Figure 1, I have a function that is written in raw C, but it is called from within the SystemTap scripting language.

The purpose of the script is to probe kernel `vfs_read` and `vfs_write` calls and return information about them to `STDOUT`. This approach has several advantages. First, it takes advantage of the fact that everything in UNIX is a file, so reads and writes to sockets, fifos, block files, etc., will all be captured. Second, since we are writing the instrumentation, we can ask for whatever pieces of information we want, including the pid/uid of the entity making the file access. Next, the probe is fairly limited in scope, so we get what we want and nothing we don't, and with a very small overhead. Finally, it plays nicely with any of your other tools that take `STDIN`. The thing

I might like the most about it is that it's actually kind of fun. It isn't every day I get to rummage about in /usr/src/linux/include, and I learned a lot about Linux in the process.

```
#from an error message I got when I misspelled a struct name,
#the structs avail to stap in the vfs_(read|write) context are:
#f_u f_dentry f_vfsmnt f_op f_count f_flags f_mode f_pos f_owner
#f_uid f_gid f_ra f_version f_security private_data f_ep_links f_ep_lock f_mapping

function get_path:string (da:long, va:long) %{
    char *page = (char *)__get_free_page(GFP_ATOMIC);
    struct dentry *dentry = (struct dentry *)((long)THIS->da);
    struct vfsmount *vfsmnt = (struct vfsmount *)((long)THIS->va);
    snprintf(THIS->__retvalue, MAXSTRINGLEN, "%s", d_path(dentry, vfsmnt, \
        page, PAGE_SIZE));
    free_page((unsigned long)page);
%}

probe kernel.function ("vfs_write"),
       kernel.function ("vfs_read")
{
    dev_nr = $file->f_dentry->d_inode->i_sb->s_dev
    path=get_path($file->f_dentry, $file->f_vfsmnt)

    subPath=substr(path,0,4)
    if((subPath != "/dev") && (dev_nr == (8 << 20 | 3)))
        printf ("%s(%d,%d) %s\n", execname(), pid(), uid(), path)
}

```

**FIGURE 1: SAMPLE SYSTEMTAP SCRIPT**

So let's step through this script starting with the function declaration in line 1:

```
function get_path:string (da:long, va:long) %{
```

As you can see, the function declaration is *not C*. The function is declared in the SystemTap language; embedded C blocks are denoted by `%{` and `%}`. The second thing you might notice is that both variables are declared long even though, when the function is called below, it is passed pointers to structs. This is because all pointers are cast to longs by the interpreter, so they need to be declared as such in the function declaration and typedef'd back into struct pointers later. The entire purpose of this function is to call the `d_path()` function to return the full path of the file in question. So the next three lines set up the required arguments for `dpath()`:

```
char *page = (char *)__get_free_page(GFP_ATOMIC);
struct dentry *dentry = (struct dentry *)((long)THIS->da);
struct vfsmount *vfsmnt = (struct vfsmount *)((long)THIS->va);
```

There may have been a way to directly refer to the file's path with SystemTap built-ins, but if there is, I couldn't find it. The next two lines call `dpath()` and free the page we allocated:

```
snprintf(THIS->__retvalue, MAXSTRINGLEN, "%s", d_path(dentry, vfsmnt, \
    page, PAGE_SIZE));
free_page((unsigned long)page);
```

Below this function is the SystemTap script proper. The first two lines tell SystemTap that we are going to probe for `vfs_(write|read)` calls:

```
probe kernel.function ("vfs_write"),
       kernel.function ("vfs_read")
```

Any number of comma-separated probes may be declared in a probe statement. The block immediately following the probe statement includes the instructions we want to carry out for each call we capture. In this script the first thing we do is dereference the device number of the current file:

```
dev_nr = $file->f_dentry->d_inode->i_sb->s_dev
```

The dentry struct is defined in `/usr/src/linux/include/linux/dcache.h`. You can directly reference anything in a struct from SystemTap without needing to resort to embedded C. It's generally preferable to avoid C when you can, because SystemTap uses kprobes' considerable safety and sanity checks as long as you stay within the bounds of its interpreted language. I should also save you some time and note that when you are dereferencing string data from kernel space in the SystemTap language, you need to use the `kernel_string()` conversion function or you'll end up with a typedef'd long once again. For example, we could directly dereference the name of the file from within the SystemTap language like so:

```
f_name=kernel_string($file->f_dentry->d_name->name)
```

Next we call our `get_path` function to derive the full path of the file:

```
path=get_path($file->f_dentry, $file->f_vfsmnt)
```

I placed some filters in here to give you a rough feel for the syntax you can use to filter probe data. Generally, the SystemTap language has all the iterative loops and conditionals you'd expect. In the line:

```
subPath=substr(path,0,4) if((subPath != "/dev") && (dev_nr == (8 << 20 | 3)))
```

the first check filters out changes to files in the `/dev` directory (`grep -v '^/dev'`). The second check filters out everything except files that reside on the third SCSI volume (`/dev/sda3`) as defined by the device number (major 8, minor 3). You can derive the device number for files on a given partition by cd'ing to that partition and performing a `stat -c '%D' *`. Without any filters you get *every* read and write happening on the system. If someone moved a mouse, for example, you'd see writes to `/dev/psaux`.

Finally, the `printf` built in prints our data to STDOUT:

```
printf ("%s(%d,%d) %s\n", execname(), pid(), uid(), path)
```

The first three arguments are also built-in functions: `execname` returns the name of the program making the change (`xterm`, `vi`, etc.), and `pid()` and `uid()` are self-explanatory. I placed the reads and writes in the same probe statement to show you it's possible, but if we wanted to differentiate reads from writes, our script could declare the probes separately, giving each its own instruction block. The read instruction block, for example, could have a `printf` that said `read: %s(%d,%d) %s\n`. We execute the script like so:

```
sudo stap -g figure1.stp
```

or, even better:

```
sudo stap -g figure1.stp | logger -t vfstprobe -p kern.info &
```

The `-g` is for "guru" mode, which allows the execution of embedded C. As I alluded to earlier, guru mode gives you the rope to hang yourself with by turning off quite a bit of sanity checking. This sort of thing should probably not be done lightly. If you are new to SystemTap and are considering running your code on production systems, I'd recommend running it by the gang on the SystemTap mailing list (as I did with this script).

I think kernel probe tools show a lot of potential to solve some of our nagging auditing needs. I've begun running a script like this one under dae-

montools [12] on a few of the boxes in our staging environment, with favorable results. I'm hoping it will eventually replace a few auditing tools we're using now, and I'd really like to expand it to include some other auditing gaps I have. It's not Blockbuster material, probably, but it is close enough to magic for the folks I hang out with.

Take it easy.

---

## REFERENCES

- [1] The “magic” entry in the jargon file: <http://catb.org/~esr/jargon/html/M/magic.html>.
- [2] Homer Simpson's classic quotation: <http://www.eventsounds.com/wav/cmputers.wav>.
- [3] *Forensic Discovery* by Dan Farmer and Wietse Venema: <http://www.porcupine.org/forensics/forensic-discovery/>.
- [4] Tripwire: <http://www.tripwire.com/>.
- [5] Samhain: <http://la-samhna.de/samhain>.
- [6] Beagle: [http://beagle-project.org/Main\\_Page/](http://beagle-project.org/Main_Page/).
- [7] See libinotifytools for a scriptable inotify implementation: <http://inotify-tools.sourceforge.net/api/index.html>.
- [8] ttysnoop: <http://freshmeat.net/projects/ttysnoop/>.
- [9] Sebek: <http://www.honeynet.org/tools/sebek/>.
- [10] DTrace: [http://www.solarisinternals.com/wiki/index.php/DTrace\\_Topics\\_Limitations](http://www.solarisinternals.com/wiki/index.php/DTrace_Topics_Limitations).
- [11] SystemTap: <http://sourceware.org/systemtap/>.
- [12] daemontools: <http://cr.yp.to/daemontools.html>.

ROBERT G. FERRELL

## /dev/random



Robert G. Ferrell is an information security geek biding his time until that genius grant finally comes through.

[rgferrell@gmail.com](mailto:rgferrell@gmail.com)

IT WASN'T THAT TERRIBLY LONG AGO, in the accelerated timeline of technologies, that the term "storage device" used in an IT context referred to little plastic rectangles with spring-loaded sliding metal covers, misleadingly flexible miniature 45 RPM records in black sleeves, or a variety of glorified cassette tapes ranging from slim 'n' sexy to stout 'n' chunky. Whatever their size and physiognomy, they were all pretty easily identifiable as gadgets you stick into a slot on a computer and pull out later covered with magnetic zeroes and ones (or sometimes with strawberry jelly and slightly melted, if you tried doing this before your morning coffee and got the wrong slot).

The digital archiving landscape has evolved radically since those simpler days, it seems. USB and flash memory together have conspired to render darn near every little piece of junk lying about the house capable of duplicating the Library at Alexandria with a couple of gigabytes to spare. USB memory dongles can now be found nestled in athletic shoes (sneaker net: the next generation), in ID badge holders, in lighters, in pocket knives, in plush toys, on keychains, on Donner, on Blitzen . . .

I can't help thinking that there is still a whole lotta fertile soil to be tilled in the "things you can plug into a USB port" field, though. Let's move beyond the foam missile launchers, heated gloves, and mini lava lamps and get to the really good stuff, such as personal laser show projectors, tasers for zapping offensive office mates, customizable electronic filters that let you disguise your voice (for making anonymous calls over VoIP), and adapters for recharging the aforementioned blinky LED running shoes. How about tiny deep-fryers for crafting one french fry at a time? Or a little spray gun for applying hand/sun lotion? Maybe even a miniature branding iron for those craving DIY monochrome tattoos would be handy too.

Admittedly, all of this has precious little to do with storage devices, which is the direction I thought I was headed, but nowhere in my contract does it say I have to come up with a relevant topic sentence and stick to it. Nowhere in my contract does it say anything at all, actually, because I don't have one, but even if I *did* have one, I doubt it would be capable of speech. Are you starting to understand why I

titled this column “/dev/random”? The “dumb” part has probably been fairly evident all along, I’ll admit.

OK, I’m over it.

Acknowledging without further comment the fact that storage devices large and small are proliferating madly like bored bacteria in an open package of processed beef salivary glands and lymph nodes (otherwise known as “hot links”), let us examine some of the underlying tissue . . . er . . . issues.

Given that the old-fashioned habit of reading is rapidly assuming the mantle of taking the fringe-top surrey down to watch the perspirational semi-clothed gentlemen driving in an afternoon’s worth of railroad stakes, one must in all lucidity wonder what, exactly, it is that so many people are so keen to store in their USB-enabled corkscrews. Sports trivia? Recipes? A few emergency episodes of their favorite television program? (You know, in case they get stranded in the back of a laptop-equipped taxi during a hurricane evacuation.) Bird calls? Maps and floor plans to the home of every single subscriber to *The Journal of Privacy Protection*? Their personal genome? Their pit bull’s genome? I Dream of Genome?

Whatever the reasons, I think it’s safe to say that the storage device craze is only going to get sillier. At least I’m going to do my part. Here are a few pre-emptive strike product suggestions for those flash memory manufacturers whose company names I am not phonologically limber enough to pronounce without sounding as though I might have bird flu.

- RAM-a-Lam-a-Ding-Dong: This little widget lets you download and install your choice of doorbell tones. Now your elegant neoclassical portico can emit “Dark Side of the Moon” at 120 dB. Get rid of solicitors and other household pests the fun and easy way. May cause structural damage.
- The DDRAMikin: This would be perfect for increasing server throughput while serving crème brûlée. Pick up the optional USB brazing torch for added convenience (2.5 farad capacitor not included).
- FlashFlash: This flashlight with lens and embedded memory automatically captures images of whatever’s being illuminated. It’s handy for proving to mom that there really was a monster in the yard last night, or maybe that was just your older sister coming home from the party a little late.
- The RAMBow: This beautiful and thoughtful accessory allows you to include whatever message you like to accompany your gift, in any media format you choose. It is especially useful for those times when YouTube just isn’t personal enough.
- USBJammin: This universal adapter converts, well, anything into a USB memory stick. It works with tea cozies, collectible figurines, pirate eye patches, die-cast trains, travel mugs, velvet Elvis paintings, cashew jars, mice, dice, egg cartons, paper towel dispensers, steering wheel covers, phonograph needles, gate hinges, most cosmetics containers, and over 3,250 additional common objects around the home.

*Some conversions will not function as expected. Possible choking hazard for children, adults, and certain of the larger iguanas. Bridge may ice before roadway. Offer void where unavailable.*

In closing, have you noticed that the once ubiquitous nine-pin port on personal computers is edging toward extinction? Coming up next on *Ohm’s Law and Order*: “USB, the Forgotten Serial Killer.”

NICK STOUGHTON

## update on standards: undue influence?



USENIX Standards Liaison  
[nick@usenix.org](mailto:nick@usenix.org)

**REGULAR READERS OF THIS COLUMN** probably know by now that I am a participant in a number of committees that run under the auspices of the ISO/IEC Joint Technical Committee on Information Technology, in the programming languages and their run-time environments area, known by insiders by the catchy title “ISO/IEC JTC 1/SC 22.” I sit in the Working Groups for POSIX, the Linux Standard Base, the C programming language, and the C++ programming language, as well as the special working group on language vulnerabilities and the top-level committee for administering all of these sub working groups.

Several years ago, that top-level committee gave me the unenviable task of handling all of the coordination and liaison required between POSIX and the programming languages that reference it. For example, there’s a big overlap between parts of the C standard and POSIX. My job is to make sure that both sides know what the other is thinking when considering a change to something in that shared space.

Now, with C it’s easy. We all know where the overlap is and how to handle the questions. But C++ is a different kettle of fish. POSIX isn’t written in terms of C++. Although POSIX and C both specify the `fopen()` call, the equivalent isn’t so obvious in C++.

But C++ is, as I’ve reported before, going through a revision. There are *lots* of new features going into the language. And for some of those features it is easier to draw parallels with POSIX (for example, multithreading). POSIX has long had a powerful set of thread APIs. C++ is adding some. Can we at least align the two so that C++ threads can be built on top of POSIX threads?

The POSIX working group saw some serious problems with implementing the proposals for C++ multithreading. The “Nick Stoughton” robot was wound up and pointed at the C++ committee with a message to make sure that its members realized there was a problem.

I did what I was asked to do.

The C++ working group was convinced that there was a problem, and its members collectively voted to change their proposal to remove the contentious thread cancellation wording. I thought about

standing on an aircraft carrier deck with a “Mission Accomplished” banner flying behind me.

At the same time, the POSIX working group decided that it would be a good idea to study creating a C++ language binding. Just as there is an Ada and a Fortran binding to POSIX, why relegate C++ to using the C interfaces, forsaking the strong type checking and object orientation and all that good stuff? A study group was formed, and a substantial number of people from the C++ working group (C++, *not* POSIX!) joined up with enthusiasm. This was their language, and their platform of choice. What could be better than such a marriage! The group did the necessary work to prove that there was a viable body to produce a standard sometime in the future (probably around 2012). They did the paperwork with the IEEE (one of the three participating bodies in the POSIX world) and formed the 1003.27 working group.

The 1003.27 working group then read through the table of contents of the current working draft of the C++ language revision document, looking for places where a POSIX language binding might touch on something that is already in the C++ draft. If it is already in the draft, and the ink isn't dry on the draft, can we influence the draft to make sure it

- provides the hooks needed for whatever POSIX extensions might be needed in the future?
- doesn't contain anything truly problematic with respect to POSIX?

Remember, the vast majority of these people in 1003.27 (the POSIX-C++ binding) are also members of the C++ working group. The draft of C++ they were reviewing was one that they had helped to write. I was a member of that group, and we came up with a relatively short list of issues.

Naturally, the messenger picked to walk into the C++ group with this list was yours truly.

There are only a few people in leadership roles in C++, and in general they are all deeply committed to doing a good job for the language. However, they do have their own agendas, and by and large, POSIX isn't a big part of that agenda. So being requested by a sizable minority of their working group to change or, worse, remove some of the wonderful new (untested and unimplemented) features of their draft certainly took them by surprise. And they have worked hard at one by one shooting down all of the requests made by 1003.27. They haven't succeeded yet, but when you consider that the simplest request (“Please could you reserve the namespaces `::posix` and `::std::posix` for our use”) generated about 90 emails in a week, you can see how hard they are fighting.

Building a standard is all about achieving consensus. But what is consensus? When do we know we have reached it? When everyone is exhausted talking about it, does the last voice win? Is it unanimity? In general, in all of the groups I have worked in, when we realize we have a contentious issue, the best thing to do is to omit the issue from the standard, however painful that might be to some. We have done this in POSIX. We have done it in C. We have done it in the LSB. So why is C++ so different that if one loud voice says, “I want this feature in my language,” we have to have it, even if 45% or 50% of the working group don't feel so strongly and another 45% or 50% feel more strongly the other way?

Every formal process I have studied has a means for reversing a previously made decision. Robert's Rules of Order has a substantial section on it. Voting something in at one meeting never stops us voting it out at the next! Within SC 22, this is one of the golden rules: “This is the decision we've made until we decide to change it.”

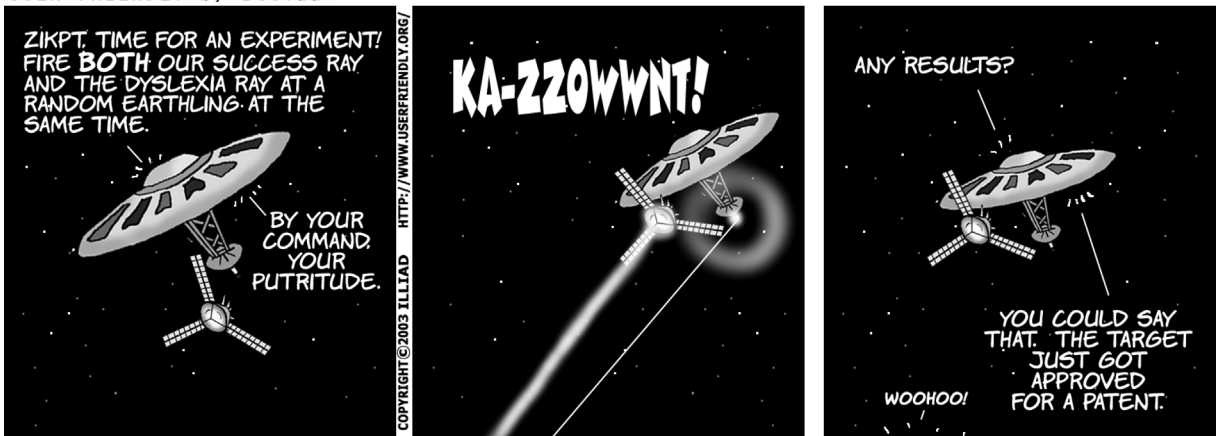


Saying, in C++, “we voted at the last meeting to add the `system_error` object, so we can’t remove it now” doesn’t fit that model.

When the message to change comes from an officer of your parent committee, you cannot simply ignore it. And if that officer happens to have a sizable contingent of your own working group agreeing with the message, you cannot ignore it. Complaining that POSIX is having an undue influence on the purity of the language is specious and pusillanimous. Certainly it is true that C++ runs on platforms other than POSIX. But POSIX is the only international standard platform on which the international standard language is going to be implemented.

And, in my role as POSIX liaison, I’m going to continue to rattle the bars at the C++ meetings. They can try to silence me, but they can never succeed! Pray for fewer “features”!

USER FRIENDLY by Illiad



## book reviews



ELIZABETH ZWICKY, WITH SAM STOVER  
AND EVAN TERAN

### HOST INTEGRITY MONITORING USING OSIRIS AND SAMHAIN

*Brian Wotring*

Syngress, 2005. 399 pages.  
ISBN 1-597490-18-0

Host integrity monitoring is not the sexy part of computer security. That would be network intrusion prevention, because networks are cooler than hosts, intrusions are cooler than integrity, and prevention is way cooler than monitoring. But host integrity monitoring has in general a strongly positive effectiveness/annoyance ratio, for some of the same reasons that it's not sexy. For instance, most of the time what it does is detect stupidity rather than malice; it's really good at tattling on authorized users who are doing things they ought not to. Fortunately for us all, your average network suffers more from stupidity than from malice.

So if you run a network of any size, you ought to be doing host integrity monitoring. It won't keep intruders out or make your hair shinier, but it will find intruders and make your site tidier. This book goes over why you ought to do host integrity monitoring, what kinds of things you need to monitor and why, and how to install and configure the most popular open source monitoring systems.

The author clearly has experience with monitoring systems and with practical security. He advocates for configurations that are as secure as practical, but no more so, with the authentic weariness of the person who has seen rookie administrators either try to do it "Right! At all costs!" or decide that all this security stuff is too much hassle anyway, and you might as well build your security products just like any other package.

Warning for ex-proofreaders and other people inclined toward pickiness: This book is edited to the usual Syngress standards, which is to say there's an error or two per page. The content is fine, but the itch to red-pen things gets overwhelming.

### TELLING AIN'T TRAINING

*Harold D. Stolovitch and Erica J. Keeps*

ASTD Press, 2002. 189 pages.  
ISBN 1-56286328-9

Training, like documentation, is one of those tasks that programmers and system administrators end up with for a number of reasons: The organization is too small to need a dedicated person, everybody is so accustomed to seeing it done badly that they don't realize how much better it could be done, or the managers are cynics who believe it's all pointless anyway, so they assign it to whoever complains about the lack as a punishment for complaining.

If you find yourself wanting to train people more effectively, I recommend this book. It captures the essence of modern training—which, like all good things, is simple in concept, difficult to implement—and communicates it vividly and practically. You will find good advice on how to decide what to cover, how to lay out the course, and what educators currently know about education and the brain. (As they point out, common sense is not your best guide to figuring out what works, and tradition is a whole lot worse.)

Do not run their advice on French verbs by your French teacher, however; I kept waiting for them to point out that oversimplification is a valid educational technique and they were doing it on purpose, and they never did. (There is more than one pattern for regular verbs in French. They classify regular verbs in -ir and -re as irregular. I believe we had already established my pedantic tendencies.)

### IMPLEMENTING ITIL CONFIGURATION MANAGEMENT

*Larry Klosterboer*

IBM Press, 2007. 216 pages.  
ISBN 0-13-242593-9

This is a book aimed at large sites, the kind of place that has an IT department that cannot all sit around the same table and eat pizza together, even when they rent the whole restaurant. Since I work for the kind of site where the IT department is measured in fractions of a person, it's not of immediate application for me. However, I've worked for enough large organizations to recognize the authentic voice of experience.

Only somebody who knows what he's talking about would address the question of how to do configuration management on a machine that nobody knows how to get into, but that can't be turned off. He gives the right answer, too: First, figure out if you can stick your fingers in your ears and pretend it doesn't exist. That's what everybody else is doing; why should you get the hot potato? Failing that, try very hard to eliminate it, because it is probably cheaper to replace it than to manage it.

If you need to implement IT Infrastructure Library (ITIL) configuration management, you definitely want this book. It assumes that you know something about ITIL but nothing much about running a large IT project. It's actually a nice overview of how to run any such project. The language is stuffier than it needs to be in places, reducing the readability, but the content is straightforward and practical and steers you through many of the minefields of big IT projects. For instance, it lays out clearly how to set up a pilot project—and what to do to save the entire project if the pilot fails.

#### **MINDSET: THE NEW PSYCHOLOGY OF SUCCESS**

*Carol S. Dweck*

Ballantine Books, 2006. 268 pages.  
ISBN 978-0-345-47232-8

Yes, it's a second nontechnical book, or, rather, it is mostly a technical book, but not in computer science.

This book is about the difference between a fixed mindset—a belief that how you do at most things is primarily determined by the talents and interests you're born with—and a growth mindset—a belief that how you do at most things is primarily determined by how much and how well you work at them. It provides both anecdotes and research supporting the conclusion that the fixed mindset is neither true nor useful. That is, success has a lot more to do with effort than talent, and the belief that talent is the major controlling factor is harmful.

Why would I bring this up? The fixed mindset is the dominant paradigm in our culture, broadly defined, but it's also the dominant paradigm in the computing subculture more narrowly defined. I've recently seen discussions about whether you can teach people problem solving (yes, you can, and yes, it is partly about intelligence, but you can teach that, too), about whether programmers can be good system administrators and about whether Microsoft system administrators can be good UNIX system administrators. All of these “Can people in

specialty X be good at specialty Y” questions presuppose a fixed mindset. These are nonsensical questions from a growth mindset. They also presuppose that all programmers, all Microsoft system administrators, and so on are somehow inherently shaped for their jobs; I can't decide whether this is because people think everybody gets the job they were born to, which you'd think most people would realize didn't work for them, or because they think some things warp your brain if you ever accept money for them. Either way, it seems a peculiar belief to me.

I could expound at length on these subjects. I have before and I probably will again. But it would save me a lot of trouble if people would just go out and read this book.

#### **SECURITY POWER TOOLS**

*Bryan Burns, Jennifer Grannick, Steve Manzuik, and many others*

O'Reilly Media, 2007, 570 pages.  
ISBN 13: 978-0-596-00963-2

#### **REVIEWED BY SAM STOVER**

One of the great things about doing book reviews is that once I run out of books that interest me, I start looking at books that I might otherwise overlook. Being in the security field, I'm almost embarrassed to say that I would have overlooked this book—and that would have been a shame. The point is summed up nicely in the foreword, where the big question of “Why write this book?” is discussed. Why indeed? There are plenty of other books on Metasploit, Nessus, Nmap, etc. etc. Does the world need another one? Yes, it does. Not only is this book the best “all-in-one” compilation of all the important security tools, it shows you how to use them. No, I mean really *use them*.

Before we get into the usage, I want to take a second to specifically mention Chapter 1, “Legal and Ethics Issues.” I can't quite say that this chapter is worth the price of admission alone, but it's close. It's the least technical chapter in the book, but arguably the most important, and certainly the one you need to read first. It was written by a real lawyer about real legal issues that arise in the security world. If you don't understand the potential ramifications of your actions, then you should get out of the sandbox. Finally, we get some substance instead of the usual “We are not lawyers, so don't do anything stupid” disclaimer so prevalent in security books. Kudos to the authors—this chapter has been a long time in coming.

Now that you've read just enough legalese to make you nervous, let's get into the tech stuff. The book is broken into seven sections, ranging from reconnaissance to exploitation to mitigation. There's even a dash of forensics in there too—this book really does cover all the bases. As I mentioned before, the usual suspects are discussed in great detail, as are some newer tools, particularly in the wireless reconnaissance and penetration chapters. One thing that I truly appreciated was the constant stream of references to other works. This book might touch on pretty much everything from rootkits to firewalls, but the authors acknowledge many other specialized books that pick up where this one leaves off. This is just another example of the classiness that is the hallmark of O'Reilly. In that vein, grammatical errors, typos, and cut-and-paste hacks were all absent, as usual. There might have been twelve self-acknowledged “nonwriters” authoring the chapters, but each topic reads cleanly and professionally.

One problem inherent with this type of book is the expectation and experience level of the reader. Some veterans might be a little frustrated with the amount of introductory material, but I think this was kept to a minimum. The book is accessible to the novice but tailored for the tech-savvy. Simply put, I would recommend this book to anyone who needs a great all-around reference and is concerned with getting stuff done. There are plenty of networking topics for the system admins and vice versa. Since the writers and editors did such a great job writing for a highly technical—but not necessarily security-minded—audience, anyone with a good technical background should snap this book up as soon as possible.

## **HACKING: THE ART OF EXPLOITATION, 2ND EDITION**

*Jon Erickson*

No Starch Press, 2008. 488 pages.  
ISBN 978-1-59327-144-2

### **REVIEWED BY EVAN TERAN**

This is a good book. It does a great job of first establishing the mindset of a hacker and then walking the reader step by step through the various techniques of finding interesting ways to solve problems. This in itself is what the author claims is the defining characteristic of a hacker, and I agree.

Unlike the first edition, this book spends about 100 pages explaining what programming is, first with pseudo-code giving very general examples, then working through all the major features of the C programming language, and finally how this all

ties to assembly language for the x86 processor. Although I feel this is all requisite knowledge for the subject, if you are already a competent C programmer you are likely going to want to skip chapter “0x300” altogether, especially if you are also comfortable with x86 assembly.

In addition to the general programming overview, the networking chapter has been expanded to include much more information about programming using the BSD sockets API. Once again, if you are already familiar with programming sockets, you may want to skip this chapter. However, since not every programmer writes networked applications, this was a valuable addition to the book.

All of the basics for program exploitation are explained well. Everything from the commonplace stack-based buffer overflow to how to make a format string vulnerability corrupt memory in a useful way is covered. The explanations and examples of the various techniques have been improved since the first edition and use more realistic code samples.

There is also a completely new chapter, “Countermeasures,” which goes over ways that a hacker can get around some of the basic preventives a programmer might use. These techniques, ranging from trivial to modern, include logging, byte restrictions, stack randomization, and nonexecutable stacks. Basically this chapter is a way of addressing many of the newer techniques that would prevent the book's examples from working on a real-world machine.

For example, pretty much all of the stack exploit examples earlier in the book assume that the stack is located at the same location for each process. In fact, the LiveCD has ASLR (Address Space Layout Randomization) disabled in order to make this work. One thing that I didn't care for is that this was mentioned as an afterthought. In addition to that, the solution of bouncing off the linux-gate is incomplete, since it does not work in the newest Linux kernels. The author does suggest another solution, which works OK (as in, not all the time, but once is enough).

Personally, if someone asked me to recommend a book on exploitation and they had no knowledge of programming, I would probably recommend they start with a book that focuses specifically on C first. In my opinion, program exploitation is an advanced programming topic that will do nothing but frustrate beginners. Overall, I would recommend this book to those who are competent C programmers, with the caveat that they may want to either skim or skip the early background chapters.

# USENIX 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 summaries of sessions at USENIX conferences.

**ACCESS TO ;LOGIN:** online from October 1997 to this month:  
[www.usenix.org/publications/login/](http://www.usenix.org/publications/login/).

**DISCOUNTS** on registration fees for all USENIX conferences.

**DISCOUNTS** on the purchase of proceedings and CD-ROMs from USENIX conferences.

**SPECIAL DISCOUNTS** on a variety of products, books, software, and periodicals: [www.usenix.org/membership/specialdisc.html](http://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/](http://www.usenix.org/membership/) or contact [office@usenix.org](mailto:office@usenix.org). Phone: 510-528-8649

## RESULTS OF THE ELECTION OF THE USENIX BOARD OF DIRECTORS FOR 2006-2008

Communicate directly with the USENIX Board of Directors by writing to [board@usenix.org](mailto:board@usenix.org).

### PRESIDENT

Clem Cole, *Intel*  
[clem@usenix.org](mailto:clem@usenix.org)

### VICE PRESIDENT

Margo Seltzer, *Harvard University*  
[margo@usenix.org](mailto:margo@usenix.org)

### SECRETARY

Alva Couch, *Tufts University*  
[alva@usenix.org](mailto:alva@usenix.org)

### TREASURER

Brian Noble, *University of Michigan*  
[brian@usenix.org](mailto:brian@usenix.org)

### DIRECTORS

Matt Blaze, *University of Pennsylvania*  
[matt@usenix.org](mailto:matt@usenix.org)

Gerald Carter, *Samba.org/Centeris*  
[jerry@usenix.org](mailto:jerry@usenix.org)

Rémy Evard, *Novartis*  
[remy@usenix.org](mailto:remy@usenix.org)

Niels Provos, *Google*  
[niels@usenix.org](mailto:niels@usenix.org)

For details of the election results, see <http://www.usenix.org/about/elections08results.html>.

## NOTICE OF ANNUAL MEETING

The USENIX Association's Annual Meeting with the membership and the Board of Directors will be held during the 2008 USENIX Annual Technical Conference, on June 25, 2008, Boston, Massachusetts. The time and place of the meeting will be announced on-site and on the conference Web site, [www.usenix.org/usenix08](http://www.usenix.org/usenix08).

## USENIX & SAGE CONTACT INFO

**Conference Registration:**  
[conference@usenix.org](mailto:conference@usenix.org)

**Executive Director:**  
Ellie Young, [ellie@usenix.org](mailto:ellie@usenix.org)

**;login: Submissions:**  
[login@usenix.org](mailto:login@usenix.org)

**Mailing List Rental:**  
Camille Mulligan, [sales@usenix.org](mailto:sales@usenix.org)

**Marketing:**  
Anne Dickison, [marketing@usenix.org](mailto:marketing@usenix.org)

**Membership:**  
[membership@usenix.org](mailto:membership@usenix.org)

**Ordering USENIX & SAGE Publications:**  
[orders@usenix.org](mailto:orders@usenix.org)

**Password Problems:**  
[office@usenix.org](mailto:office@usenix.org)

**Permission to Reprint from USENIX & SAGE Publications:**  
Jane-Ellen Long, [jel@usenix.org](mailto:jel@usenix.org)

**Publicity:**  
Anne Dickison, [marketing@usenix.org](mailto:marketing@usenix.org)

**Publishing with USENIX and SAGE:**  
Jane-Ellen Long, [jel@usenix.org](mailto:jel@usenix.org)

**Campus Rep Program:**  
Anne Dickison, [anne@usenix.org](mailto:anne@usenix.org)

**SAGE:**  
Jane-Ellen Long, [jel@sage.org](mailto:jel@sage.org)

**Sponsorship and Exhibiting:**  
Camille Mulligan, [sponsorship@usenix.org](mailto:sponsorship@usenix.org)

**Standards:**  
Nick Stoughton, [nick@usenix.org](mailto:nick@usenix.org)

**Student Conference Grants/Stipends:**  
Devon Shaw, [students@usenix.org](mailto:students@usenix.org)

**System Administrator:**  
Tony Del Porto, [tony@usenix.org](mailto:tony@usenix.org)

**Tutorial Speakers:**  
Dan Klein, [tutorials@usenix.org](mailto:tutorials@usenix.org)

**Web Site:**  
Casey Henderson, [casey@usenix.org](mailto:casey@usenix.org)

---

**PROFESSORS, CAMPUS STAFF, AND STUDENTS—**

**DO YOU HAVE A USENIX REPRESENTATIVE ON YOUR CAMPUS?**

**IF NOT, USENIX IS INTERESTED IN HAVING ONE!**

---

The USENIX Campus Rep Program is a network of representatives at campuses around the world who provide Association information to students, and encourage student involvement in USENIX. This is a volunteer program, for which USENIX is always looking for academics to participate. The program is designed for faculty who directly interact with students. We fund one representative from a campus at a time. In return for service as a campus representative, we offer a complimentary membership and other benefits.

A campus rep's responsibilities include:

- Maintaining a library (online and in print) of USENIX publications at your university for student use
- Distributing calls for papers and upcoming event brochures, and re-distributing informational emails from USENIX
- Encouraging students to apply for travel grants to conferences
- Providing students who wish to join USENIX with information and applications
- Helping students to submit research papers to relevant USENIX conferences
- Providing USENIX with feedback and suggestions on how the organization can better serve students

In return for being our “eyes and ears” on campus, representatives receive a complimentary membership in USENIX with all membership benefits (except voting rights), and a free conference registration once a year (after one full year of service as a campus rep).

To qualify as a campus representative, you must:

- Be full-time faculty or staff at a four year accredited university
- Have been a dues-paying member of USENIX for at least one full year in the past

For more information about our Student Programs, see <http://www.usenix.org/students>

USENIX contact: Anne Dickison, Director of Marketing, [anne@usenix.org](mailto:anne@usenix.org)

## conference reports

### THANKS TO OUR SUMMARIZERS

#### 6th USENIX Conference on File and Storage Technologies (FAST '08) .....93

Medha Bhadkamkar  
Swapnil Bhatia  
Chris Frost  
James Hendricks  
Elie Krevat  
Dutch Meyer  
Shafeeq Sinnamohideen

#### 2008 Linux Storage & Filesystem Workshop (LSF '08).....107

Grant Grundler, with help from James Bottomley, Martin Petersen, and Chris Mason

*The LSF '08 summaries were substantially abbreviated for publication. For the complete summaries, see <http://www.usenix.org/events/lfsf08/lfsf08sums.pdf>.*

### FAST '08: 6th USENIX Conference on File and Storage Technologies

San Jose, CA  
February 26–29, 2008

#### KEYNOTE ADDRESS: “IT’S LIKE A FIRE. YOU JUST HAVE TO MOVE ON”: RETHINKING PERSONAL DIGITAL ARCHIVING

Cathy Marshall, Senior Researcher, Microsoft

Summarized by Swapnil Bhatia ([sbhatia@cs.unh.edu](mailto:sbhatia@cs.unh.edu))

To a storage systems researcher, all user bytes are created opaque and equal. Whether they encode a timeless wedding photograph, a recording of a song downloaded from the WWW, or tax returns of yesteryear is not germane to the already complex problem of their storage. But according to Cathy Marshall, this is only one stripe of the storage beast. There is a bigger problem that we will have to confront eventually: The exponentially growing size of our personal digital estate will soon—if it has not done so already—surpass our management abilities.

In a visionary keynote address delivered in her own unique style, Marshall successfully argued the increasing importance, complexity, and enormity of the problem of personal digital archiving: the need for tools and methods for the collection, preservation, and long-term care of digital artifacts valuable to us.

In the first part of her talk, using data gathered from extensive surveys, interviews, and case studies, Marshall characterized the prevailing attitudes of users toward the fate of their digital data. She discovered a strange mix of reckless resignation, paranoia, and complacency based on flawed assumptions about the role of backups, the perceived permanence of the content on the WWW, and a misperception of one’s own ability to manage one’s data over a variety of time scales.

Marshall outlined four fundamental challenges in personal digital archiving: value, variation, scale, and context. According to Marshall, personal digital archiving is not just about backing up data; it is the selective preservation and care of digital artifacts that are of value to us. Quantifying the value of a digital artifact appears to be a difficult task, even for its owner. Furthermore, the variability in the value of a digital artifact over time to its user only adds complexity to the problem. Current methods of backup are value oblivious and therefore promote blind duplication rather than selective archival.

The second challenge arises from the distributed storage of our digital assets. Personal digital artifacts are almost never stored in a single central repository. Typically, they end up copied from their source to a number of personal computers and servers, some of which may not be controlled by the owner(s) of the digital artifacts. Moreover, all the many copies created in the process of the distribu-

tion may not be identical; for example, differences in resolution of images or associated metadata may result in many differing versions of essentially the same digital artifact. Without any supervening method of preserving provenance, the problem of archiving a family of related but different digital artifacts dispersed across many locations—building a digital Noah's ark of sorts—quickly becomes intractable.

Archiving would still be a manageable problem, were it not for its sheer enormity. According to Marshall, there are about seven billion pictures on the Yahoo! and Facebook sites. Most users are simply incapable of dealing with large numbers of digital artifacts because of a lack of either technological savvy or the time and effort needed. Archiving personal data requires stewardship, but no tools currently exist to facilitate it at this scale.

Finally, even the perfect archiving tool augmented with the best search interface would be of no help if, years later, one has forgotten the content and the context of one's archive. Such forgetfulness is—as Marshall found through her user interviews—an often underestimated problem. Marshall suggested that re-encountering archived data is a promising solution to this problem. The idea behind re-encountering is to enable archived data to remind the user of their own provenance by facilitating periodic review or revisitation.

Marshall concluded by saying that solving these challenges would require a method for assessing the value of digital artifacts and better curatorial tools and services with built-in facilities for re-encountering. Marshall also pointed out that this problem will require cooperation and partnership among social Web sites, software companies, data repositories, ISPs, and content publishers.

In response to a question from an audience member, Marshall mentioned that, as yet, users were not willing to pay for an archiving service. Another questioner asked Marshall to explain why it was not enough to back up all user data. The same misequation of backup with archiving also arose in audience discussions after the talk. Marshall explained that users were not looking to save everything: This would only make the problem of context and re-encounter harder. Rather, what users need is a selective way of preserving personally valuable data, along with the context that makes the data valuable, and doing so over a time scale that is significantly longer than that of a backup.

## **DISTRIBUTED STORAGE**

*Summarized by Chris Frost (frost@cs.ucla.edu)*

### ■ **Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage**

*Mark W. Storer, Kevin M. Greenan, and Ethan L. Miller, University of California, Santa Cruz; Kaladhar Voruganti, Network Appliance*

Mark Storer spoke on the Pergamum system, which uses disks, instead of tape, for archival storage, and their work

toward reducing power costs. With Pergamum, Storer et al. wanted to achieve power cost and data space scalability similar to tape systems but achieve random access performance similar to disk array and Massive Array of Idle Disk systems. Their approach uses an evolvable distributed network of disk-based devices, called tomes. Each tome can function independently and is low-power enough to run on Power over Ethernet. Pergamum uses intradisk and interdisk reliability to protect against corruption and tome failure, including trees of data algebraic signatures to efficiently detect and locate corruption. A tome spins down its disk when inactive and stores metadata in nonvolatile RAM (NVRAM) to help keep its disk spun down even longer.

Storer et al.'s experiments show that adding one to three backup tomes per tome increases the mean time to failure by orders of magnitude. A tome's low-power CPU and SATA disk sustain a 5 MB/s write speed; they anticipate raising this with further CPU optimizations. One thousand tomes with a spin rate of 5% could ingest 175 MB/s. They cite costs as being 10%–50% of today's systems.

David Rosenthal of Stanford asked how well they understood drive failures, especially given a tome's difference from the expected environment. Storer noted that accelerated drive testing is a good idea and also that this is one reason they use interdisk redundancy and data scrubbing. Geof Kuenning of Harvey Mudd asked what would happen if a tome's NVRAM were to fail. Storer replied that they would replace the device and use interdisk reliability to rebuild. Another person asked about how much energy routers use, since disk arrays do not have this component. Storer answered that router and disk array backplane energy costs are similar.

### ■ **Scalable Performance of the Panasas Parallel File System**

*Brent Welch, Marc Unangst, and Zainul Abbasi, Panasas, Inc.; Garth Gibson, Panasas, Inc., and Carnegie Mellon University; Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou, Panasas, Inc.*

Brent Welch presented Panasas's parallel file system, which has installations as large as 412 TB with sustained throughput of 24 GB/s using thousands of Panasas servers. The Panasas system is composed of storage and manager nodes; a storage node implements an object store and a manager node runs a metadata service that layers a distributed file system (PanFS, NFS, or CIFS) over the object store. Scalability is the goal of Panasas and is primarily achieved by distributing metadata and supporting scalable rebuilds.

Each file and its metadata are stored together, in an Object Storage Device File System. The system automatically selects redundancy (RAID) levels on a per-file basis based on file size, optimizing space usage for small files and run-time performance for large files. By randomly placing data among storage nodes, failure recovery is made scalable, an essential feature for such a large, distributed system. Panasas servers also include integrated batteries to permit them to treat



RAM as nonvolatile RAM (NVRAM), significantly increasing metadata manipulation performance. Welch et al. find that system write performance scales linearly with the number of servers; read performance scales fairly well, although effective readahead becomes difficult at large scales.

One audience member asked how many objects backed a typical file. Welch said, “Several; a new 512-kB file would consist of about ten objects.” Another person asked how much of Panasas’s scalability is due to using an object store. Welch replied that the object store primarily simplified the software. Another person, worried about the lack of true NVRAM, asked what would happen if a hardware fault lost the contents of RAM. Welch said an exploded storage server would not harm the file system, because data and logs are replicated synchronously.

■ **TierStore: A Distributed Filesystem for Challenged Networks in Developing Regions**

*Michael Demmer, Bowei Du, and Eric Brewer, University of California, Berkeley*

Michael Demmer spoke on sharing data among computers in developing regions, where network connectivity is intermittent and of low bandwidth. Many approaches exist for working in such environments, but all start from scratch and use ad hoc solutions specific to their environment–application pair. Demmer et al. aim to extend the benefits of Delay Tolerant Networking to provide replicated storage for applications to easily use for storage and communication.

TierStore’s design has three goals. (1) Software should be able to easily use TierStore. They achieve this goal by exposing TierStore as a file system. This permits many existing programs to use TierStore and takes advantage of the well-known, simple, programming language-agnostic, and relatively weakly consistent filesystem interface. (2) TierStore should provide offline availability. Therefore TierStore provides locally consistent views with a single-file coherence model and uses application-specific conflict resolvers to merge split views after a write–write conflict. (3) TierStore should distribute data efficiently. Delay-tolerant publish–subscribe is used to provide transport portability. A simple publish–subscribe protocol is used to manage interest on fine-grained publications (e.g., a user’s mailbox or an RSS feed).

One audience member asked how TierStore deals with churn. Demmer replied that their current prototype is manually configured and has a fairly stable overall topology. Another person asked why Demmer et al. propose a fancy solution when they also say one cannot use fancy technology in such environments. Demmer answered that economics is often the barrier. When asked about the usability of conflict resolution in their system, Demmer said that, thus far, applications have been designed to be conflict-free (e.g., Maildirs). They hope applications will continue to be able to store data so that conflicts are not a problem.

**YOU CACHE, I CACHE . . .**

*Summarized by Swapnil Bhatia (sbhatia@cs.unh.edu)*

■ **On Multi-level Exclusive Caching: Offline Optimality and Why Promotions Are Better Than Demotions**

*Binny S. Gill, IBM Almaden Research Center*

Binny S. Gill presented the two primary contributions of his work on a multi-level cache hierarchy: a new PROMOTE operation for achieving exclusivity efficiently, and policies that bound the optimal offline performance of the cache hierarchy.

Gill started his talk with a discussion of the DEMOTE technique used for achieving exclusivity. When a higher-level cache evicts a page, it DEMOTES it to the lower cache, which in turn makes room for the new page, possibly demoting a page in the process itself. Gill argued that DEMOTE is an expensive operation and performs poorly when bandwidth is limited. Furthermore, for workloads without temporal locality, DEMOTES are never useful.

Gill proposed a new technique for achieving exclusivity based on a new operation called PROMOTE. PROMOTE achieves exclusivity by including an ownership bit with each page traversing the hierarchy, indicating whether ownership of the page has been claimed by some lower-level cache. Each cache on a path PROMOTES a page with a certain probability, which is adapted so as to equalize the cache life along a path. (A higher-level cache periodically sends its cache life to the next-lower-level cache.) Experimental results show that PROMOTE is better than DEMOTE when comparing average response time, and it cuts the response time roughly in half when intercache bandwidth is limited.

Gill also presented two policies, OPT-UB and OPT-LB, which bound the performance of an optimal offline policy for a multi-level cache hierarchy. Essentially, both policies use Belady’s optimal offline policy for a single cache and apply it incrementally to a path. In his paper, Gill proves that no other policy can have a better hit rate, intercache traffic, and average response time than OPT-UB. Gill presented experimental results that showed that the two bounds were close to each other.

One audience member pointed out that using PROMOTE would require a change in the command set of the protocol used for intercache communication. Gill argued that the performance gained by using PROMOTE would hopefully incentivize such a change. In response to another question, Gill clarified that the response times of the caches need only be monotonically increasing, with no constraint on the magnitude of the differences. Another questioner provided a counter-example scenario in which the working set of the workload is highly dynamic and asked what impact this would have on the adaptive PROMOTE probabilities. Gill pointed out that when cache lives are equalized, the behavior of the PROMOTE scheme would be no different—in terms of hits—than that with DEMOTE.

### ■ **AWOL: An Adaptive Write Optimizations Layer**

Alexandros Batsakis and Randal Burns, Johns Hopkins University; Arkady Kanevsky, James Lentini, and Thomas Talpey, Network Appliance, Inc.

Although an application writes data to the disk, in reality the file system caches writes in memory for destaging later. Alexandros Batsakis's talk tried to answer the following questions about such write-behind policies: How much and which dirty data should be written back? And when should this be done? The answers require careful consideration of the tradeoffs between writing too quickly or waiting too long, and using available memory for caching writes versus reads. To this end, Batsakis proposed a three-part solution: adaptive write-back, ghost-caching, and opportunistic queuing.

Batsakis explained an adaptive high–low watermark algorithm in which write-back commences when the “dirtying” rate crosses the high mark and stops when it falls below the low mark. The two watermarks are dynamically adapted in harmony with the dirtying and flushing rates.

Batsakis proposed ghost-caching to balance memory usage across reads and writes. The scheme involves the use of two ghost caches. The Ghost Miss Cache (GMC) records meta-data of evictions resulting from write buffering. A cache miss with a GMC hit is used to deduce that write buffering is reducing the cache hit ratio. The Ghost Hit Cache (GHC) records a subset of the cached pages. A read hit that falls outside the GHC is used to deduce that additional write buffering will lower the read hit rate. Thus, the GHC is used to prevent interference from writes early on, rather than recover from it using the GMC later.

Batsakis proposed the use of opportunistic queuing to decide which data to write back. An I/O scheduler maintains separate queues for blocking (read) and nonblocking (writes) requests with requests sorted by block number to minimize seek time. In Batsakis's scheme, dirty blocks are added to a third (nonblocking) opportunistic queue. When a page is flushed from any of the other queues, the scheduler is free to service a “nearby” page from the opportunistic queue. Overall, experiments show that the three optimizations can improve performance by 30% for mixed workloads.

### ■ **TaP: Table-based Prefetching for Storage Caches**

Mingju Li, Elizabeth Varki, and Swapnil Bhatia, University of New Hampshire; Arif Merchant, Hewlett-Packard Labs

Mingju Li presented the two primary contributions of her work: a method for detecting sequential access patterns in storage-level workloads and a method for resizing the storage-level prefetch cache optimally.

Table-based prefetching (TaP) is a sequential detection and cache resizing scheme that uses a separate table for recording workload history and resizing the prefetch cache optimally. TaP records the address of a cache miss in a table. If

a contiguous request arrives later, then TaP concludes that a sequential access pattern exists in the workload, and it begins prefetching blocks on every subsequent cache hit from that stream. Separating the workload history needed for detection from other prefetched data prevents cache pollution and allows TaP to remember a longer history. As a result, TaP can detect sequential patterns that would have otherwise been lost by interleaving.

When the prefetch cache is full, TaP evicts a cache entry, but it records its address in the table. If a request for this recorded address arrives later, then TaP concludes this to be a symptom of cache shortage and expands the cache. Thus, TaP uses the table to resize the cache to a value that is both necessary and sufficient and hence optimal. As a result, TaP exhibits a higher useful prefetch ratio, i.e., the fraction of prefetches resulting in a hit. In many cases, TaP can achieve a given hit ratio using a cache that is an order of magnitude smaller than competing schemes.

In response to audience questions, Li mentioned that she planned to address the design of a prefetching module, which is responsible for deciding the size and time of prefetching, in her future work. Another question called attention to the cost of prefetch cache resizing: What is the impact of the frequent change in the size of the cache on performance? Li responded by saying that the rate at which the cache size is decreased can be controlled and set to a reasonable value. The final questioner asked Li how TaP would compare in performance to AMP (Gill and Bathen, FAST '07). Li pointed out that AMP could in fact be incorporated into TaP as one possible prefetching module and that this would be addressed in her future work.

## **WORK-IN-PROGRESS REPORTS (WIPs)**

Summarized by Shafeeq Sinnamohideen ([shafeeq@cs.cmu.edu](mailto:shafeeq@cs.cmu.edu))

### ■ **Byzantine Fault-Tolerant Erasure-Coded Storage**

James Hendricks and Gregory R. Ganger, Carnegie Mellon University; Michael K. Reiter, University of North Carolina at Chapel Hill

Hendricks presented a scheme that provides Byzantine fault tolerance for a slight overhead over non-Byzantine-fault-tolerant erasure-coded storage. Traditionally, storage systems have used ad hoc approaches to deciding which faults to tolerate and how to tolerate them. As storage systems get more complex, the kinds of faults that can occur get harder to predict; thus, tolerating arbitrary faults will be useful. Providing Byzantine fault tolerance in an erasure-coded system requires each storage server to be able to validate that the data fragment stored on that server is consistent with the data stored on the other servers. This is difficult because no server has a complete copy of the data block. Using the recent technique of homomorphic fingerprinting, however, each server can validate its fragment against a fingerprint of the complete data block, and a client can validate that all the fragments it received from the servers are consistent

with a single fingerprint. In a prototype system, Hendricks's scheme provides write throughput almost as good as a crash-only tolerant system, and with far better performance than existing Byzantine fault-tolerant schemes.

#### ■ **Mirror File System**

*John Wong, Twin Peaks Software*

Wong presented an overview of Twin Peaks's filesystem replication product. Since files stored on a local file system are vulnerable to failures of the local machine, and files stored on remote servers are vulnerable to failures of that server or the network, the Mirror File System (MFS) attempts to get the best of both worlds by mirroring the state of a local EXT3 file system onto a remote NFS server. It does so by transparently intercepting file system operations at the VFS layer and applying them to both the local and remote file systems, while requiring no modifications to applications, EXT3, or NFS.

#### ■ **Quantifying Temporal and Spatial Localities**

*Cory Fox, Florida State University*

Fox states that accurately describing workloads is critical to comparing different real workloads and creating accurate synthetic workloads. Because locality is an important property of a workload, understanding how the locality of a workload is transformed as it passes through system components, such as caches, is crucial. Fox suggests that current metrics such as cache hit ratios, reference distance, and block distance do not adequately describe locality. Instead he proposes a metric called "affinity," which builds on block and reference distances, while being less sensitive to hardware details. Much future work is anticipated in showing how well it captures locality and in studying how the workloads seen by individual components relate to the overall workload.

#### ■ **Filesystems Should Be Like Burger Meals: Supersize Your Allocation Units!**

*Konstantin Koll, University of Dortmund, Germany*

Koll discussed the tradeoff between small and large filesystem allocation units, through a humorous analogy to fast food meal sizes. In both file systems and restaurants, excessively small allocation units reduce waste, but at the same time they reduce performance and add administrative overhead. Koll stated that fast food vendors have realized that since food is cheap, avoiding waste is less important than reducing overheads, and since a study of filesystem workloads reveals that the amount of wasted space grows less than linearly with allocation unit size, filesystem designers should use larger allocation units. An unnamed questioner stated that XFS and other extent-based filesystems do use large allocation units. Koll responded: Then I hope you found this talk amusing.

#### ■ **Zumastor: Enterprise NAS for Linux**

*Daniel Phillips*

Zumastor is a NAS product, built on Linux, that provides live volume snapshots, remote volume replication, online volume backup, NFS, Samba, and CIFS interfaces, and easy administration. It is based on the ddsnap engine, which is a mostly userspace driver presenting a block device interface with copy-before-write snapshots. The snapshots can be replicated to other Zumastor servers using techniques such as compression or binary differencing to reduce the amount of data to be transmitted. Future work includes adding a graphical administrative console, better volume management, online resizing, and incremental backup.

#### ■ **View-based Collective I/O for MPI-IO**

*Javier Garcia Blas, Florin Isaila, and Jesus Carratero, University Carlos III of Madrid*

Blas proposed an alternative to two-phase collective I/O with the goal of increasing performance by reducing the cost of data scatter-gather operations, minimizing the overhead of metadata transfer, and reducing the amount of synchronous communication. The alternative, called view-based collective I/O, relies on clients once providing the aggregators with additional information about the type of the data the client will be accessing. This reduces the amount of information the client must send with every access, as well as permitting the aggregators to cache file data across operations, allowing one operation to benefit from a previous operation's cache miss. In the MPI-IO benchmark performed, view-based I/O reduced write times by more than a factor of 2 and read times by at least 10%.

#### ■ **Towards a Performance Model for Virtualised Multi-Tier Storage Systems**

*Nicholas Dingle, Peter Harrison, William Knottenbelt, Abigail Lebrecht, and Soraya Zertal, Imperial College London, UK*

Lebrecht's goal is to model the performance of a complex system using nested queuing network models of its basic components. Starting with models of simple disk and RAID system, the models she has developed closely match the performance of the real devices for random workloads. Future work includes extending these results to more complex and heterogeneous workloads.

#### ■ **Adapting RAID Methods for Use in Object Storage Systems**

*David Bigelow, Scott A. Brandt, Carlos Maltzahn, and Sage Weil, University of California, Santa Cruz*

Bigelow proposes characterizing the tradeoffs among various methods of implementing RAID in object-based storage systems. These include "client-based RAID," in which the client performs parity calculations, "RAID across objects," in which the storage system stores an object on one node but includes a different object on each node in the parity calculation, and "RAID within objects," in which the storage system stores a portion of each object on each node and computes parity across all of them. Bigelow is currently working on implementing these schemes in the Ceph Object

Storage System and evaluating their relative performance, as well as developing more complex and hierarchical schemes.

#### ■ **How Shareable Are Home Directories?**

*Carlos Maltzahn, University of California, Santa Cruz*

Maltzahn hypothesizes that most users manage a subset of their files in a way identical to other users. Thus, it should be possible to share the work of managing files across users. He proposes to quantify this shareability by categorizing files through a user survey. His survey, in which users categorized files as “unshareable,” “shareable within one user,” “shareable within one group of users,” and “publicly shareable,” revealed that 75% of users have at least half their files shareable in some way, and 50% of users have at least half their files in common with a different user.

#### ■ **Load Balancing in Ceph: Load Balancing with Pseudorandom Placement**

*Esteban Molina-Estolano, Carlos Maltzahn, and Scott Brandt, University of California, Santa Cruz*

Molina described several issues that could be encountered in Ceph owing to its use of pseudo-random placement of objects, along with potential solutions to these issues. In the case of one node that happens to hold the primary replica of many popular objects, that node can be switched to be a secondary replica of some of them, moving the load to the new primary replica. In the case of a read flash crowd, some of the readers can be directed to other nodes that hold a secondary replica of the object, or even to other clients that have a recently cached copy of the object. In the case of a write flash crowd, some clients can be directed to write to the secondary replicas, but this relies on HPC I/O extensions that allow the application to describe ordering and dependencies. Preliminary results show that these techniques allow load to be shifted away from an overloaded storage node.

#### ■ **Ringer: A Global-Scale Lightweight P2P File Service**

*Ian Pye, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz*

Pye presented a global file service that provides filesystem semantics as well as indexing based on document contents. Filesystem semantics are necessary for application compatibility, and indexing is necessary to help users find the files they are interested in. The architecture Pye proposes is a Hybrid P2P approach in which metadata servers maintain the filesystem indices and perform searches, but file data is transferred directly from the peer that has it. Future work includes implementing, testing, and evaluating the system.

#### ■ **The New and Improved FileBench**

*Eric Kustarz, Spencer Shepler, and Andrew Wilson, Sun Microsystems*

Spencer described the current state of the FileBench framework. It provides a collection of configurable workloads and can apply them to a number of storage server types. It recently underwent a large code cleanup and is distributed

in OpenSolaris and through SourceForge.net. Features in development include support for random workloads, NFS, CIFS, and multiple clients.

#### ■ **HyFS: A Highly Available Distributed File System**

*Jianqiang Luo, Mochan Shrestha, and Lihao Xu, Wayne State University*

Luo proposes a Linux filesystem that uses erasure coding to provide redundancy against hardware failure. HyFS is implemented at the user level by using FUSE and erasure codes file data across a user-configured number of NFS servers. Performance and scalability evaluations are ongoing.

#### ■ **Virtualizing Disk Performance with Fahrrad**

*Anna Povzner, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz; Richard Golding and Theodore M. Wong, IBM Almaden Research Center*

Povzner extends existing work in providing soft performance isolation to provide hard isolation guarantees. The Fahrrad disk scheduler allows clients to reserve disk time and attempts to minimize the seeks required to serve the client workloads. If seeks between streams are necessary, they are accounted to the streams that required them and thus the necessary overhead time can be reserved, allowing for hard isolation. Experiments show that this can provide complete isolation of competing workloads, with only a 2% overhead.

#### ■ **RADoN: QoS in Storage Networks**

*Tim Kaldeway and Andrew Shewmaker, University of California, Santa Cruz; Richard Golding and Theodore M. Wong, IBM Almaden Research Center*

Kaldeway presented RADoN, which aims to coordinate the individual network, cache, and storage QoS parameters in order to provide end-to-end QoS guarantees for a given application. Doing so requires discovering how the parameters of individual system components affect the overall QoS. This project seeks to discover the important parameters through modeling and simulating the system and coordination strategies and will build a framework for applications to specify their QoS requirements.

#### ■ **Improving Efficiency and Enhancing Concurrency of Untrusted Storage**

*Christian Cachin, IBM Zürich; Idit Keidar and Alexander Shraer, Technion: Israel Institute of Technology*

Cachin summarized recent improvements in protecting against storage servers that present different views of history to different clients. Fork linearizability is a useful building block because it ensures that once a server has forked a view, it must remain forked. The authors reduce the communication cost of a fork-linearizable protocol to  $nO(n)$  messages instead of  $O(n^2)$  and show that such a protocol can never be wait-free. Instead they introduce a weak fork-linearizable protocol that is wait-free and has the same communication cost.

## ■ *Reliability Markov Models Are Becoming Unreliable*

*Kevin M. Greenan and Jay J. Wylie, Hewlett-Packard*

Greenan described the Markov models traditionally used for reliability analysis. Whereas the single disk and RAID-5 models accurately model the reliability of such systems, a naive RAID-6 model underestimates the MTTF of a RAID-6 system by a factor of 2. This is because the memorylessness of the model ignores data that may have been rebuilt onto other disks after a disk failure. Although not accurate, Markov models may provide the correct intuition in reasoning about failures, and future work is necessary to develop them further or devise new models.

---

### **KEYNOTE ADDRESS: SUSTAINABLE INFORMATION TECHNOLOGY ECOSYSTEM**

---

*Chandrakant D. Patel, HP Fellow, Hewlett-Packard Labs*

*Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)*

In a far-looking presentation, Chandrakant Patel explored the hidden costs of disks and the data center, along with the potential to deliver more efficient and reliable services. Patel suggested that in the future, bringing IT to all levels of developing economies will require a sustainable ecosystem. This is based on the conviction that the technologies with the least footprint, lowest power consumption, and least materials will eventually have the lowest total cost of ownership. To realize this transformative change, we must recharacterize the costs associated with IT, drawing on deep technical knowledge of the physical processes involved.

The costs of IT can be measured at several levels. Initially, materials are extracted and formed into a usable product; next, the product is used in operation for some time; finally, the product is recycled, allowing some resources to be reclaimed while others are irreversibly lost. Other considerations such as transportation and human effort can also be incorporated. A quantifiable metric for these costs is “exergy,” which is drawn from thermodynamics as the measure of the available and usable energy in a system. In the illustrative case of a cellular phone, the CPU draws power from the battery in order to perform its function. In the process, exergy consumed by the CPU is converted to heat, which is dispelled passively in the body of the phone. In the future, owing to high power density and the difficulty of removing heat passively from stacked chip packages, even heat removal will require powered solutions. To mitigate the added power requirements, Patel suggested an active-passive solution—a phone filled with a phase-change material such as wax, which absorbs heat for short conversations and switches to active solid state heat removal. Such active-passive cooling solutions will be necessary to provision power based on need.

Using some of these principles, Patel and his associates are developing data centers that are significantly more energy-efficient. The key observation is that the common practice

of over-provisioning results in unnecessary redundancy and cooling. By emphasizing the pervasive use of sensors, one can develop a flexible and configurable approach driven by policies. Applying this technique to the cooling system of a data center has resulted in a 35% energy savings. Patel argued that the system also results in improved reliability, because it can quickly adapt to problems as they occur. The cooling is provided on-demand and over-provisioning, although it remains important, can be limited to a cost-efficient and pragmatic level.

In closing, Patel elaborated on some future directions for the work. He stressed the need to analyze the costs associated with the lifetime use of IT and showed how software tools can help with this analysis. He suggested combining the data from sensors with thermo-mechanical attributes of compute and storage components, to detect anomalies and predict failure. Stressing that the currency of the flat world will be joules of exergy consumed, Patel emphasized the importance of collaboration among computer scientists, mechanical engineers, and electrical engineers.

Eric Brewer, of Intel Research and U.C. Berkeley, asked if the joule is an accurate measure of total cost of ownership. He suggested that market inefficiencies are prevalent and seem to be growing. Patel acknowledged this discrepancy but postulated that in the long run sustainability concerns will come to dominate the costs. He also pointed to recent successes in the data center, where a sustainability-oriented approach has allowed him to quickly reduce costs. Mochan Shrestha noted that limiting provisioning could result in an increased error rate and wondered whether it was necessary to incorporate the value of the data into the model. Patel said that in practice this valuation is problematic but that it can be approximated with service-level agreements.

---

### **FAILURES AND LOSS**

---

*Summarized by Medha Bhadkamkar  
(medha@cs.fiu.edu)*

#### ■ *The RAID-6 Liberation Codes*

*James S. Plank, University of Tennessee*

Plank offered an alternate RAID-6 encoding scheme that has near-optimal performance for encoding, decoding, and modifications. Plank first described the motivation, which is based on the drawbacks of the current implementation of RAID-6 systems: They are typically slow and modifications are suboptimal and inflexible. The proposed code, termed Liberation Codes, uses parity arrays, which are  $w \times w$  bit matrices, where  $w$  is a prime number equal to or greater than the number of devices. The performance is compared with the Reed-Solomon coding. The evaluations show that the encoding is primarily focused on parity-based protection and single errors in RAID systems. Modification performance is overoptimal, but decoding performance is 15% of optimal. To further optimize decoding operations, a Bit Matrix scheduler for the XOR operations is proposed

to reduce the Liberation decoding overhead by a factor of between 6 and 11, depending on the values of  $w$ . Optimal values have also been achieved for the nonprime values of  $w = \{2,4\}$ . The paper also provides a URL to the freely available source of the Liberation Coding Library.

Nitin Garg of Data Domain posited that other matrices can have bad cache performance and wondered whether Plank had compared his method with any other methods, such as the Reed-Solomon error correcting code. Is it true that evaluating cache performance is important? Plank replied that they haven't explored caching with Reed-Solomon codes. To a question about the optimal value of  $k$  for an 8- to 10-GB disk, Plank responded that roughly a factor of 2 for encoding, but up to 4 for decoding, was optimal.

■ ***Are Disks the Dominant Contributor for Storage Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics***

*Weihang Jiang, Chongfeng Hu, and Yuanyuan Zhou, University of Illinois at Urbana-Champaign; Arkady Kanevsky, Network Appliance, Inc.*

Jiang began by stating the importance of reliability and availability in storage systems. As storage systems have evolved from single hard disks to network storage systems, it is necessary to have a good understanding of the failure characteristics of disk drives. The data used in this study was obtained by analyzing failure logs for about 44 months from 39,000 commercial storage systems and about 1.8 million disks. The data was analyzed in three dimensions, with four failure types being classified based on their root cause and symptoms, the effect of design factors such as disk models and enclosures, and statistical properties. The results show that, first, whereas disk failures (29%) form a substantial part of storage system failures, failures of other components also make a substantial contribution (7% protocol failures and 60% interconnect failures). Second, after a failure, the probability of another failure of the same type is higher. Third, interconnect redundancy is an important factor. Finally, shelf enclosures play an important role in failure patterns. This study does not take into account the impact of workloads, the reason behind failures, or the consequences of different failure types.

Someone from Wayne State University asked how disk failures are defined: by data loss or by service loss? Jiang answered that problems such as scratches, vibrations, or malfunctioning of internal components are responsible for disk failures. Data loss is a consequence of disk failures.

■ ***Parity Lost and Parity Regained***

*Andrew Krioukov and Lakshmi N. Bairavasundaram, University of Wisconsin, Madison; Garth R. Goodson, Kiran Srinivasan, and Randy Thelen, Network Appliance, Inc.; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison*

Andrew Krioukov explained that RAID systems offer numerous data protection techniques and it is unclear which

technique or combination of techniques can protect against which kind of errors. The focus is primarily on parity-based protection and single errors in RAID systems. To solve this problem, a formal method based on model checking is used to analyze the design of the protection techniques. The model checker outputs a state machine that shows the state transitions that are obtained and searches the space for all possible states. It provides primitives for disk operations, such as atomic reads and writes, and for data protection, such as checksums and parity. For every analysis, exactly one error is injected. The model checker is also used to generate data loss or corruption probabilities. The results show that, for all designs, single errors can cause data loss. In addition, data scrubbing, which is used to reduce double disk failure, actually spreads corrupt data in one block to other blocks because of parity calculations. Also, data loss has a higher probability than data corruption. To address the issues uncovered, the authors also propose a protection mechanism, which uses version mirroring and combines block checksums, identity information, parity, and scrubbing.

Someone asked how one would handle a case where a mis-directed write overwrites a parity block. Krioukov said that since parity blocks are protected by checksums, by a comparison of blocks on the data disk and the parity disk we know whether data has been lost, and it can be restored by reconstruction. Erik Reidel of Seagate asked about the complexity involved in representing the operations in a model. Krioukov said that building the model checker was simple, but building the framework with primitives was difficult. John Carrier of Cray, Inc., asked whether the model checker can be extended to RAID 6, especially since RAID 5 will be phased out soon. Krioukov answered that it can be used with double parity and can definitely be extended.

**CPUS, COMPILERS, AND PACKETS, OH MY!**

*Summarized by Elie Krevat (ekrevat@cs.cmu.edu)*

■ ***Enhancing Storage System Availability on Multi-Core Architectures with Recovery-Conscious Scheduling***

*Sangeetha Seshadri, Georgia Institute of Technology; Lawrence Chiu, Cornell Constantinescu, Subashini Balachandran, and Clem Dickey, IBM Almaden Research Center; Ling Liu, Georgia Institute of Technology; Paul Muench, IBM Almaden Research Center*

As legacy storage systems transition toward multi-core architectures and embedded storage software systems (controllers) are becoming more complex and difficult to test, Sangeetha Seshadri and her co-authors argue that it is important not to focus just on performance but also on system availability. Storage controllers have many interacting components and exhibit many different types of transient failures (these types are classified in the paper). A transient failure that occurs in one thread is typically handled by restarting and reinitializing the entire system, which also requires consistency checks. As systems grow to

larger numbers of cores, systemwide recovery may not scale. However, task-level recovery has the potential for recovering a smaller subset of components in the system, with the additional challenges of determining the correct recovery semantics (dynamic and stateful), identifying recovery dependencies, and bounding the recovery process in time and resource consumption.

To improve recovery time and better scale the recovery process with system growth, the authors propose a framework for more fine-grained task-level recovery. The design goals of these recovery-conscious scheduling algorithms include creating a nonintrusive recovery framework, dynamically determining recovery actions, tracking recovery dependencies, and generally improving system availability by reducing the ripple effect of failures. Developers specify clean-up blocks (task-specific recovery procedures) and explicit dependencies between tasks, which are then refined by the system into recovery groups that include implicit dependencies. To limit the number of dependent tasks dispatched concurrently, resource pools partition the processors into smaller independent units. A recovery-conscious scheduler maps recovery groups to resource pools in a static or dynamic fashion while adhering to recoverability constraints. The scheduler bounds the time of recovery and reduces the impact of a failure in a proactive manner by limiting the number of outstanding tasks per recovery group, and in a reactive manner after a failure occurs by waiting to dispatch new tasks for a group currently undergoing recovery until after the recovery completes. The authors implement their recovery-conscious scheduler (RCS) on real industry-strength storage controllers and compare it with a standard performance-oriented scheduler (POS) that does not consider recovery dependencies. They measure the good-path performance during normal operation, and the bad-path performance under localized failure and recovery, using the z/OS Cache-Standard workload for which they identify 16 recovery groups. Experiments show that Dynamic RCS closely matches the good-path throughput of POS while improving bad-path throughput by 16.3%.

One of the participants asked how easy it is to define recovery groups, and if the developer gets it wrong, how that would affect performance. Sangeetha answered that a developer defines recovery groups explicitly based on whether a task accesses the same resource, which depends on the system, complexity of code, and other interactions. She noted that task-level recovery is an option, but system-level recovery is still needed as a safety net. Another participant asked what properties of tasks make them easier to identify than components as a reasonable boundary for recovery. The response was that techniques such as micro-reboots will reset the system, but tasks handle resources across different components. In some situations the controller can retry the operation at the task level and succeed, and based on task functionality these situations can be identified. The last question addressed how the system would scale for

pool sizes greater than a single processor. The response was that the experimental setup had eight processors, a larger pool size is possible, and it's just an issue of defining the right granularity. For coarser constraints between groups, it would make sense to have a larger pool size.

#### ■ **Improving I/O Performance of Applications through Compiler-Directed Code Restructuring**

*Mahmut Kandemir and Seung Woo Son, Pennsylvania State University; Mustafa Karakoy, Imperial College*

Large-scale applications in science and engineering have grown dramatically in complexity, requiring huge computational needs and generating large amounts of data. According to Seung Woo Son and his co-authors, I/O is always a pressing problem for these data-intensive applications, but disk performance has not kept pace with the large annual growth in storage capacity density and processor speeds. One promising way to handle this problem and improve I/O performance is to reduce the number of disk accesses, achieved at different layers of the I/O subsystem by caching or restructuring the application code to maximize data reuse. Since the compiler has better knowledge of the entire application code, including data access patterns, the authors address the growing I/O problem through compiler-directed code restructuring, which can be used along with other OS- and hardware-based schemes.

The authors propose an approach to increase disk reuse in the compiler, which will hopefully also reduce the number of disk accesses by improving the chances of finding data in the cache. Their approach optimizes the entire program code rather than individual loop-nests, and they discussed file layout optimizations for adapting to parallel execution. The targeted disk system architecture is one in which file striping occurs over parallel disks, where a compiler should know which disks are accessed by certain portions of an array, either because this information is already supplied to the compiler or because it is available from an API. Inter-iteration data dependencies may not allow for code restructuring for better disk reuse, but if an ordering is legal for the particular data dependencies, then the authors make use of polyhedral algebra based on Presburger arithmetic to capture and enumerate those legal loop iterations that exhibit disk access locality. A disk map is defined to capture a particular set of disks, and a disk locality set is a set of loop iterations that access the same set of disks. Then the authors use two procedures to maximize disk reuse. First, for a given disk array, iterations in the disk locality set are executed consecutively. Second, when moving from one disk locality set to another, a disk locality set that accesses a new disk map is selected to have minimum Hamming distance from the current disk map. This second condition minimizes the number of disks whose status is changed when executing the iterations in a new disk locality set. Since real applications have other data dependencies, the authors also demonstrate how to use existing heuristics to merge or split nodes in a locality set graph (LSG) that captures these

dependencies, thereby converting a nonschedulable LSG to a schedulable one. The authors also show how file layout modifications (striping info) can be changed, using profiling to detect the most suitable file layout for each file and then transforming the layout during optimization. Because good disk reuse for a single CPU does not imply that good disk reuse happens overall, the scheduling algorithm determines the global (inter-thread) usage of disks and selects disk locality sets based on a global estimate.

To evaluate these scheduling algorithms, a compiler was implemented using SUIF, and different algorithms were tested on a number of applications. The whole program-based disk reuse optimization (DRO-WP) algorithm achieved on average 23.9% better performance in I/O time than the base algorithm and 15% better performance than using conventional data locality optimization (CLO) techniques. This performance improvement occurs because the average number of times a given data block is visited is much lower with DRO-WP than with CLO (2.1 compared to 3.9). Other results show that performance gains are sensitive to the size of the cache but still substantial with higher cache sizes, and parallel thread optimization is important in maximizing overall disk reuse, especially with a large number of CPUs.

One of the participants tried to understand the limitations of this approach by asking whether there have been other optimizations besides using the polyhedral approach that were considered but did not map well. Seung answered that they use conventional solutions from other domains, which looks reasonable for now, but there is still room for more optimizations. Another participant asked whether the approach to file layout optimization, when looking over all the access patterns to find a better layout, used estimated weights for different bits of code accessing the same block or assumed the same rate of access. The response was that a ranking system is used to determine the best candidate, rating each optimization and selecting the highest value, but weights for the rate of access in different code regions were not used. Another participant, noting that in large-scale systems things change constantly, wanted to know if he would need to recompile for a changing environment to better optimize I/O. The answer to this question was that reoptimization is suggested in changing environments. To the question of how this work of limiting data reads performs in areas of HPC that don't do many data reads (e.g., internal file systems may not even cache data), the answer was that, when possible, the compiler can reduce the set of disks that are accessed at any time. The last question, whether the compiler can identify data reads and reuse explicitly or can only monitor disk reuse, elicited the response that no information on data reuse is given to the compiler.

#### ■ **Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems**

*Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan, Carnegie Mellon University*

Building cluster-based storage systems using commodity TCP/IP and Ethernet networks is attractive because of their low cost and ease of use, along with the desire to use the same routing infrastructure for LAN, SAN, and high-performance computing traffic. However, Amar Phanishayee and his co-authors argue that an important barrier to high-performance storage using these commoditized networks is the problem of TCP throughput collapse: the incast problem. Incast occurs during synchronized reads of data striped over multiple storage servers, where the system is limited by the completion time of the slowest storage node, and the concurrent flood of traffic from many servers increases past the ability of an Ethernet switch to buffer packets. Dropped packets at the switch can cause one or more TCP timeouts which impose a relatively large delay until TCP recovers, resulting in a significant degradation of throughput. The authors recreate the incast problem while performing synchronized reads on an Ethernet-based storage cluster. They fix the amount of data read from each storage server, defined as a Server Request Unit (SRU), and increase the number of servers involved in a data transfer. This experiment shows an initial improvement of throughput up to around 900 Mbps with three servers, and then a sharp order-of-magnitude collapse. Other experiments fixing the block size while scaling the number of servers produce the same collapse. Although one might expect TCP to completely utilize the bottleneck link, and a tool that measures throughput using long-lived TCP streams does not experience incast, it is perplexing that a particular setup with typical communication patterns in storage systems can cause such a significant performance loss.

The authors study the network conditions that cause TCP throughput collapse, characterize its behavior under a variety of conditions, and examine the effectiveness of TCP- and Ethernet-level solutions. To understand the reasons for throughput collapse, a distinction is made between TCP's mechanism for data-driven loss recovery, which occurs when a sender receives three duplicate acknowledgments and is relatively fast, and timeout-driven loss recovery, when no feedback is available from receiver to sender and the sender must wait until the Retransmission TimeOut (RTO) time has passed before continuing the flow, a relatively slow process. Timeouts cause throughput collapse because a server loses its packets at the switch, and without any feedback it must fall back to timeout-driven recovery. Simulations of the incast problem show that doubling the switch buffer size from 32 to 64KB also doubles the number of servers supported before a collapse. However, switches that support fast buffers are expensive. Increasing the SRU size means that servers have more data to send per data



block and produce less idle time on the link; however, a larger SRU size also requires each server to do more data prefetching while the client has to allocate more memory for the complete block. The TCP variants of NewReno and SACK avoid certain timeout situations when compared to Reno, with NewReno performing best, but all variants eventually experience throughput collapse as the number of participating servers is increased. Reducing the penalty of a timeout by lowering TCP's minimum retransmission timeout period from 200 milliseconds to 200 microseconds helps significantly but is impractical and unsafe, because it requires timer support in microseconds from the operating system and may create unnecessary timeouts and retransmissions when talking to clients in the wide area network. Ethernet flow control helps only for the simplest network settings, but in more common multi-switched systems it has adverse effects on other flows, produces head-of-line blocking, and is inconsistently implemented across different switches. New standards for Data Center Ethernet are being developed to create a lossless version of Ethernet, but it is unclear when these new standards will be implemented in switches, and there are no guarantees that implementation of these standards will be uniform or that new switches will be as inexpensive as they are currently. Without any single convincing network-level solution, application-level solutions by the storage system may be more appropriate, since the storage system has the knowledge of all data flows and the control to limit situations that may cause throughput collapse.

One of the participants asked whether this problem is related to TCP using a single stream, and if a solution such as SCTP, which transports multiple message streams, would be better. Amar answered that only TCP was considered because it is used by most developers and is very simple and workable, since most machines have TCP implementations. In response to the question of whether work on an adaptive RTO would apply, Amar said that RTO is already adaptive and is based on the round-trip time estimation. Another participant asked about the queuing discipline implemented at the switch. Amar said that drop tail and random drops were used, but these didn't provide a solution. Another participant, remarking that the problem with TCP flows is that they are bursty and stay open for a long time, asked whether explicitly causing TCP to close its congestion window was used. The answer was that disabling TCP slow start in experiments did not help. To the suggestion that other TCP variants that avoid loss altogether be used, Amar responded that with the TCP variants that were tried, RED (which drops packets early) and ECN (which notifies the server to back off) were not successful in preventing incast. Two participants asked about application-level solutions, such as introducing random delay at the servers, and the response was that indeed this was one of the solutions that might help, that staggering should help to limit overflow at the switch buffer, but that very early experiments did not demonstrate much improvement.

## WHERE DID WE GO WRONG?

Summarized by James Hendricks  
(James.Hendricks@cs.cmu.edu)

### ■ Portably Solving File TOCTTOU Races with Hardness Amplification

Dan Tsafir, IBM T.J. Watson Research Center; Tomer Hertz, Microsoft Research; David Wagner, University of California, Berkeley; Dilma Da Silva, IBM T.J. Watson Research Center

#### Awarded Best Paper!



Dan Tsafir started by explaining the time-of-check-to-time-of-use (TOCTTOU) race problem. Suppose some root-privileged script deletes files in /tmp that are not accessed for a while. The script would (1) check the access time of each file F and (2) delete F if it had not been accessed recently. This approach, however, may allow an attacker to trick the script into deleting any file because there is a window of vulnerability between the check operation (examining F's access time) and the use operation (deleting F). For example, an attacker may make a directory /tmp/etc and file /tmp/etc/passwd, then symlink /etc to /tmp/etc at the right moment. Check will decide to delete /tmp/etc/passwd, but use will delete /etc/passwd because /tmp/etc will be pointed to /etc during the window of vulnerability. (Search the Web for "TOCTTOU symlink" for real-world vulnerabilities.)

TOCTTOU vulnerabilities occur between any check-use pair of system calls that involve a name of a file; in the example here it's lstat(F) and unlink(F). Thus, there are many variants of the problem, often providing attackers with the ability to obtain permanent root access. One proposed solution is *hardness amplification* (Dean and Hu). The idea is to check the condition multiple times, reducing the probability of a TOCTTOU race. Unfortunately, a maze of symbolic links makes TOCTTOU attacks much more feasible (Borisov et al.) because traversing symbolic links is slow, easily defeating such defenses and many similar proposals. The authors propose a generic check-use mechanism that emulates the kernel's file-path resolution procedure in user mode. This approach allows programmers to safely execute

Photo by Ethan Miller

most check-use operations without suffering from the associated TOCTTOU problems, effectively binding the check-use pair into an atomic transaction. The fine-grained control over the path resolution process allows programmers to express policies such as forbidding symbolic links along the path or verifying that a given user is allowed to operate on a given file. The solution is portable and works for existing systems, in contrast to prior proposals which changed the kernel or the API.

Bill Bolosky from Microsoft Research noted that the race exists because the check-use mechanism is not atomic, that transactions remove this race, and that Windows Vista has transactions. The speaker responded that the solution in the paper is portable. Another questioner asked about the cost of doing the check in userspace. The speaker responded that some performance is lost. There is a tradeoff between efficiency and safety, and the proposed mechanism takes 3–6 times longer to complete than the naive insecure alternative.

■ **EIO: Error Handling Is Occasionally Correct**

*Haryadi S. Gunawi, Cindy Rubio-González, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Ben Liblit, University of Wisconsin, Madison*

Haryadi Gunawi started by stating that errors are often improperly propagated. The authors considered 34 error codes (e.g., EIO, ENOMEM) on 51 file systems (all in linux/fs/\*) and three storage drivers (SCSI, IDE, software RAID) and found that 1153 of 9022 function calls do not save the propagated error codes. More complex file systems have more propagation violations, and writes are worse than reads. Propagation errors are common, not just corner cases. Common problems occur when the error code goes unchecked (e.g., `err = func()` is called, but `err` is not propagated), unsaved (e.g., `func()` is called, ignoring any returned errors), or overwritten (e.g., `err = func1()`; `err = func2()` is called, discarding the error code from `func1`).

The authors built a tool to map the call graph, demonstrating which error calls are incorrectly propagated. The call graphs are impressive and the reader is encouraged to peruse them in the online proceedings. Coda correctly propagates all errors (the audience applauded efforts by the Coda team); several systems incorrectly propagate more than a quarter of all errors. The authors concluded the talk and the paper with entertaining responses from developers (e.g., “Should we pass any errors back?” and “Just ignore errors at this point. There is nothing we can do except try to keep going.”).

Dave Chinner of SGI noted that some of the XFS faults have been fixed. SGI has recently implemented a tool to ensure that errors that should be checked are checked. He would be interested in a more recent run. Another questioner asked whether the tool would be released; developers often hear anecdotes of problems but are given no way to correct them. The speaker responded that the tool will be released

(it will be located at <http://www.cs.wisc.edu/adsl/Publications/eio-fast08/readme.html>). Keith Smith of NetApp asked whether they had any experience trying this technique in other parts of the kernel; the speaker responded that he is beginning to look into other parts of the kernel.

■ **An Analysis of Data Corruption in the Storage Stack**

*Lakshmi N. Bairavasundaram, University of Wisconsin, Madison; Garth Goodson, Network Appliance, Inc.; Bianca Schroeder, University of Toronto; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison*

**Awarded Best Student Paper!**



Photo by Ethan Miller

Lakshmi Bairavasundaram started by saying that files get corrupted and that corruptions are often correlated. Unfortunately, our understanding of how data is corrupted is mostly anecdotal. The authors analyzed over 1.5 million disks in thousands of NetApp systems over 41 months. There are many types of corruption, such as basic bit corruption, lost writes (writes ignored by disk), misdirected writes (blocks sent to the wrong physical location), and torn writes (in which only part of the block is written). NetApp applies various techniques, including checksums and disk scrubbing, to detect the effects of these faults, such as checksum mismatches, parity inconsistencies, and identity discrepancies. Enterprise disks have 10% as many faults as nearline disks, and bit corruption and torn writes are more common than lost writes or misdirected writes. Different models age differently (some fail early, some fail late, and some are less affected).

Nearline drives have checksum mismatches more often. But when enterprise drives have any checksum mismatches, they tend to get several mismatches. Checksum mismatches are often for consecutive or nearby blocks, with high temporal locality. Furthermore, they are not independent in a disk array. In one drive model, a particular logical block number was much more likely to exhibit faults, and certain ranges

of blocks were somewhat more likely to develop faults than others. Thus, staggering RAID stripes for each disk may be a good idea. Given the variety of faults, it is wise to make efforts to detect, prevent, and recover from faults. Preventing data loss may require double parity and vigilant scrubbing.

Mary Baker from HP Labs asked whether the numbers were broken down by firmware revision in addition to model; the speaker responded that this is a good point but that they were not. Rik Farrow asked whether, given that hard drives reorganize data on the disk, the speaker could say anything about the fact that he found correlations with block numbers. The speaker replied that the correlations were found using logical block numbers. Another questioner suggested that error correlations could be due to the NetApp boxes, given that all tests were run on NetApp systems. The speaker said that was not likely, since he found very different types of errors for different disk models. Another questioner asked whether the type of errors correlates to the type of end user, but the speaker said this was not studied. Bruce Worthington of Microsoft asked whether part of the correlation could be due to overwriting critical blocks repeatedly and whether it would make sense to move master blocks around. The speaker responded that the errors seem to be more due to firmware and software, so overwrite frequency was not a likely culprit.

---

## **BUFFERS, POWER, AND BOTTLENECKS**

---

*Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)*

### ■ **BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage**

*Hyojun Kim and Seongjun Ahn, Software Laboratory of Samsung Electronics, Korea*

Hyojun Kim proposed adding a RAM-based cache and management system to flash devices in order to improve random write performance. NAND-based flash memory has grown to become the primary storage solution for mobile devices owing to its good overall performance and low power consumption. However, the medium suffers from low random write performance, which may hamper its growth in the future.

Existing filesystems perform write operations that are ill-suited to flash memory because of hardware limitations that force I/O operations to be aligned at a coarse granularity. In addition, overwrite operations require first erasing the target area, which incurs additional cost. By adding a RAM buffer, similar to those that exist in conventional disk drives, write operations can be coalesced and performed more efficiently. The Block Padding Least Recently Used (BPLRU) algorithm was introduced to manage this buffer. It operates by merging adjacent requests such that they can be written at the block size of the flash unit and by prioritizing these lower-cost write operations. The system was evaluated with trace-based simulations, a subset of which were then verified on a prototype directly.

Brent Welch of Panasas wondered about the buffer's operation under power failure. The current system does not tolerate this scenario, but Kim identified the problem as one that may be solved in the future. The potential for increased latency was also a concern, as it was not formally evaluated.

### ■ **Write Off-Loading: Practical Power Management for Enterprise Storage**

*Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron, Microsoft Research Ltd.*

Dushyanth Narayanan made the case that significant power savings can be achieved in data centers by spinning down idle disks. He also demonstrated that further savings are possible when writes to idle disks are redirected to already active disks. The results, which were based on a trace of measurements of real workloads, stand in contrast to previous benchmark-based findings that suggested such an approach to be impractical.

Narayanan described an approach that allows incremental deployment by working with existing storage and file systems. By adding a management module to the storage stack of each volume, drives that idle for some period of time can be spun down. In addition, future writes to such a disk can be redirected to another disk that is already active, thus avoiding the overheads associated with spinning up idle disks, and allowing fewer disks to remain in operation at any given time. This latter technique was referred to as write off-loading.

In servicing these off-loaded write requests, a log structure associated with the remapping operation is stored on the active disk at a known location. In addition, a record of the remapped block is stored in the memory of the original host. Future reads may then be redirected to the appropriate disk, which will be activated if needed. These remapping operations are temporary, which ensures that the underlying storage structure is not degraded. The system tolerates failure by reestablishing the block locations after an unclean shutdown. Narayanan also showed that the increased latency associated with an operation that activates an idle disk was significant but that this occurrence was rare in practice.

The audience responded actively, allowing Narayanan to elaborate on several topics. He described the load-based heuristic for selecting an off-loading target and the transparency benefits to ensuring that remapped blocks are short-lived. He also asserted that the benefits of write off-loading would be present even on a system that had consolidated its data onto fewer disks.

### ■ **Avoiding the Disk Bottleneck in the Data Domain Deduplication File System**

*Benjamin Zhu, Data Domain, Inc.; Kai Li, Data Domain, Inc., and Princeton University; Hugo Patterson, Data Domain, Inc.*

Data Domain's system for enterprise data backup and network-based disaster recovery was described by Benjamin

Zhu. He showed how the company's file system provides a high degree of data deduplication and off-site backup, with good performance. The system was evaluated with data gathered from real customers, which showed a data compression rate up to 30 times, with throughput meeting or exceeding the system's 100 MB/s target.

After providing some background, Zhu detailed the differences between primary storage and backup. Whereas the former focuses on latency, seek times, and throughput, the latter is concerned with large batched writes and space efficiency. In the case of a remote backup, bandwidth over the WAN link is also a concern. The Data Domain File System targets these problems with a combination of compression and deduplication based on content hash.

In a large data set, detecting duplicates can be costly. The size of the hash table can quickly exceed the available RAM and, when flushed to disk, the table shows no temporal or spatial locality. To address this challenge, a Bloom filter is used to determine whether a particular segment is a duplicate. Since a negative result from the Bloom filter is only probabilistically correct, such a hypothesis must be confirmed by consulting an on-disk structure. To help regain locality, segments written by the same stream are placed together on-disk in containers. Once duplicate data is merged in storage, it is compressed to further save space.

After the presentation, a lengthy exchange ensued regarding the performance of the system at some boundary conditions. At least one attendee was concerned that a long series of nonduplicate data could overwhelm the system's ability to efficiently flush metadata. The debate was deferred until later before any consensus was reached. Others wondered about the potential for hash collision. Zhu initially said that such a collision was rare enough to be of no concern; he later clarified, explaining that a mechanism to test for collisions existed but was disabled by default.

## COMPLIANCE AND PROVISIONING

*Summarized by Chris Frost (frost@cs.ucla.edu)*

### ■ **Towards Tamper-evident Storage on Patterned Media**

*Pieter H. Hartel, Leon Abelman, and Mohammed G. Khatib, University of Twente, The Netherlands*

Everyone wants to prevent data tampering, and Write-Once Read-Many (WORM) devices may be of help. Compared to disks and flash, they have high access times and low throughput. Hartel suggested that an alternative approach to tamper-resistant storage is tamper-evident storage, where data may be modified, but any modification will be detected. Common techniques include calculating the hash of data and writing this to a notebook and giving it to a notary public. The difficulty with these approaches lies in managing these hashes and keeping them consistent with their tracking of the data-hash pairs. This talk proposed a hardware device based on patterned media that can store

both Write-Many Read-Many and WORM data, effectively providing Selectively Eventually Read-Only (SERO) storage.

Hartel et al. show how, in theory, a device could support magnetic read and write operations as well as electrical read and write operations. An electrical write would permanently change the location's magnetic properties, but electrical reads and writes are significantly slower than the magnetic equivalents, so one could make data tamper-evident by storing only a secure hash electrically. Specifically, they propose storing each hash bit in two bits with a parity of one (the Manchester encoding) of electrical storage. To modify data an attacker would need to either find data with an equal hash or modify the hash value. But an attacker could only set unset bits in the hash, and this would change the bit's parity. They also presented a filesystem design for SERO storage, in which fragmentation becomes a serious concern as more of the disk becomes read-only.

Peter Honeyman asked whether they have simulated operation timings. Hartel answered that they have not, but that they hope patterned media will support both archival and online storage. Bill Bolosky asked what would stop an attacker from blanking the entire SERO device and rewriting it with modifications, to work around the tamper-evidence guards. The authors replied that one could, but this would take some time, and perhaps the device could record the fact.

### ■ **SWEEPER: An Efficient Disaster Recovery Point Identification Mechanism**

*Akshat Verma, IBM India Research; Kaladhar Voruganti, Network Appliance; Ramani Routray, IBM Almaden Research; Rohit Jain, Yahoo! India*

Akshat Verma presented their work on quickly finding a backup snapshot from before a latent error occurred. Current methods for quickly finding such a backup are ad hoc; their system systematizes the location process. SWEEPER logs system events that, with a goal recovery time and recovery point objective, help speed up good backup identification. SWEEPER's balanced search strategy calculates probabilities that certain events are correlated with the specified error and uses its event log, along with a binary search, to locate good backups. Example events include misconfiguration, virus activity, hardware warnings and errors, and applications logs.

An audience member asked about the sensitivity of their benchmark to event weights. Verma replied that SWEEPER's informed search can be way off but that this is acceptable because the binary search will still find a good backup.

### ■ **Using Utility to Provision Storage Systems**

*John D. Strunk, Carnegie Mellon University; Eno Thereska, Microsoft Research, Cambridge, UK; Christos Faloutsos and Gregory R. Ganger, Carnegie Mellon University*

In provisioning a storage system (e.g., for OLTP or scientific or archival purposes), trade-offs are unavoidable. For

example, increasing data protection can harm performance or increase purchase cost. Whereas the existing practice is to consult an area expert, John Strunk spoke on how utility functions can convey the cost-benefit structure to an automated provisioning tool. Users are then able to make appropriate trade-offs among various system metrics.

Strunk et al. use utility functions, functions from a set of metrics (e.g., revenue, availability, data value, power usage, or purchase cost) to a utility value (e.g., dollars), to characterize a particular point in the purchase space. To find a desirable point in this (large) space they use a genetic algorithm to refine a configuration population over many generations. Strunk then illustrated the value of this approach through three case studies, including scenarios with a limited budget and where system cost can affect the long-term solution.

Peter Honeyman asked why linear programming was not used instead of a genetic algorithm. Strunk answered that linear programming's constraints on objective function form rules out many real-world utility functions. Honeyman also asked whether one can maximize multiple objectives; Strunk replied that you would convert these to one utility. Another audience member asked whether they had looked at a method for generating good utility functions, noting that Strunk's seemed simplistic. Strunk said they have, that the paper has more examples, and that this is also an area where they are doing further work. One person asked whether this approach can determine whether it is better to upgrade an existing system or migrate to a new system. Strunk answered that they can do this, but that it is the second part of his thesis. Two audience members asked whether Strunk's approach supported varying input values as a function of time. Strunk answered that their system focuses only on static provisioning. The final questioner asked whether not finding the most optimal solution is a problem. Strunk replied that in the real world one often only gets in the ballpark, and that this approach already does at least as well as today's ad hoc approaches.

---

## LSF '08: 2008 Linux Storage & Filesystem Workshop

San Jose, CA  
February 25-26, 2008

---

### STORAGE TRACK

Summarized by Grant Grundler ([grundler@google.com](mailto:grundler@google.com))  
Copyright 2008 Google, Inc. (Creative Commons Attribution License, <http://code.google.com/policies.html> or <http://creativecommons.org/licenses/by/2.5/>)

Several themes came up over the two days:

#### *Theme 1: Solid State Drives*

SSDs (Solid State Disks) are coming. There was a good presentation by Dongjun Shin (Samsung) on SSD internal

operation, including some discussion on which parameters were needed for optimal operation (theme #2). The I/O stack needs both micro-optimizations (performance within driver layers) and architectural changes (e.g., you have to parameterize the key attributes so that file systems can utilize SSDs optimally). Intel presented SCSI RAM and ATA\_RAM drivers to help developers tune the SCSI, ATA, and block I/O subsystems for these orders-of-magnitude-faster (random read) devices. Hybrid drives were a hot topic at LSF '07 but were only briefly discussed in the introduction this year.

#### *Theme 2: Device Parameterization*

The device parameters discussion is just beginning on how to parameterize device characteristics for the block I/O schedulers and file systems. For instance, SSDs want all writes to be in units of the erase block size if possible, and device mapping layers would like better control over alignment and placement. The key object here is how to provide enough parameters to be useful but not so many that "users" (e.g., the file system) get it wrong. The general consensus was that having more than two or three parameters would cause more problems than it solved.

#### *Theme 3: I/O Priorities*

I/O priorities and/or bandwidth sharing has lots of folks interested in I/O schedulers. There was consideration about splitting the I/O scheduler into two parts: an upper half to deal with different needs of feeding the Q (limit block I/O resource consumption) and a lower half to rate-limit what gets pushed to the storage driver.

#### *Theme 4: Network Storage*

Two technologies were previewed for addition to the Linux kernel: pNFS (parallel NFS) and FCoE (Fiber Channel over Ethernet). Neither is ready for kernel.org inclusion, but some constructive guidance was given on what directions specific implementations needed to take.

The issues facing iSCSI were also presented and discussed. User- versus kernel-space drivers was a hot topic in Networked Block Storage forums.

#### ■ *Introduction and Opening Statements: Recap of Last Year*

*Chris Mason and James Bottomley*

This session was primarily a scorecard of how many topics discussed last year are fixed or implemented this year. The bright spots were the new filesystem (BTRFS, pronounced "butter FS," which incorporates B-trees for directories and an extent-based filesystem with 2<sup>64</sup> maximum file size) and emerging support for OSD (Object-base Storage Device) in the form of bidirectional command integration (done) and long CDB commands (pending); it was also mentioned that Seagate is looking at producing OSD drives.

Error handling was getting better, but there's still a lot of work to be done and we have some new tools to help test error handling. The 4k sector size, which was a big issue

last year, has receded in importance because manufacturers are hiding the problem in firmware.

## ■ SSD

*Dongjun Shin, Samsung Electronics*

Dongjun gave an excellent introduction and details of how SSDs are organized internally (sort of a two-dimensional matrix). The intent was to give FS folks an understanding of how data allocation and read/write requests should be optimally structured. “Stripes” and “channels” are the two dimensions to increase the level of parallelization and thus increase the throughput of the drive. The exact configurations are vendor-specific. The tradeoff is to reduce stripe size to allow multithreaded apps to have multiple I/Os pending without incurring the “lock up a channel during erase operation” penalty for all pending I/Os. Hard disk drives (HDDs) prefer large sequential I/Os, whereas SSDs prefer many smaller random I/Os.

Dongjun presented postmark (mail server benchmark) performance numbers for various file systems. An obvious performance leader seemed to be nilfs for most cases, and it was never the worst. Successive slides gave more details on some of the FSes tested. Some notable issues were that flush barriers kill XFS performance and that BTRFS performance was better with 4k blocks than with 16k blocks.

Flush barriers are the only block I/O barriers defined today, and the flush barriers killed performance on the SSDs since the flash translation layer could no longer coalesce I/Os and had to write data out in blocks smaller than the erase block size. Ideally, the file system would just issue writes using erase block sizes.

## ■ Error Handling

*Ric Wheeler, EMC*

Ric Wheeler introduced the perennial error-handling topic with the comment that bad sector handling had markedly improved over the “total disaster” it was in 2007. He moved on to silent data corruption, noting that the situation here was improving with data checksumming now being built into file systems (most notably BTRFS and XFS) and emerging support for T10 DIF. The “forced unmount” topic provoked a lengthy discussion, with James Bottomley claiming that, at least from a block point of view, everything should just work (surprise ejection of USB storage was cited as the example). Ric countered that NFS still doesn’t work and others pointed out that even if block I/O works, the file system might still not release the inodes. Ted Ts’o closed the debate by drawing attention to the paper by Gunawi et al. at FAST ’08 showing over 1,300 cases where errors were dropped or lost in the block and filesystem layers.

Error injection was the last topic. Everybody agreed that if errors are forced into the system, it’s possible to consistently check how errors are handled. The session wrapped up with Mark Lord demonstrating new `hdparm` features that

induce an uncorrectable sector failure on a SATA disk with the `WRITE_LONG` and `WRITE_UNC_EXT` commands. This forces the on-disk CRCs to mismatch, thus allowing at least medium errors to be injected from the base of the stack.

## ■ Power Management

*Kristen Carlson Accardi, Intel*

Arjan van de Ven wrote `PowerTOP` and it’s been useful in tracking down processes that cause CPU power consumption but not I/O. Although `kjournald` and `pdflush` are shown as the apps responsible, obviously they are just surrogates for finishing async I/O. For example, `postfix` uses sockets, which triggers inode updates. Suggestions for preventing this include using lazy update of nonfile inodes and virtual inodes.

With ALPM (Aggressive Link Power Management, <http://www.lesswatts.org/tips/disks.php>), up to 1.5 watts per disk can be saved on desktop systems. Unlike disk drives, no hardware issues have been seen with repeated powering up or down of the physical link, so this is safer to implement. Performance was of interest since trading off power means some latency will be associated with coming back up to a full-power state. The transition (mostly from Async Negotiation (AN) when restoring power to the Phys) from SLUMBER to ACTIVE state costs about ~10 ms. Normal benchmarks show no performance hit, as the drive is always busy. We need to define a bursty power benchmark that is more typical of many environments.

Kristen presented three more ideas on where Linux could help save power. The first was to batch-average group I/O; 5–30 seconds is normal to flush data, so instead wait up to 10 minutes before flushing these. The second suggestion was a question: Can the block layer provide hints to the low-level driver? For example, “Soon we are going to see I/O; wake up.” The third suggestion was making smarter timers to limit CPU power-up events—that is, coordinate the timers so they can wake up at the same time, do necessary work, then let the CPU go to a low-power state for a longer period of time.

Ric Wheeler (EMC) opened the discussion on powering down disks, since the savings there are typically 6–15 watts per disk. But powering up disks requires coordination across the data center.

Eric Reidel (Seagate) mentioned EPA requirements: Should we idle CPU versus the hard drive? One would be trading off power consumption for data access. He said that Seagate can design for higher down/up lifecycles. Currently, it’s not a high count only because Seagate is not getting data from OEMs on how high that count needs to be. It was noted that one version of Ubuntu was killing drives after a few months by spinning them down or up too often.

## ■ **Block IO Resources and cgroups**

*Fernando Luis Vazquez Cao*

Cao touched on three related topics: block I/O (BIO) resources and cgroups, which define arbitrary groupings of processes; I/O group scheduling; and I/O bandwidth allocation (ioband drivers, which manage I/O bandwidth available to those groups). The proposals were not accepted as is but the user-facing issues were agreed upon. The use case would be Xen, KVM, or VMware.

Currently, the I/O priority is determined by the process that initiated the I/O. But the I/O priority applies to *all* devices that process is using. This changed in the month preceding the conference, and the speaker acknowledged that. A more complex scheme was proposed that supports hierarchical assignment of resource control (e.g., CPU, memory, I/O priorities). Proposed was `page_cgroup` to track write bandwidth. The page would get assigned to a cgroup when the BIO is allocated. One advantage of the `get_context()` approach is that it does *not* depend on the current process and thus would also work for kernel threads.

Idea #1 proposed a layer between the I/O scheduler and the I/O driver. This requires some changes to `elevator.c` and additional infrastructure changes. Jens Axboe pointed out that one can't control the incoming queue from below the block I/O scheduler. The scheduler needs to be informed when the device is being throttled from below in order to prevent the I/O scheduler queue from getting excessively long and consuming excessive memory resources. Jens suggested they start with #1 since it implements fairness.

Idea #2 was generally not accepted. For idea #3 (group scheduler above LVM `make_request`), adding a hook so cgroup can limit I/O handed to a particular scheduler was proposed and this idea got some traction. Jens thought #3 would require less infrastructure than #1. Effectively, #3 would lead to a variable-sized Q-depth. And #3 would limit BIO resource allocation.

## ■ **NCQ Emulation**

*Gwendal Grignou, Google*

Gwendal started by explaining what Native Command Queuing (NCQ) was, his test environment (`fio`), and which workloads were expected to benefit. In general, the idea is to let the device determine (and decide) the optimal ordering of I/Os since it knows current head position on the track and the seek times to any I/Os it has in its queue. Obviously, the more choices the device has, the better choices it can make and thus the better the overall throughput the device will achieve. Results he presented bear this out, in particular for small (<32k), random read workloads (e.g., for a classic database).

But the problem is that since the device is deciding the order, it can choose to ignore some I/Os for quite a while too. And thus latency-sensitive applications will suffer occasionally, with I/Os taking more than 1–2 seconds to complete.

He implemented and showed the results of a queue plugging that starved the drive of new I/O requests until the oldest request was no longer over a given threshold. Other methods to achieve the same effect were discussed but each had its drawbacks (including this one).

He also showed how by pushing more I/O to the drive, we affect the behavior of block schedulers to coalesce I/O and anticipate which I/Os to issue next. And although NCQ was effective on a best-case benchmark, it was debated how effective it would be in real life (perhaps <5%).

## ■ **Making the IO Scheduler Aware of the Underlying Storage Topology**

*Aaron Carroll and Joshua Root, University of New South Wales*

Disclosure: Grant Grundler arranged the grant from Google to fund this work. HP is also funding a portion of this work.

Aaron and Joshua have created an infrastructure to measure the performance of any particular block trace and were interested in seeing how I/O schedulers behave under particular workloads. The performance slides are graphs of how the various schedulers perform as one increases the number of processes generating the workload. They tested the following schedulers: AS (Anticipatory Scheduler), CFQ (Completely Fair Queueing), Deadline, FIFO, and NOOP.

They tested a few different configs: RAID 0 sequential, async; single-disk random and sequential; and 10-disk RAID 0 random and sequential. Of the various parameters—queue depth, underlying storage device type, and RAID topology—they wanted to establish which parameters were relevant and find the right way to determine those parameters (e.g., by user input, with runtime microbenchmark measurements, by asking lower layers). Queue depth is generally not as important nor is it very helpful for any sort of anticipation. For device type, it would be obvious to ask the underlying device driver but we need a suitable level of abstraction. For RAID topology, the key info was “stripe boundaries.”

Ric Wheeler said that he can see differences in performance depending on the seek profile if most I/Os are to one disk at a time and if Array is doing read ahead. Random reads for RAID 3/5/6 depend on worst case (i.e., the slowest drive). Jens mentioned that disk type could be exported easily by plugging (stopping Q to build a bigger I/O) or through an anticipatory maneuver (starting new I/O, after the previous one has completed but before the application has requested the data/metadata). We discussed how to split fairness/bandwidth sharing/priorities (or whatever you want to call it) so that a component above the SW RAID md driver would manage incoming requests. A lower half of the scheduler would do a time slice. It was also noted that CFQ can unfairly penalize bursty I/O measurements. One suggestion was to use Token Bucket to mitigate bursty traffic. Aaron and Joshua introduced two new schedulers that

might be useful in the future: FIFO (true fifo, without merging) and V(R) SSTF. There was no discussion on these.

#### ■ **DMA Representations: SG\_table vs. SG\_ring IOMMUs and LLD's Restrictions**

*Fujita Tomonori*

(LLD stands for Low Level Driver, e.g., a NIC or an HBA device driver.)

Fujita did an excellent job of summarizing the current mess that is used inside the Linux kernel to represent DMA capabilities of devices. As Fujita dove straight into the technical material with no introduction, I'll attempt to explain what an IOMMU is and the Kernel DMA API. Historically, I/O devices that are *not* capable of generating physical addresses for all of system RAM have always existed. The solution without an IOMMU is a “bounce buffer” in which you DMA to a low address the device can reach and then memcopy to the target location. I/O Memory Management Units (IOMMUs) can virtualize (a.k.a. remap) host physical address space for a device and thus allow these legacy devices to directly DMA to any memory address. The bounce buffer is no longer necessary and we save the CPU cost of the memcopy. IOMMUs can also provide isolation and containment of I/O devices (preventing any given device from spewing crap over random memory—think Virtual Machines), merge scatter-gather lists into fewer I/O bus addresses (more efficient block I/O transfers), and provide DMA cache coherency for virtually indexed/tagged CPUs (e.g., PA-RISC).

The PCI DMA Mapping interface was introduced into the Linux 2.4 kernel by Dave Miller primarily to support IOMMUs. James Bottomley updated this to support noncache coherent DMA and become bus-agnostic by authoring the Documentation/DMA-API.txt in Linux 2.6 kernels. The current DMA API also does not require the IOMMU drivers to respect the max segment length (i.e., IOMMU support is coalescing DMA into bigger chunks than the device can handle). The DMA alignment (i.e., boundaries a DMA cannot cross) has similar issues (e.g., some PCI devices can't DMA across a 4-GB address boundary). Currently, the drivers that have either length or alignment limitations have code to split the DMA into smaller chunks again. The `max_seg_boundary_mask` in the request queue is not visible to IOMMU, since only `struct device *` is passed to IOMMU code.

The next issue discussed was IOMMU performance and I/O TLB flushing. The IOMMU driver (and HW) performance are critical to good system performance. New x86 platforms support virtualization of I/O; and thus it's not just a high-end RISC computer problem. Issues included the following:

1. How does one best manage IOMMU address space? Through common code? Some IOMMU drivers use bit-map (most RISC); Intel uses a “Red Black” tree. Fujita tried converting POWER to use Red/Black tree and lost 20% performance with `netperf`. Bottomley and Grundler agree that the address allocation policy needs to be managed by

the IOMMU or architecture-specific code since I/O TLB replacement policy dictates the optimal method for allocating IOMMU address space.

2. When should we flush I/O TLB? One would like to avoid flushing the I/O TLB since (a) it's expensive (as measured in CPU cycles) and (b) it disturbs outstanding DMA (forces reloading I/O TLB). However, if we flush the entries when the driver claims the DMA is done, we can prevent DMA going to a virtual DMA address that might have been freed and/or reallocated to someone else. The bottom line is that there is a tradeoff between performance and safety (a.k.a. robustness).

3. Should we just map everything once? The performance advantage is that you don't need to map, unmap, and flush I/O TLB for individual pages, but the tradeoff is isolation (since any device can DMA anywhere), which can be useful in some cases (e.g., embedded devices such as an NFS server).

The last DMA-mapping-related issue was SG (SCSI Generic) chaining versus SG rings.

#### ■ **iSCSI Transport Class Simplification**

*Mike Christie and Nicholas Bellinger*

The main thrust here is that common libs are needed to share common objects between transport classes. In particular, Mike called out the issues that the iSCSI maintainer has faced across different kernel versions where `/sys` has evolved. James Bottomley conceded that there were issues with the original implementation. Mike also mentioned problems with parsing `/sys` under iSCSI devices. The goal is to provide a common starting point for user-space-visible names.

Mike proposed a `scsi_transport_template` that contained new `scsi_port` and `scsi_i_t_nexus` data structures. iSCSI also needs an abstraction between SCSI ports—an `I_T_nexus`. Other users of `I_T_nexus` were also discussed.

James Bottomley pointed out that `libsas` already has an `I_T_nexus` abstraction. It provides a `host/port/phy/rphy/target/lun` hierarchy for `/sys`. However, the exported paths need to be more flexible. Mike floated the idea of a new library to encapsulate the SCSI naming conventions so that tools like `lsscsi` wouldn't have to struggle.

Development for iSCSI focuses on `Linux-iSCSI.org`. iSCSI exposed issues with error recovery. The slides neatly summarize most of the points Nicholas wanted to make. The lively but inconclusive debate left me thinking that most of the code will be forced to live in user space until evidence is presented otherwise. iSCSI, FC, and SAS would be better in kernel because concurrency control fundamentally resides in the kernel. And LIO-SE assumes most drivers belong and are implemented in kernel space because transport APIs force middle code into kernel. KVM performance suffers because of movement among virtual kernels.



### ■ **Request-based Multi-pathing**

*Kiyoshi Ueda and Jun'ichi Nomura, NEC*

The key point was proposed multi-path support below the I/O scheduler; this seems to be the favored design. Problems are expected with request completion and cleaning up the block layer. An RFC for a request stacking framework was posted to linux-scsi and linux-ide mailing lists. See the last slide (37) for URLs to postings. The big advantage of request-based DM (Device Mapper) multi-path is that, since BIOs are already merged, the multi-path driver can do load balancing since it knows exactly how many I/Os are going to each available path.

Three issues were raised. The first issue was that `blk_end_request()` will deadlock because the queue lock is held through the completion process. Bottomley suggested moving completions to tasklet (soft IRQ) since SCSI at one point had the same issue. There was also some discussion about migrating drivers to use `blk_end_request` instead of `__blk_end_request()`. The second issue involved busy stack drivers that won't know when the lower driver is busy, and once a request is removed from the scheduler queue, it's no longer mergeable. Slides 14–21 have very good graphic representations of the problem. Bottomley suggested prep and unprep functions to indicate whether requests are mergeable or not. One basic difference between BIO (existing code) and proposed Request DM is that device locking (queue lock) will be required for both submission and completion of the Request DM handler I/Os and is not required by BIO. The third issue was that `req->end_io()` is called too late and is called with a queue lock held. Solutions were offered and discussed in the remaining slides (29–36).

Regarding issue 1, one should only allow use of nonlocking drivers (i.e., drivers that do not lock in the completion path). All SCSI drivers, `cciss`, and `i2o` already meet this criterion; Block Layer is using locking completion; a DASD driver change is needed. There was a discussion about how much work it was to convert other drivers.

### ■ **FS and Volume Managers**

*Dave Chinner, SGI*

Dave covered several major areas: a proposal he called “BIO hints” (which Val Hansen called “BIO commands”); DM multi-path; chunk sizes; and I/O barriers. BIO hints is an attempt to let the FS give the low-level block hints about how the storage is being used. The definition of “hint” was something that the storage device could (but was not required to) implement for correct operation. The function `mkfs` could provide the “space is free” hints and would be good for RAID devices, transparent security (zero released data blocks), and SSDs, which could put unused blocks in its garbage collection.

DM multi-path has a basic trust issue. Most folks don't trust it because the necessary investment wasn't made to make it trustworthy. This is a chicken-and-egg problem. Ric Wheeler said that EMC does certify DM configs. Other com-

plaints were poor performance, the lack of proper partitioning, the poor user interface for management tools, and the total lack of support for existing devices.

Barriers today are only for cache flushing, both to force data to media and to enforce ordering of requests. Bottomley suggested implementing commit on transaction.

### ■ **OSD-based pNFS**

*Benny Halevy and Boaz Harrosh, Panasas*

Benny first described the role of the layout driver for OSD-based pNFS. Layouts are a catalog of devices, describing the byte range and attributes of that device. The main advantage of the layout driver is that one can dynamically determine the object storage policy. One suggestion was to store small files on RAID1 and large files on RAID5. Striping across devices is also possible. By caching the layouts (object storage server descriptions), one can defer cataloging all the OSD servers at boot time and implement on-demand access to those servers.

Current device implementations include iSCSI, iSER, and FC. SCSI over USB and FCoE are also possible. Functional testing has been done and performance was described as being able to “saturate a GigE link.” Future work will include OSD 2.0 protocol development, and it's already clear there will be changes to the OSD protocol.

Requirements of the Linux kernel to support OSD pNFS were discussed. Bidirectional SCSI CDB support is in 2.6.25-rcX kernels. There are no objections to patches for variable-length CDBs, which might go into 2.6.26. Recent patches to implement “Long Sense Buffers” were rejected; a better implementation is required.

The discussion ended on DM and ULD (Upper Level Driver; e.g., `sd`, tape, CD/DVD). DM contains the desired striping functionality, but it also takes ownership of the device. Distributed error handling is not possible unless the DM would pass errors back up to high layers. Each ULD is expected to register an OSD type. But the real question is whether we want to represent objects as block devices (segue to the next talk) and how to represent those in some namespace.

### ■ **Block-based pNFS**

*Andy Adamson, University of Michigan; Jason Glasgow, EMC*

Afterward, pNFS was summarized to me as “clustered FS folks . . . trying to pull coherency into NFS.” The underlying issue is that every clustered filesystem (e.g., Lustre) requires coherency of metadata across nodes of the cluster. NFS historically has bottlenecked on the NFS server, since it was the only entity managing the metadata coherency.

The first part of this talk explained the Volume Topologies and how pNFS block devices are identified (`fsid`). Each `fsid` can represent arbitrarily complex volume topologies, which under DM get flattened to a set of DM targets. But they didn't want to lose access to the hierarchy of the underlying storage paths in order to do failover.

The proposal for “Failover to NFS” survived Benny’s explanation of how a dirty page would be written out via block path, and if that failed, then via NFS code path. The main steps for the first path would be write, commit, and log commit and, for the failover path, write to MDS and commit. This provoked sharp criticism from Christoph Hellwig: this adds complexity without significant benefit. The client has two paths that are completely different, and the corner cases will kill us. The complexity he referred to was the unwinding of work after starting an I/O request down the block I/O code path and then restarting the I/O request down a completely different code path. A lively debate ensued around changes needed to Block I/O and VFS layers. Christoph was not the only person to object and this idea right now looks like a nonstarter. The remaining issue covered block size: 4k is working but is not interoperable with other implementations.

### ■ **FS and Storage Layer Scalability Problems**

*Dave Chinner, SGI*

Dave offered random thoughts on 3- to 5-year challenges. The first comment was “Direct I/O is a solved problem and we are only working on micro-optimizations.”

He resurrected and somewhat summarized previous discussion on exposing the geometry and status of devices. He wanted to see independent failure domains being made known to the FS and device mapper so that those could automate recovery. Load feedback could be used to avoid hot spots on media I/O paths. Similarly, failure domains and dynamic online growing could make use of loss-redundancy metrics to automate redistribution of data to match application or user intent.

Buffered I/O writeback (e.g., using `pdflush`) raised another batch of issues. It’s very inefficient within a file system because the mix of metadata and data in the I/O stream causes both syncing and ordering problems. `pdflush` is also not NUMA aware and should use CPUsets (not Containers) to make `pdflush` NUMA aware. James Bottomley noted that the I/O completion is on the wrong node as well (where the IRQ is handled). Finally, different FSes will use more or less CPU capacity and functionality such as checksumming data, and aging FS might saturate a single CPU. He gave an example where the raw HW can do 8 GB/s but only sees 1.5 GB/s throughput with the CPU 90% utilized. Dave also revisited the topic of error handling with the assertion that given enough disks, errors are common. He endorsed the use of the existing error injection tools, especially `scsi_debug` driver.

His last rant was on the IOPS (I/O per second) challenge SSDs present. He questioned that Linux drivers and HBAs are ready for 50k IOPS from a single spindle. Raw IOPS are limited by poor HBA design with excessive per-transaction CPU overhead. HBA designers need to look at NICs. Using MSI-X direct interrupts intelligently would help, but both SW and HW design to evolve. I’d like to point folks

to `mmio_test` (see [gnumonks.org](http://gnumonks.org)) so they can measure this for themselves. Disclaimer: I’m one of the contributors to `mmio_test` (along with Robert Olsson, Andi Kleen, and Harald Welte). Jörn Engel added that about 2 years ago tasklets were added which now do the equivalent of NAPI (“New API” for NIC drivers). NAPI was added about 5 or 6 years ago to prevent incoming NIC traffic from live-locking a system. All the CPU cycles could be consumed exclusively handling interrupts. This interrupt mitigation worked pretty well even if HW didn’t support interrupt coalescing.

### ■ **T10 Dif**

*Martin Petersen, Oracle*

Martin pointed to the FAST ’08 paper “An Analysis of Data Corruption in the Storage Stack” by Bairavasundaram et al. (See the related article in this issue of *login*.)

His first point was that data can get corrupted in nearly every stage between host memory and the final storage media. The typical data-at-rest corruption (a.k.a. “grown media defects”) is just one form of corruption. Remaining data corruption types are grouped as “while data is in flight” and applications need to implement the first level of protection here. He also characterized Oracle’s HARD as the most extreme implementation and compared others to “bong hits from outer space.” Given the volume of data being generated, there was agreement that the trivial CRCs would not be sufficient.

Although some vendors are pushing file systems with “logical block cryptographically strong checksumming” and similar techniques as bullet-proof, they only detect the problems at read time. This could be months later, when the original data is long gone. The goal of the TDIF (T10 Data Integrity Feature) standard was to prevent bad data from being written to disk in the first place.

HW RAID controllers routinely reformat FC and SCSI drives to use 520-byte sectors to store additional data integrity/recovery bits on the drive. The goal of TDIF was to have end-to-end data integrity checks by standardizing and transmitting those extra 8 bytes from the application all the way down to the media. This could be validated at every stop on its way to media and provide end-to-end integrity checking of the data.

He pointed out which changes are needed in the SCSI; one of those (variable-length CDBs) is already in the kernel. James Bottomley observed that he could no longer get SCSI specs to implement new features like this one, owing to recent changes in distribution. He also pointed out that the FS developers could use some of the tag CRC bits to implement a reverse-lookup function they were interested in. The best comment, which closed the discussion, came from Boaz Harrosh: Integrity checks are great! They catch bugs during development!

## ■ FCoE

*Robert Love and Christopher Leech*

Robert and Christopher took turns giving a description of the project, providing an update on current project status, and leading a discussion of issues they needed help with.

FCoE is succinctly described as “an encapsulation protocol to carry Fibre Channel frames over Ethernet” and standardized in T11. The main goal of this is to integrate existing FC SAN into a 10-GigE network and continue to use the existing FC SAN management tools. The discovery protocol is still under development. James Bottomley observed that VLAN would allow the FC protocol to pretend there is no other traffic on the Ethernet network, since the on-wire protocol supports 802.1Q tags.

Open-FCoE.org seems to be making good progress on several areas but it’s not ready for production use yet. Current problems discussed included the complexity of the code, frustration with the (excessive) number of abstractions, and wanting to take advantage of current NIC offload capabilities. Current rework is taking direction from James Smart, making better use of existing Linux SCSI/FC code and then determining how much code could be shared with existing FC HBA drivers.

Discussion covered making use of proposed “IT\_Nexus” support. Robert and Christopher agreed that IT\_Nexus would be useful for FCoE as well, since they had the same issues as others managing the connection state. James Bottomley also pointed out that their current implementation didn’t properly handle error states; he got a commitment back that Robert would revisit that code.

## ■ Linux Storage Stack Performance

*Kristen Carlson Accardi and Mathew Wilcox, Intel*

Kristen and Mathew “willy” Wilcox provided forward-looking performance tools to address expected performance issues with the Linux storage stack when used with SSDs (see [http://www.usenix.org/events/lsf08/tech/Carlson\\_Accardi\\_powermgmt.pdf](http://www.usenix.org/events/lsf08/tech/Carlson_Accardi_powermgmt.pdf)). This follows the “provide data and the problem will get solved” philosophy. Storage stacks are tuned for seek avoidance (waste of time for SSDs) and SSDs are still fairly expensive and uncommon. The underlying assumption is that lack of SSDs in the hands of developers means the data won’t get generated and no one will accept optimizations that help SSDs.

Kristen’s first slides, providing a summary of SSD cost/performance numbers (e.g., Zeus and Mtron), showed that a single device is now capable of 50,000+ IOPS (I/O per second). Current rotational media can only do 150–250 IOPS per device on a random read workload (3000–4000 if it’s only talking to the disk cache) and largely depend on I/O request merging (larger I/O sizes) to get better throughput. Ric Wheeler pointed out that EMC’s disk array can actually do much more, but it requires racks of disks. Kristen’s point was that this level of performance will be in many laptops

soon and it would be great if Linux could support that level of performance.

## ■ SYSFS Representations

*Hannes Reinecke and Kay Sievers, SuSE*

*Summarized by James Bottomley*

*(James.Bottomley@HansenPartnership.com)*

Hannes Reinecke and Kay Sievers led a discussion on sysfs in SCSI. They first observed that SCSI represents pure SCSI objects as devices with upper-layer drivers (except SCSI Generic) being SCSI bus drivers. However, everything else, driven by the transport classes, gets stored as class devices. Kay and Greg want to eliminate class devices from the tree, and the SCSI transport classes are the biggest obstacle to this. The next topic was object lifetime. Hannes pointed to the nasty race SCSI has so far been unable to solve where a nearly dead device gets re-added to the system and can currently not be activated (because a dying device is in the way). Hannes proposed the resurrection patch set (bringing dead devices back to life). James Bottomley declared that he didn’t like this. A heated discussion ensued, during which it was agreed that perhaps simply removing the dying device from visibility and allowing multiple devices representing the same SCSI target into the device list but only allowing one to be visible might be the best way to manage this situation and the references that depend on the dying device.

Noncontroversial topics were reordering target creation at scan time to try to stem the tide of spurious events they generate and moving SCSI attributes to default attributes so that they would all get created at the correct time and solve a race today where the upward propagation of the device creation uevent races with the attribute creation and may result in the root device not being found if udev wins the race.

The session wound up with Bottomley demanding that Greg and Kay show exactly what the sysfs people have in store for SCSI.

*For the complete summaries, see <http://www.usenix.org/events/lsf08/lfs08sums.pdf>.*

# writing for *;login:*

Writing is not easy for most of us. Having your writing rejected, for any reason, is no fun at all. The way to get your articles published in *;login:*, with the least effort on your part and on the part of the staff of *;login:*, is to submit a proposal first.

## PROPOSALS

In the world of publishing, writing a proposal is nothing new. If you plan on writing a book, you need to write one chapter, a proposed table of contents, and the proposal itself and send the package to a book publisher. Writing the entire book first is asking for rejection, unless you are a well-known, popular writer.

*;login:* proposals are not like paper submission abstracts. We are not asking you to write a draft of the article as the proposal, but instead to describe the article you wish to write. There are some elements that you will want to include in any proposal:

- What's the topic of the article?
- What type of article is it (case study, tutorial, editorial, mini-paper, etc.)?
- Who is the intended audience (syadmins, programmers, security wonks, network admins, etc.)?
- Why does this article need to be read?

- What, if any, non-text elements (illustrations, code, diagrams, etc.) will be included?
- What is the approximate length of the article?

Start out by answering each of those six questions. In answering the question about length, bear in mind that a page in *;login:* is about 600 words. It is unusual for us to publish a one-page article or one over eight pages in length, but it can happen, and it will, if your article deserves it. We suggest, however, that you try to keep your article between two and five pages, as this matches the attention span of many people.

The answer to the question about why the article needs to be read is the place to wax enthusiastic. We do not want marketing, but your most eloquent explanation of why this article is important to the readership of *;login:*, which is also the membership of USENIX.

## UNACCEPTABLE ARTICLES

*;login:* will not publish certain articles. These include but are not limited to:

- Previously published articles. A piece that has appeared on your own Web server but not been posted to USENET or slashdot is not considered to have been published.
- Marketing pieces of any type. We don't accept articles about products. "Marketing" does not include being enthusiastic about a new tool or software that you can download for free, and you are encouraged to write case

studies of hardware or software that you helped install and configure, as long as you are not affiliated with or paid by the company you are writing about.

- Personal attacks

## FORMAT

The initial reading of your article will be done by people using UNIX systems. Later phases involve Macs, but please send us text/plain formatted documents for the proposal. Send proposals to [login@usenix.org](mailto:login@usenix.org).

## DEADLINES

For our publishing deadlines, including the time you can expect to be asked to read proofs of your article, see the online schedule at <http://www.usenix.org/publications/login/sched.html>.

## COPYRIGHT

You own the copyright to your work and grant USENIX permission to publish it in *;login:* and on the Web. USENIX owns the copyright on the collection that is each issue of *;login:*. You have control over who may reprint your text; financial negotiations are a private matter between you and any reprinter.

## FOCUS ISSUES

In the past, there has been only one focus issue per year, the December Security edition. In the future, each issue may have one or more suggested focuses, tied either to events that will happen soon after *;login:* has been delivered or events that are summarized in that edition.



## Announcement and Call for Papers **USENIX**

# 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

<http://www.usenix.org/nsdi09>

April 22–24, 2009

Boston, MA

### Important Dates

Paper titles and abstracts due: *October 3, 2008, 6:00 p.m. EDT*

Complete paper submissions due: *October 10, 2008, 6:00 p.m. EDT (hard deadline)*

Notification of acceptance: *December 19, 2008*

Papers due for shepherding: *February 2, 2009*

Final papers due: *February 25, 2009*

Poster proposals due: *March 1, 2009*

Notification to poster presenters: *March 15, 2009*

### Conference Organizers

#### Program Co-Chairs

Jennifer Rexford, *Princeton University*

Emin Gün Sirer, *Cornell University*

#### Program Committee

Miguel Castro, *Microsoft Research*

Jeff Dean, *Google, Inc.*

Nick Feamster, *Georgia Institute of Technology*

Michael J. Freedman, *Princeton University*

Steven D. Gribble, *University of Washington*

Krishna Gummadi, *Max Planck Institute for Software Systems*

Steven Hand, *University of Cambridge*

Farnam Jahanian, *University of Michigan*

Dina Katabi, *Massachusetts Institute of Technology*

Arvind Krishnamurthy, *University of Washington*

Bruce Maggs, *Carnegie Mellon University/Akamai*

Petros Maniatis, *Intel Research Berkeley*

Nick McKeown, *Stanford University*

Greg Minshall

Michael Mitzenmacher, *Harvard University*

Jeff Mogul, *HP Labs*

Venugopalan Ramasubramanian, *Microsoft Research*

Pablo Rodriguez, *Spain Telefónica*

Kobus van der Merwe, *AT&T Labs—Research*

Geoffrey M. Voelker, *University of California, San Diego*

Matt Welsh, *Harvard University*

Hui Zhang, *Carnegie Mellon University/Rinera*

Yuanyuan Zhou, *University of Illinois at Urbana-Champaign*

### Steering Committee

Thomas Anderson, *University of Washington*

Greg Minshall

Mike Schroeder, *Microsoft Research*

Margo Seltzer, *Harvard University*

Amin Vahdat, *University of California, San Diego*

Ellie Young, *USENIX*

### Overview

NSDI focuses on the design principles and practical evaluation of large-scale networked and distributed systems. Systems as diverse as Internet routing, peer-to-peer and overlay networks, sensor networks, Web-based systems, and measurement infrastructures share a set of common challenges. Progress in any of these areas requires a deep understanding of how researchers are addressing the challenges of large-scale systems in other contexts. Our goal is to bring together researchers from across the networking and systems community—including communication, distributed systems, and operating systems—to foster a broad approach to addressing our common research challenges.

### Topics

NSDI will provide a high-quality, single-track forum for presenting new results and discussing ideas that overlap these disciplines. We seek a broad variety of work that furthers the knowledge and understanding of the networked systems community as a whole, continues a significant research dialog, or pushes the architectural boundaries of large-scale network services. We solicit papers describing original and previously unpublished research. Specific topics of interest include but are not limited to:

- Self-organizing, autonomous, and federated networked systems
- Scalable techniques for providing high availability and reliability
- Energy-efficient computing in networked environments
- Clean-slate approaches to communication systems
- Distributed storage, caching, and query processing
- Security, robustness, and fault-tolerance in networked environments

- Overlays and peer-to-peer systems
- Systems and protocols for mobile and wireless systems
- Protocols and OS support for sensor networking
- Novel operating system support for networked systems
- Virtualization and resource management for networked systems
- Design and evaluation of large-scale networked system testbeds
- Network measurements, workload, and topology characterization
- Managing, debugging, and diagnosing problems in networked systems
- Practical protocols and algorithms for networked systems
- Addressing novel challenges of the developing world
- Experience with deployed networked systems

## What to Submit

Submissions must be full papers, at most 14 single-spaced 8.5" x 11" pages, including figures, tables, and references, two-column format, using 10-point type on 12-point (single-spaced) leading, with a maximum text block of 6.5" wide x 9" deep with .25" intercolumn space. Papers that do not meet the size and formatting requirements will not be reviewed. Submissions will be judged on originality, significance, interest, clarity, relevance, and correctness.

NSDI is single-blind, meaning that authors should include their names on their paper submissions and do not need to obscure references to their existing work.

Authors must submit their paper's title and abstract by October 3, 2008, and the corresponding full paper is due by October 10, 2008 (hard deadline). All papers must be submitted via the Web form on the Call for Papers Web site, <http://www.usenix.org/nsdi09/cfp>. Accepted papers may be shepherded through an editorial review process by a member of the Program Committee. Based on initial feedback from the Program Committee, authors of shepherded papers will submit an editorial revision of their paper to their Program Committee shepherd by February 2, 2009. The shepherd will review the paper and give the author additional comments. All authors (shepherded or not) will produce a final, printable PDF and the equivalent HTML by February 25, 2009, for the conference Proceedings.

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs

and journal editors to ensure the integrity of papers under consideration.

Previous publication at a workshop is acceptable as long as the NSDI submission includes substantial new material. For instance, submitting a paper that provides a full evaluation of an idea that was previously sketched in a 5-page position paper is acceptable. Authors of such papers should cite the prior workshop paper and clearly state the submission's contribution relative to the prior workshop publication.

Authors uncertain whether their submission meets USENIX's guidelines should contact the Program Co-Chairs, [nsdi09chairs@usenix.org](mailto:nsdi09chairs@usenix.org), or the USENIX office, [submissionspolicy@usenix.org](mailto:submissionspolicy@usenix.org).

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions will be treated as confidential prior to publication on the USENIX NSDI '09 Web site, <http://www.usenix.org/nsdi09>.

One author per paper will receive a registration discount of \$200. USENIX will offer a complimentary registration upon request.

## Best Paper Awards

Awards will be given for the best paper and the best paper for which a student is the lead author.

## Birds-of-a-Feather Sessions

Birds-of-a-Feather sessions (BoFs) are informal gatherings organized by attendees interested in a particular topic. BoFs will be held in the evening. BoFs may be scheduled in advance by emailing the USENIX Conference Department at [bofs@usenix.org](mailto:bofs@usenix.org). BoFs may also be scheduled at the conference.

## Poster Session

NSDI will be continuing its long-running tradition of showcasing early research in progress at a poster session. New, ongoing work, early findings from measurement studies, and demonstrations of newly deployed systems are highly encouraged. We are particularly interested in presentations of student work. To submit a poster, please send a proposal, one page or less, by March 1, 2009, to [nsdi09posters@usenix.org](mailto:nsdi09posters@usenix.org). The poster session chairs will send back decisions by March 15, 2009.

## Registration Materials

Complete program and registration information will be available in January 2009 on the conference Web site. If you would like to receive the latest USENIX conference information, please join our mailing list at <http://www.usenix.org/about/mailling.html>.



## Announcement and Call for Papers **USENIX**

# 12th Workshop on Hot Topics in Operating Systems (HotOS XII)

Sponsored by USENIX, the Advanced Computing Systems Association, in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

<http://www.usenix.org/hotos09>

**May 18–20, 2009**

**Monte Verità, Switzerland**

### Important Dates

Paper submissions due: *January 13, 2009*  
(hard deadline)

Notification to authors: *March 10, 2009*

Final papers due: *April 20, 2009*

Papers available online for attendees: *April 27, 2009*

### Workshop Organizers

#### Program Chair

Armando Fox, *University of California, Berkeley*

#### Program Committee

George Candea, *EPFL*

Garth Gibson, *Carnegie Mellon University and  
Panasas, Inc.*

Rebecca Isaacs, *Microsoft Research*

Kimberly Keeton, *Hewlett-Packard Labs*

Eddie Kohler, *University of California, Los Angeles*

Petros Maniatis, *Intel Research Berkeley*

Timothy Roscoe, *ETH Zürich*

Michael L. Scott, *University of Rochester*

Marvin Theimer, *Google, Inc.*

Amin Vahdat, *University of California, San Diego*

Dan S. Wallach, *Rice University*

### Overview

The practice of computing continues to move at astonishing speed. In the past few years alone, we've seen cloud computing and software as a service, containerized computing, multicore/manycore becoming mainstream, batch processing of petabyte datasets, and biological and statistical approaches to computing and systems. The 12th Workshop on Hot Topics in Operating Systems will bring together innovative practitioners and researchers in computing systems, broadly construed. Continuing the HotOS tradition, participants will present and discuss new ideas about computer systems research and how technological advances and new

applications are shaping our computational infrastructure.

We solicit position papers of **five or fewer pages** that propose new directions of research, advocate non-traditional approaches, report on noteworthy actual experience in an emerging area, or generate lively discussion around an important topic. HotOS takes a broad view of systems, including operating systems, storage, networking, languages and language engineering, security, dependability, and manageability. We are also interested in contributions influenced by other fields such as hardware design, machine learning, control theory, networking, economics, social organizations, and biological or other nonsilicon computing systems.

To ensure a vigorous workshop environment, attendance is limited to about 60 participants who are active in the field. Participants will be invited based on their submissions' originality, technical merit, topical relevance, and likelihood of leading to insightful technical discussions that will influence future systems research. Submissions may not be under consideration for any other venue. In order to promote discussion, the review process will heavily favor submissions that are forward-looking and open-ended, as opposed to those that summarize more mature work on the verge of conference publication. In general, at most two authors per accepted paper will be invited to the workshop.

### Submitting a Paper

Position papers must be received by 11:59 p.m. PST on January 13, 2009. **This is a hard deadline—no extensions will be granted.**

Submissions must be no longer than 5 pages including figures, tables, and references. Text should be formatted in two columns on 8.5-inch by 11-inch paper using 10 point fonts on 12 point (single-spaced) leading, and 1-inch margins. Author names and affiliations should appear on the title page (reviewing is not blind).

Pages should be numbered, and figures and tables should be legible in black and white without requiring magnification. Papers not meeting these criteria will be rejected without review, and no deadline extensions will be granted for reformatting.

Papers must be submitted in PDF format via a Web submission form on the HotOS XII Call for Papers Web site, <http://www.usenix.org/hotos09/cfp>.

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about sub-

mitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair, [hotos09chair@usenix.org](mailto:hotos09chair@usenix.org), or the USENIX office, [submissionspolicy@usenix.org](mailto:submissionspolicy@usenix.org).

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions will be treated as confidential prior to publication on the USENIX Web site.



There's a whole lot of technology in the queue. are you ready?

What's next?



Get ready with **ACM Queue**—the technology magazine focused on problems that don't have easy answers—yet.

**Queue** dissects the challenges of emerging technologies. **Queue** targets the problems and pitfalls just ahead. **Queue** helps you plan for the future. **Queue** poses the hard questions you'd like to ask.

Isn't that what you've been looking for?

[www.acmqueue.org](http://www.acmqueue.org)

Subscribe now at **ACM Queue's** special, limited-time charter subscription rate of \$19.95 for ACM members. Use the subscription card in this issue or go to the **ACM Queue** web site at [www.acmqueue.org](http://www.acmqueue.org)

[www.acmqueue.org](http://www.acmqueue.org)

# LINUX+

DVD

The best source for Linux users

[www.linuxmagazine.org/en](http://www.linuxmagazine.org/en)

Check it out at the nearest  
Barnes&Noble and Borders stores!

# WANTED

2 DVDs CentOS Linux 5.0 Fedora 7.0 Open Office 2.2.1

## LINUX+ LiNux+

LINUX ENVIRONMENT FOR EXPERTS

**EXPLORE CENTOS**  
Install & Configure Step-by-Step

**FEDORA 7.0**  
Closer Look at RedHat Distributions

Using IP Tables  
How to use Netfilter's logging capabilities

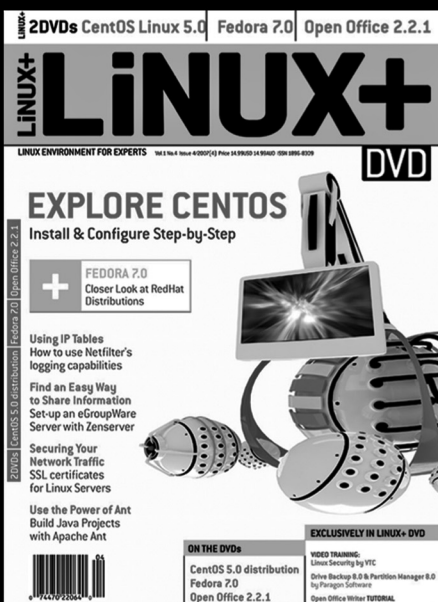
Find an Easy Way to Share Information  
Set-up an eGroupWare Server with Zenserver

Securing Your Network Traffic  
SSL certificates for Linux Servers

Use the Power of Ant  
Build Java Projects with Apache Ant

**ON THE DVD's**  
CentOS 5.0 distribution  
Fedora 7.0  
Open Office 2.2.1

**EXCLUSIVELY IN LINUX+ DVD**  
VIDEO TRAINING:  
Linux Security by YTC  
Drive Backup 8.0 & Partition Manager 8.0  
by Paragon Software  
Open Office Writer TUTORIAL



2 DVDs Mandriva One 2008 & Extras

## LINUX+ LiNux+

LINUX ENVIRONMENT FOR EXPERTS

**Mandriva Uncovered**  
Install & Configure - Ins and Outs

**GREAT multimediaBOX SECTION**  
LINUX WORLD OF GAMES

IP Spoofing Attacks  
How to prevent IP spoofing attacks

Kernel Security  
Additional level of Linux security

Amanda-Backup Solution  
How to keep your files safe

PCLinuxOS  
Linux hard to use?  
Find out the truth

**ON THE DVD's**  
MANDRIVA ONE 2008  
HOW TO USE LINUX FROM THE  
MENUING "GUIDE"

**PLUS**  
BY OFFERS SECURITY  
FOR MAIL SERVICES  
GENERAL SERVER 8.0



2 DVDs Debian 4.0 plus Extras Filezilla 3.0.4

## LINUX+ LiNux+

LINUX ENVIRONMENT FOR EXPERTS

**HANDS-ON WITH DEBIAN**  
Installation & Configuration Guide

**GRAPHICS CARDS - CONSUMERS' CHOICE**  
CHOOSE THE BEST CARD FOR LINUX

Linux video drivers  
Which driver is right for you

Debian Package Management  
How to keep your system safe & up-to-date

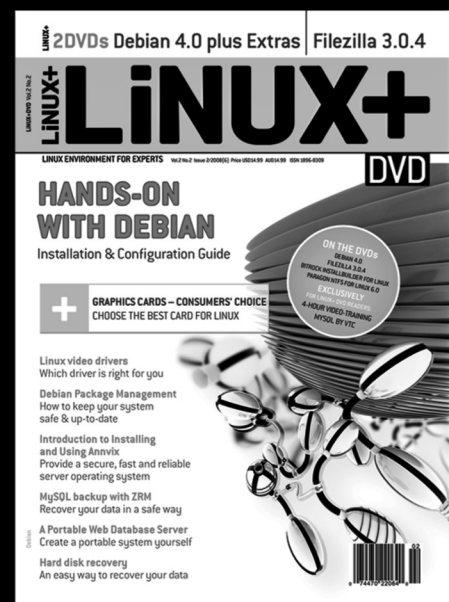
Introduction to Installing and Using Annniv  
Provide a secure, fast and reliable server operating system

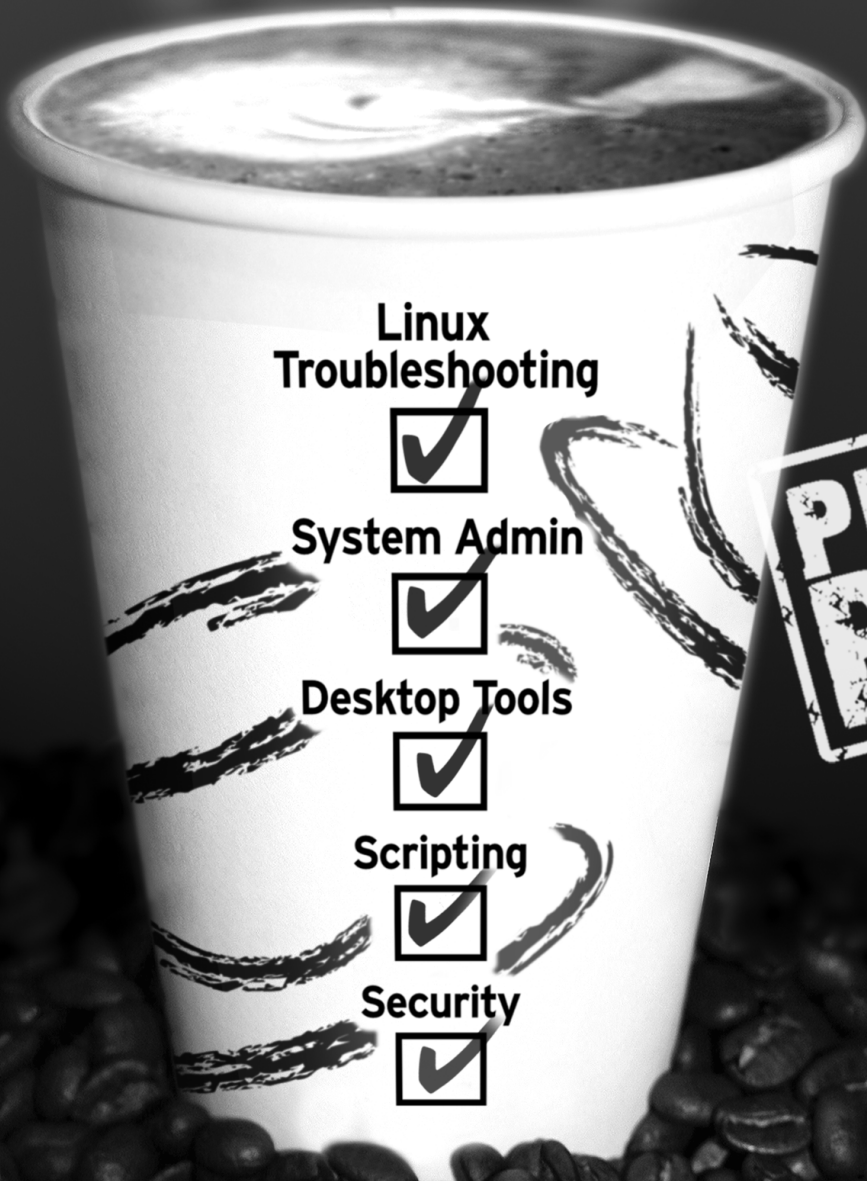
MySQL backup with ZRM  
Recover your data in a safe way

A Portable Web Database Server  
Create a portable system yourself

Hard disk recovery  
An easy way to recover your data

**ON THE DVD's**  
DEBIAN 4.0  
BRIDGE NETWORKING FOR LINUX  
EXCLUSIVELY  
HANDICRAFTED FOR LINUX 4.0  
4-HOUR VIDEO TRAINING  
MYSQL BY YTC





Linux  
Troubleshooting



System Admin



Desktop Tools



Scripting



Security



**PREMIUM  
BLEND**

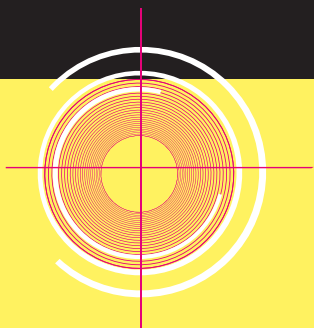
**LINUX** PRO  
MAGAZINE

If you like the taste  
of Linux, why not treat  
yourself to the best?

Linux Pro Magazine delivers real-world solutions for the technical reader. In every issue, you'll find advanced techniques for configuring and securing Linux systems. Learn about the latest tools and discover the secrets of the experts in Linux Pro. Each issue includes a full Linux distribution on DVD.

[www.linuxpromagazine.com](http://www.linuxpromagazine.com)

*Save the Date!*



# 17th USENIX SECURITY SYMPOSIUM

July 28—August 1, 2008

## Join us in San Jose, CA, for:

- Keynote address, “Dr. Strangevote or: How I Learned to Stop Worrying and Love the Paper Ballot,” by Debra Bowen, California Secretary of State
- 2 days of training by industry experts
- Invited talkss by leaders in security
- Refereed papers covering the latest research
- Plus BoFs, a poster session, WiPs, and more!

## Don't miss these co-located workshops:

2008 USENIX/ACCURATE Electronic Voting Technology Workshop  
July 28–29, 2008  
<http://www.usenix.org/evt08/>

Workshop on Cyber Security Experimentation and Test  
July 28, 2008  
<http://www.usenix.org/cset08/>

**Register by July 14, 2008, and save!**

**<http://www.usenix.org/sec08/jlo>**

## **;login:**

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

---