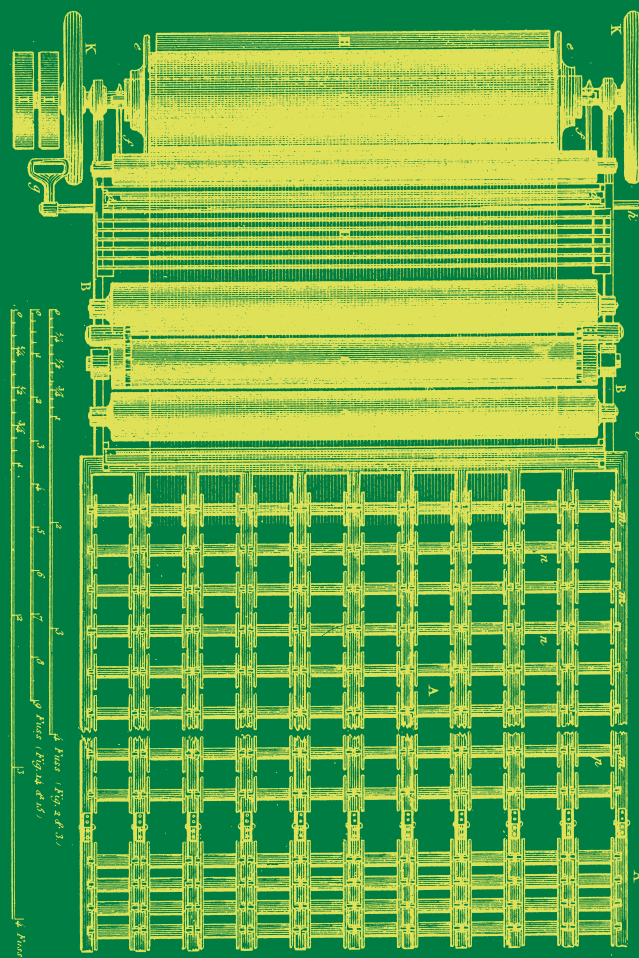
JUNE 2007 VOLUME 32 NUMBER 3

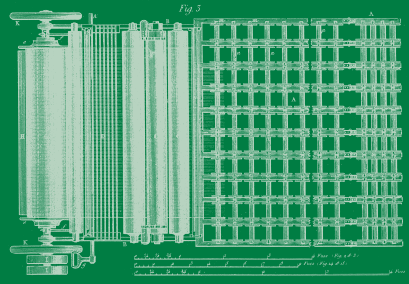
;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events



5th ACM/USENIX International Conference on Mobile Computing Systems, Applications, and Services (MobiSys 2007)

Jointly sponsored by USENIX and ACM SIGMOBILE, in cooperation with ACM SIGOPS

JUNE 11–15, 2007, San Juan, Puerto Rico
http://www.sigmobile.org/mobisys/2007/

Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '07)

Sponsored by ACM SIGPLAN and ACM SIGOPS in cooperation with USENIX

JUNE 13–15, 2007, San Diego, CA, USA
http://vee07.cs.ucsb.edu

Second Workshop on Hot Topics in Autonomic Computing (HotAC II)

Sponsored by IEEE in cooperation with USENIX and ACM

JUNE 15, 2007, Jacksonville, FL, USA
http://www.aqualab.cs.northwestern.edu/HotACII/

2007 USENIX Annual Technical Conference

JUNE 17–22, 2007, Santa Clara, CA, USA
http://www.usenix.org/usenix07

3rd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '07)

Co-located with the 2007 USENIX Annual Technical Conference

JUNE 18, 2007, Santa Clara, CA, USA
http://www.usenix.org/sruti07

Inaugural International Conference on Distributed Event-Based Systems (DEBS 2007)

Organized in cooperation with USENIX, the IEEE and IEEE Computer Society, and ACM (SIGSOFT)

JUNE 20–22, 2007, Toronto, Canada
http://www.debs.msrg.utoronto.ca

Third Workshop on Hot Topics in System Dependability (HotDep '07)

Co-sponsored by USENIX

JUNE 26, 2007, Edinburgh, UK
http://hotdep.org/2007

USENIX Training at SANSFIRE 2007

JULY 25–AUGUST 3, 2007, Washington, D.C., USA
http://www.usenix.org/sansfire07

16th USENIX Security Symposium

AUGUST 6–10, 2007, Boston, MA, USA
http://www.usenix.org/sec07

2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)

Co-located with USENIX Security '07

AUGUST 6, 2007, Boston, MA, USA
http://www.usenix.org/evt07

First USENIX Workshop on Offensive Technologies (WOOT '07)

Co-located with USENIX Security '07

AUGUST 6, 2007, Boston, MA, USA
http://www.usenix.org/woot07

DETER Community Workshop on Cyber Security Experimentation and Test 2007

Co-located with USENIX Security '07

AUGUST 6–7, 2007, Boston, MA, USA
http://www.usenix.org/deter07

2nd USENIX Workshop on Hot Topics in Security (HotSec '07)

Co-located with USENIX Security '07

AUGUST 7, 2007, Boston, MA, USA
http://www.usenix.org/hotsec07

Second Workshop on Security Metrics (MetriCon 2.0)

Co-located with USENIX Security '07

AUGUST 7, 2007, Boston, MA, USA
http://www.securitymetrics.org

21st Large Installation System Administration Conference (LISA '07)

Sponsored by USENIX and SAGE

NOVEMBER 11–16, 2007, Dallas, TX, USA
http://www.usenix.org/lisa07

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

contents

;login:

VOL. 32, #3, JUNE 2007

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
David Couzens
proofshop@usenix.org

PRODUCTION
Lisa Camp de Avalos
Casey Henderson

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.

$85 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.

RIK FARROW

musings

rik@usenix.org

NOT EVERYONE WILL BE AS FASCI- nated by hardware as I am. But some people certainly are, so I will feed those who want to know more about the physical parts of the computers that, in the end, feed those of us who work with them.

I got to attend the tutorial given by Dave Anderson and Willis Whittington, both longtimers at Seagate Technologies. Anderson and Whittington shared what they could, that is, that which is not proprietary and secret, before the 5th USENIX Conference on File and Storage Technologies began in San Jose in February 2007. Later on, I will share a different perspective with you, as researchers presented two papers about disk failure rates that conflict with what drive vendors report. But for now, let me present a digest of what I learned.

First and most obvious, Anderson and Whittington speak from the manufacturer's perspective. Before you go mentally disregarding everything they say, you need to realize that they live in the world where shiny new drives get made, drives that people expect will have high capacities, high I/O rates, and low error rates and will last at least five years while costing as little as possible. This list of expectations is self-conflicting to start with. And, from a vendor's perspective, testing how long drives will survive is actually impossible, outside of field tests, at which point, said drives will be obsolete by several years.

Drive vendors see their market split into many categories, certainly more than I had considered: enterprise, near-line enterprise, home PC, notebook, and consumer devices. As a computer user and researcher, I find myself focused on just three of these: the enterprise drives, SCSI, FC, and SAS; near-line, SATA and FC; and PC, SATA. The old PC standard, ATA, is now called PATA, for Parallel ATA, and is expected to disappear, with the exception of replacement drives, very soon.

The consumer market for drives is the most volatile, with price and capacity being the driving factors. Vendors view the enterprise market differently, with reliability and high I/O rate being critical. Note that these categories were not created by drive vendors but are driven by the demands of the two biggest purchasers of hard drives: the makers of high-end servers and storage systems. If your entire business revolves around providing fast and reliable storage systems (EMC and Network Appliance as examples), then the behavior of the millions of drives you use each year influences buyer perception of your own servers.

Anderson and Whittington didn't spend much time explaining where the demand for enterprise drives comes from. I just wanted to make that clear myself, as enterprise drives are designed and manufactured to suit the needs of special classes of users. Those differences do show up as they talk about the hardware, so knowing the reason for enterprise drives helps to make sense of why enterprise drives have lower capacities, have higher I/O rates, and cost more.

Speaking of drive capacities, the areal density—the product of bits per inch times tracks per inch—is the key factor. Bits per inch (BPI) means the number of bits that can be written and later reread per linear inch. You might first think that BPI just has to do with the magnetic coating on a surface, but you would be wrong. The magnetic coating is just one of six layers, starting with the substratum, which can be aluminum or glass (used in notebook drives for its greater rigidity), and ending with a lubrication layer. The number of magnetic grains is a limiting factor for BPI, and one that gets attacked by creating smaller grains, all of nearly the same size. In the future, the magnetic coating will likely be composed of self-organized particles in the 6.3 +/- 0.3 nm range. The current particle sizes range from 8 to 15 nm.

Head technology represents another limiting factor for BPI. The head flies over the surface of a disk at about the distance of a wavelength of visible light (about 0.5 micrometers) and as fast as 118 miles per hour (in 15k rpm drives). Heads must be fabricated to exacting standards to read and write the tiny magnetic regions on narrow tracks. In the most recent advance in head technology, called perpendicular recording, the write field penetrates the media at a right angle, instead of along the surface of the media (longitudinal recording). The read portion of the head now uses GMR, Giant Magnetoresistive effect, to sense the magnetic orientation of bits. At current bit densities, 80–100 grains make up one bit.

The number of tracks that can fit within an inch is governed by head positioning, runout, and rotational vibration (RV). Head positioning is the easiest to grasp, but it requires great precision when there can be over 90,000 tracks per inch. Runout describes the shape of a track, which is never quite round. So following a track long enough to read a sector not only means seeking to the correct track but also following the track, because it does not describe a circle.

As if this feat weren't difficult enough, RV indicates the amount of vibration, created by other hard drives as they seek, by fans, and by other sources of vibration. Consider that if you have a single hard drive, each time it seeks, its case (and thus its mounting hardware) must resist the angular momentum created by swiveling the head. Now, put a bunch of hard drives into one unit, then stack many of those units up in a rack, and imagine all of the shaking going on, all in the same approximately horizontal plane within which the drives are rotating their heads. Anderson described tests of drive cabinets where one-third of the cabinets tested allowed an unacceptable level of RV for any type of drive. The more rigid the drive mounts, where metal is good and plastic bad, the better the cabinet.

This is also one of the areas where enterprise drives differ from other drives. Enterprise drives have two accelerometers, each sensing movement about the drive axis, and one drive CPU (enterprise drives have two) works to compensate for RV, keeping the head on track. All drives have positioning information created when the drives are formatted, and this information is used to keep the heads aligned with the track too. But enterprise drives can recover from more RV (21 rad/s^2) than SATA drives

(just 6 rad/s^2). If the drive fails to follow a track, there will be a read error, and the drive CPU will reattempt the read. The goal is for enterprise drives to have higher I/O rates in environments with lots of activity by avoiding having to reread sectors.

Reading the data from sectors also differs between enterprise and other drives. All drives include Error Correction Code (ECC) that allows the drive to recover from bit errors while reading data. Enterprise drives also include Error Detection Code (and this is not the CRC that's included when data is sent to the drive) and an additional IOECC that makes it possible for enterprise drives to recover from more bit errors than other drives. Enterprise drives also use additional sync marks within the data field, instead of just at the beginning of the data field, as in other drives. All of these techniques subtract from the amount of space left for data in exchange for more reliability.

I've already mentioned that future development of disk drives will require a smaller grain size and more even distribution, to increase the areal density. Another future technique will be the development of write heads that include a small laser that heats the grains before the perpendicular write head passes over them. With this Heat Assisted Magnetic Recording (HAMR), consumer drives are expected to reach capacities of 8 TB by 2013, and enterprise drives 2.4 TB.

And what about increasing disk rpms? Today, only enterprise drives spin at 15,000 rpm, with consumer drives running at 7,200 rpm and notebook drives at 5,400 rpm. Increasing the rpms increases the IO transfer rate, as more bits pass under the head in each second the faster the disks spin. But the power required to rotate a disk increases as the cube of the rpm. The roadmap does have consumer drives reaching 10,000 rpm and enterprise drives staying at 15,000 rpm. Because areal density will be increasing as well as rpms, consumer drives should reach a maximum transfer rate of 5 GB/s by 2013, but enterprise drives will actually be slower, at 4 GB/s (because their areal density is lower).

Enterprise read seek times are already about half that of consumer drives (which have seek times of 8 ms, compared to 3.7 ms for enterprise drives), and this ratio will remain about the same, with only modest improvements in read seek speed (seek times of 6.5 ms to 2.8 ms projected for 2013).

There was, of course, much more in this half-day class, and if you ever get a chance to listen to either Anderson or Whittington speak, I'd recommend that you be there if you find yourself fascinated by the details of modern disk drives.

The Competition

Not only do disk vendors compete with each other, they now find themselves competing with memory vendors as well. Various forms of flash memory have improved in speed, capacity, and number of rewrites while offering lower cost, and you can already buy flash "drives" for laptops, as well as the now ubiquitous USB memory sticks, at prices that compare well with hard drives ($10/GB for flash versus $1/GB or less for disks). As an interesting side note, IBM developed the first form of rotating magnetic memory, which cost $10,000/MB (in 1956 US dollars; perhaps $70,000/MB in today's dollars). That makes my first hard drive, at $60/MB, or $2000 for a 34-MB drive, not seem quite so outrageous.

The competition to disk vendors that I really want to address is not other hardware vendors, but researchers. During the first session at FAST '07,

two groups presented papers in which they examined hard-drive replacement rates based on field data. Disk vendors perform accelerated aging tests on the drives they build by subjecting the drives to high temperatures and high utilization (continuous IO with lots of seeks) in an attempt to tease out how long a particular drive type will last. The vendors publish the Annual Failure Rate, AFR, based on these tests. In these two papers, the researchers report actual failure rates several times higher than the ones suggested by vendors.

Bianca Schroeder (with Garth Gibson, winner of the Best Paper Award) collected information about disk failures from several High Performance Computing (HPC) centers as well as a couple of Internet service providers. Although the information collected from each of the sources differed in many ways, she statistically analyzed the data to pry out a number of interesting observations. For example, the expected rate of failures for disk drives is supposed to resemble a curve like a bathtub, with high failure rates at the beginning of drive life as well as toward its end. In Schroeder's analysis, the failure rate, which she termed ARR for Annual Replacement Rate, was highest in the third and fourth years, placing a big hump where there is supposed to be a comfortable dip in the replacement graph.

Schroeder's paper lists many observations, and I suggest you read her paper for all of them. I do want to mention another point that you need to be aware of: the possibility of a drive failing while a RAID system is in the process of rebuilding the replacement drive. The standard (and vendor) view of this process is based on the Unrecoverable Error Rate (UER), something that Anderson and Whittington discussed in their tutorial. Enterprise drives have a lower UER, 10^{-16}, compared to SATA drives, 10^{-14}. To rebuild one disk in a RAID 5 array composed of 500-GB SATA drives, 2^{13} bits must be read successfully, one-fifth the value of the UER for SATA drives. In other words, the odds of encountering a second error while rebuilding this RAID 5 array are 1 in 5. For people who are counting on RAID for reliable access to data, a 20% chance of failure is much too high.

Although this potential for failure already appears high, Schroeder shows that it fits poorly with observed data. First, just consider this quote from Schroeder and Gibson:

> The failure probability of disks depends for example on many factors, such as environmental factors, like temperature, that are shared by all disks in the system. When the temperature in a machine room is far outside nominal values, all disks in the room experience a higher than normal probability of failure.

I think we can agree that this makes good, intuitive sense, partially shredding the notion of relying on UER for calculating risk without looking at real data. Then Schroeder goes on to test for autocorrelation: the notion that disk failures appear to be related in time. If one just considers UER, failures should be completely random and unrelated. Schroeder shows that, in practice, disk failures appear related, exhibiting a decreasing hazard rate over time. A decreasing hazard rate implies that a subsequent disk failure is likely to occur sooner, rather than later. So the likelihood of a second disk failure while rebuilding a RAID 5 array appears much higher than a simple UER suggests.

I have felt uncomfortable when I hear or read about people relying on RAID systems with no backups. All I had to rely on was the UER, which seemed dangerous enough when applied to large arrays. But Schroeder's work makes relying on RAID without a backup appear more like expecting

lightning not to strike in the same place twice, even if the spot in question is the antenna on top of a very tall building. RAID arrays are composed of drives all within the same environment, likely the exact same type of drive manufactured perhaps within the same batch.

Schroeder also observes that she did not find any difference in the failure rates for SATA and SCSI drives in her data. One of the points in building, or buying, enterprise drives is to gain a higher level of reliability, but the data in this case do not back up that goal. I find this point very interesting, as both disk and file server manufacturers appear to believe in enterprise drives, and I suspect they have reasons that go beyond the higher profit margins in enterprise drives.

Google Drives

Schroeder and Gibson weren't the only people looking at drive failures at FAST '07. Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso of Google Inc. wrote "Failure Trends in a Large Disk Drive Population," which examines another very large data set about disk drive failures. Google is known to collect huge amounts of data on vast distributed storage systems. Just how large is actually a secret, and Pinheiro et al. don't tell us exactly how many drives, but they do tell us that they are looking at "more than one hundred thousand," a decent-sized data set.

This paper shares one of the problems faced by Schroeder and Gibson: The exact failure time of drives is often unknown. Pinheiro et al. use the time of replacement of drives as their failure metric, and the reason for failure was not included in their data. The focus of this paper is quite different in that they examined SMART (Self-Monitoring, Analysis and Reporting Technology) data collected from disk drives to see if this data could be used to predict drive failures.

SMART has always seemed like a good idea to me. Modern disk drives are embedded systems, and having the drive expose some of the data it collects makes perfect sense. But I can't say that I have routinely run the SMART data collection tools, as I've experienced plenty of disk failures, usually at the most inappropriate times and without any useful warnings. Pinheiro reports, sadly enough, similar findings.

Like Schroeder, Pinheiro reports an AFR that is not at all bathtub-shaped, with the same big hump in the middle. The Google data actually shows peaks earlier than Schroeder's data, at years two and three. Google data also includes a utilization metric, missing from the other paper. The expected result would be that heavily utilized drives, particularly the SATA drives favored by Google, would fail more frequently. In fact, their data show no difference between heavily and lightly utilized drives, except in the first three months of use and during the fifth year. The authors suggest that the spike in failures in heavily utilized drives represents

> the survival-of-the-fittest theory. It is possible that the failure modes that are associated with higher utilization are more prominent early in the drive's lifetime. If that is the case, the drives that survive the infant mortality phase are the least susceptible to that failure mode, and the resulting population is more robust with respect to variations in utilization levels.

This finding might also account for what disk vendors discover when they stress-test new drives. However, Pinheiro et al. did not perceive any significant difference in the rate of drive failures related to higher temperatures.

Pinheiro et al. looked at four SMART data variables to see whether they can predict drive failure. Scan errors (when the drive detects an error while performing reads during background testing) do indicate that these drives are 39 times more likely to fail within 60 days than drives with no scan errors. Reallocation errors (when a drive remaps a sector because of repeated soft read errors or a hard read error) also indicate that a drive may be likely to fail sooner than drives with no errors (being 14 times more likely to fail within 60 days).

Pinheiro et al. examine seven other SMART parameters, then attempt to create predictive models for drive failures. Since some SMART data appears highly correlated with drive failures, they hoped they could create a predictive failure model. Unfortunately, using SMART data, with and without temperature values, still left 36% of all replaced drives with no failure signals at all. The authors conclude that SMART data is useful for provisioning, as it can predict the aggregate reliability of large disk populations, but it cannot suggest when an individual drive is about to die. Too bad.

Intelligent Drives

Disk drives have been getting "smarter" for many years. I did ask Dave Anderson whether programmers should make any assumptions about the relationship between logical disk layout and the physical disk. Anderson told me that we should forget any notion of "cylinders"; as for disk layout, he implied that when writing a collection of logically sequential blocks a disk would attempt to write those blocks so that they could be read again quickly. In other words, what happens inside the physical disk may be quite different from what we expect. I asked people involved with Linux and BSD filesystem design whether they knew about this, and both said they were quite aware that disks, not the filesystem designer, have control over where data gets written. I was a bit amazed, even though I had heard stories about this. Guess I am just a bit out of date.

Given that disk drives have gotten a lot smarter, perhaps it makes sense to share more responsibility for file systems. I believe that day is coming, and you will see research in other FAST papers that considers object data storage, making the disk aware of file metadata, and other newfangled notions. Take a look at the FAST summaries included in this issue for more new ideas about file systems.

I also suggest you read the filesystem articles included in this issue of *;login:*. Kirk McKusick leads off with an excellent survey of UNIX filesystem design since 1980, a must for anyone who wants to understand modern file systems. Pawel Jakub Dawidek writes a related but much more focused article about porting ZFS to FreeBSD. You can learn a lot more about ZFS as well as modern OS support for new filesystem designs by reading Dawidek's article. To provide a bit of balance, the lead authors of ext4, the newest version of the Linux ext file system lineage, explain motivations behind creating a new filesystem type, as well as the advantages they have seen in performance and capabilities in this new design. I had expected to have an article on XFS as well, but that will have to wait for another time.

We have two articles about security this month. Dan Geer has been studying security metrics for years now, and he has created a talk that examines the future of security based on current trends. If you want to have a feel for current threats and get a better idea of the security threats you can expect to be facing over the next several years, I invite you to study Geer's observations. Also in the security section, Vassilis Prevelakis demonstrates

how you can use VMware and VMs to simulate both local and routed networks for security classes.

In the Legal section, Dan Appelman concludes his two-part series on spam, blogs, U.S. law, and the system administrator. Appelman provides advice that can be followed by diligent system administrators, whether or not they work in the United States. Alexander Muentz follows Appelman, with a different way of looking at search warrants, subpoenas, and other forms of legal demands. Muentz compares these demands to a DoS attack and suggests both how to prepare for potential demands and how to handle them.

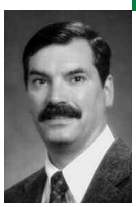
Two regular columnists opted not to submit columns for this issue, but David Blank-Edelman did decide that we needed more entertainment when learning Perl and the Acme module. Before the book review section, packed as usual, Robert Ferrell exercises his development skills with his very own filesystem design.

I've already mentioned that we have FAST summaries, but we also have the summaries from the Linux Storage & Filesystem Workshop. As you might imagine, the workshop and FAST sparked my imagination to create this longer than usual Musings—some things just fire me up.

A while back, I wrote in "Musings" that I didn't yet feel as though I was living in the future. I was referring to the future I had seen in images when I was growing up, with satellite dishes everywhere and flying cars. Since the time I wrote that column, I've acquired a microwave dish on my roof, solar panels, and a hybrid car and can claim to feel vague stirrings of the future around me. But I still run insecure operating systems, have disk drives I can't trust (but am willing to back up), and carry both a cell phone and a laptop when I travel. My own vision of the future includes more than just all-electric vehicles: it also includes a computing device I carry with me everywhere that provides secure storage, networking, and identification. Server systems, too, need to be more reliable, more secure, and easier to manage. We still have a long way to go, with lots of interesting work ahead for enterprising computer scientists.

MARSHALL KIRK MCKUSICK

a brief history of the BSD Fast File System



Dr. Marshall Kirk McKusick writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues, particularly those related to operating systems and file systems. While at the University of California at Berkeley, he implemented the 4.2BSD Fast File System and was the Research Computer Scientist at the Berkeley Computer Systems Research Group (CSRG) overseeing the development and release of 4.3BSD and 4.4BSD.

mckusick@mckusick.com

I FIRST STARTED WORKING ON THE UNIX file system with Bill Joy in the late 1970s. I wrote the Fast File System, now called UFS, in the early 1980s. In this article, I have written a survey of the work that I and others have done to improve the BSD file systems. Much of this research has been incorporated into other file systems.

1979: Early Filesystem Work

The first work on the UNIX file system at Berkeley attempted to improve both the reliability and the throughput of the file system. The developers improved reliability by staging modifications to critical filesystem information so that the modifications could be either completed or repaired cleanly by a program after a crash [14]. Doubling the block size of the file system improved the performance of the 4.0BSD file system by a factor of more than 2 when compared with the 3BSD file system. This doubling caused each disk transfer to access twice as many data blocks and eliminated the need for indirect blocks for many files.

The performance improvement in the 3BSD file system gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the 3BSD file system was still using only about 4% of the maximum disk throughput. The main problem was that the order of blocks on the free list quickly became scrambled as files were created and removed. Eventually, the free-list order became entirely random, causing files to have their blocks allocated randomly over the disk. This randomness forced a seek before every block access. Although the 3BSD file system provided transfer rates of up to 175 kbytes per second when it was first created, the scrambling of the free list caused this rate to deteriorate to an average of 30 kbytes per second after a few weeks of moderate use. There was no way of restoring the performance of a 3BSD file system except to recreate the system.

1982: Birth of the Fast File System

The first version of the current BSD file system was written in 1982 and became widely distributed in 4.2BSD [13]. This version is still in use today on systems such as Solaris and Darwin. For

large blocks to be used without significant waste, small files must be stored more efficiently. To increase space efficiency, the file system allows the division of a single filesystem block into fragments. The fragment size is specified at the time that the file system is created; each filesystem block optionally can be broken into two, four, or eight fragments, each of which is addressable. The lower bound on the fragment size is constrained by the disk-sector size, which is typically 512 bytes. As disk space in the early 1980s was expensive and limited in size, the file system was initially deployed with a default blocksize of 4 kbytes so that small files could be stored in a single 512-byte sector.

1986: Dropping Disk-Geometry Calculations

The BSD filesystem organization divides a disk partition into one or more areas, each of which is called a cylinder group. Historically, a cylinder group comprised one or more consecutive cylinders on a disk. Although the file system still uses the same data structure to describe cylinder groups, the practical definition of them has changed. When the file system was first designed, it could get an accurate view of the disk geometry, including the cylinder and track boundaries, and could accurately compute the rotational location of every sector. By 1986, disks were hiding this information, providing fictitious numbers of blocks per track, tracks per cylinder, and cylinders per disk. Indeed, in modern RAID arrays, the "disk" that is presented to the file system may really be composed from a collection of disks in the RAID array.

Although some research has been done to figure out the true geometry of a disk [5, 10, 23], the complexity of using such information effectively is high. Modern disks have greater numbers of sectors per track on the outer part of the disk than on the inner part, which makes calculation of the rotational position of any given sector complex to calculate. So in 1986, all the rotational layout code was deprecated in favor of laying out files using numerically close block numbers (sequential being viewed as optimal), with the expectation that this would give the best performance. Although the cylinder group structure is retained, it is used only as a convenient way to manage logically close groups of blocks.

1987: Filesystem Stacking

The early vnode interface was simply an object-oriented interface to an underlying file system. By 1987, demand had grown for new filesystem features. It became desirable to find ways of providing them without having to modify the existing and stable filesystem code. One approach is to provide a mechanism for stacking several file systems on top of one another [22B]. The stacking ideas were refined and implemented in the 4.4BSD system [7]. The bottom of a vnode stack tends to be a disk-based file system, whereas the layers used above it typically transform their arguments and pass on those arguments to a lower layer.

Stacking uses the mount command to create new layers. The mount command pushes a new layer onto a vnode stack; a umount command removes a layer. Like the mounting of a file system, a vnode stack is visible to all processes running on the system. The mount command identifies the underlying layer in the stack, creates the new layer, and attaches that layer into the filesystem name space. The new layer can be attached to the same place as the old layer (covering the old layer) or to a different place in the tree (allowing both layers to be visible).

When a file access (e.g., an open, read, stat, or close) occurs to a vnode in the stack, that vnode has several options:

- Do the requested operations and return a result.
- Pass the operation without change to the next-lower vnode on the stack. When the operation returns from the lower vnode, it may modify the results or simply return them.
- Modify the operands provided with the request and then pass it to the next-lower vnode. When the operation returns from the lower vnode, it may modify the results or simply return them.

If an operation is passed to the bottom of the stack without any layer taking action on it, then the interface will return the error "operation not supported."

The simplest filesystem layer is *nullfs*. It makes no transformations on its arguments, simply passing through all requests that it receives and returning all results that it gets back. Although it provides no useful functionality if it is simply stacked on top of an existing vnode, *nullfs* can provide a loopback file system by mounting the file system rooted at its source vnode at some other location in the filesystem tree. The code for *nullfs* is also an excellent starting point for designers who want to build their own filesystem layers. Examples that could be built include a compression layer or an encryption layer.

The *union* file system is another example of a middle filesystem layer. Like *nullfs*, it does not store data but just provides a name-space transformation. It is loosely modeled on the work on the 3-D file system [9], on the Translucent file system [8], and on the Automounter [19]. The *union* file system takes an existing file system and transparently overlays the latter on another file system. Unlike most other file systems, a union mount does not cover up the directory on which the file system is mounted. Instead, it shows the logical merger of the two directories and allows both directory trees to be accessible simultaneously [18].

1988: Raising the Blocksize

By 1988, disk capacity had risen enough that the default blocksize was raised to 8-kbyte blocks with 1-kbyte fragments. Although this meant that small files used a minimum of two disk sectors, the nearly doubled throughput provided by doubling the blocksize seemed a reasonable tradeoff for the measured 1.4% of additional wasted space.

1990: Dynamic Block Reallocation

Through most of the 1980s, the optimal placement for files was to lay them out using every other block on the disk. By leaving a gap around each allocated block, the disk had time to schedule the next read or write following the completion of the previous operation. With the advent of disk-track caches and the ability to handle multiple outstanding requests (tag queueing) in the late 1980s, it became desirable to begin laying files out contiguously on the disk.

The operating system has no way of knowing how big a file will be when it is first opened for writing. If it assumes that all files will be big and tries to place them in its largest area of available space, it will soon have only small areas of contiguous space available. Conversely, if it assumes that all files will be small and tries to place them in its areas of fragmented space, then the beginning of files that do grow large will be poorly laid out.

To avoid these problems the file system was changed in 1990 to do dynamic block reallocation. The file system initially places the file's blocks in small areas of free space, but then moves them to larger areas of free space as the file grows. With this technique, small files use the small chunks of free space whereas the large ones get laid out contiguously in the large areas of free space. The algorithm does not tend to increase I/O load, because the buffer cache generally holds the file contents long enough that the final block allocation has been determined by the first time that the file data is flushed to disk.

The effect of this algorithm is that the free space remains largely unfragmented even after years of use. A Harvard study found only a 15% degradation in throughput on a three-year-old file system versus a 40% degradation on an identical file system that had had the dynamic reallocation disabled [25].

1996: Soft Updates

In file systems, metadata (e.g., directories, inodes, and free block maps) gives structure to raw storage capacity. Metadata provides pointers and descriptions for linking multiple disk sectors into files and identifying those files. To be useful for persistent storage, a file system must maintain the integrity of its metadata in the face of unpredictable system crashes, such as power interruptions and operating system failures. Because such crashes usually result in the loss of all information in volatile main memory, the information in nonvolatile storage (i.e., disk) must always be consistent enough to deterministically reconstruct a coherent filesystem state. Specifically, the on-disk image of the file system must have no dangling pointers to uninitialized space, no ambiguous resource ownership caused by multiple pointers, and no unreferenced live resources. Maintaining these invariants generally requires sequencing (or atomic grouping) of updates to small on-disk metadata objects.

Traditionally, the file system used synchronous writes to properly sequence stable storage changes. For example, creating a file involves first allocating and initializing a new inode and then filling in a new directory entry to point to it. With the synchronous write approach, the file system forces an application that creates a file to wait for the disk write that initializes the on-disk inode. As a result, filesystem operations such as file creation and deletion proceed at disk speeds rather than processor or memory speeds [15, 17, 24]. Since disk access times are long compared to the speeds of other computer components, synchronous writes reduce system performance.

The metadata update problem can also be addressed with other mechanisms. For example, one can eliminate the need to keep the on-disk state consistent by using NVRAM technologies, such as an uninterruptible power supply or Flash RAM [16, 31]. Filesystem operations can proceed as soon as the block to be written is copied into the stable store, and updates can propagate to disk in any order and whenever it is convenient. If the system fails, unfinished disk operations can be completed from the stable store when the system is rebooted.

Another approach is to group each set of dependent updates as an atomic operation with some form of write-ahead logging [3, 6] or shadow-paging [2, 22A, 26]. These approaches augment the on-disk state with a log of filesystem updates on a separate disk or in stable store. Filesystem operations can then proceed as soon as the operation to be done is written into

the log. If the system fails, unfinished filesystem operations can be completed from the log when the system is rebooted. Many modern file systems successfully use write-ahead logging to improve performance compared to the synchronous write approach.

In Ganger and Patt [4], an alternative approach called soft updates was proposed and evaluated in the context of a research prototype. Following a successful evaluation, a production version of soft updates was written for BSD in 1996. With soft updates, the file system uses delayed writes (i.e., write-back caching) for metadata changes, tracks dependencies between updates, and enforces these dependencies at write-back time. Because most metadata blocks contain many pointers, cyclic dependencies occur frequently when dependencies are recorded only at the block level. Therefore, soft updates track dependencies on a per-pointer basis, which allows blocks to be written in any order. Any still-dependent updates in a metadata block are rolled back before the block is written and rolled forward afterward. Thus, dependency cycles are eliminated as an issue. With soft updates, applications always see the most current copies of metadata blocks, and the disk always sees copies that are consistent with its other contents.

1999: Snapshots

In 1999, the file system added the ability to take snapshots. A filesystem snapshot is a frozen image of a file system at a given instant in time. Snapshots support several important features, including the ability to provide backups of the file system at several times during the day and the ability to do reliable dumps of live file systems.

Snapshots may be taken at any time. When taken every few hours during the day, they allow users to retrieve a file that they wrote several hours earlier and later deleted or overwrote by mistake. Snapshots are much more convenient to use than dump tapes and can be created much more frequently.

To make a snapshot accessible to users through a traditional filesystem interface, the system administrator uses the mount command to place the replica of the frozen file system at whatever location in the namespace is convenient.

Once filesystem snapshots are available, it becomes possible to safely dump live file systems. When dump notices that it is being asked to dump a mounted file system, it can simply take a snapshot of the file system and run over the snapshot instead of on the live file system. When dump completes, it releases the snapshot.

2001: Raising the Blocksize, Again

By 2001 disk capacity had risen enough that the default blocksize was raised to 16-kbyte blocks with 2-kbyte fragments. Although this meant that small files used a minimum of four disk sectors, the nearly doubled throughput provided by doubling the blocksize seemed a reasonable tradeoff for the measured 2.9% of additional wasted space.

2002: Background Fsck

Traditionally, after an unclean system shutdown, the filesystem check program, fsck, has had to be run over all the inodes in a file system to ascer-

tain which inodes and blocks are in use and to correct the bitmaps. This check is a painfully slow process that can delay the restart of a big server for an hour or more. Soft updates guarantee the consistency of all filesystem resources, including the inode and block bitmaps. With soft updates, the only inconsistency that can arise in the file system (barring software bugs and media failures) is that some unreferenced blocks may not appear in the bitmaps and some inodes may have to have overly high link counts reduced. Thus, it is completely safe to begin using the file system after a crash without first running fsck. However, some filesystem space may be lost after each crash. Thus, there is value in having a version of fsck that can run in the background on an active file system to find and recover any lost blocks and adjust inodes with overly high link counts.

With the addition of snapshots, the task becomes simple, requiring only minor modifications to the standard fsck. When run in background cleanup mode, fsck starts by taking a snapshot of the file system to be checked. Fsck then runs over the snapshot filesystem image doing its usual calculations, just as in its normal operation. The only other change comes at the end of its run, when it wants to write out the updated versions of the bitmaps. Here, the modified fsck takes the set of blocks that it finds were in use at the time of the snapshot and removes this set from the set marked as in use at the time of the snapshot—the difference is the set of lost blocks. It also constructs the list of inodes whose counts need to be adjusted, then uses a new system call to notify the file system of the identified lost blocks so that it can replace them in its bitmaps. It also gives the set of inodes whose link counts need to be adjusted; those inodes whose link count is reduced to zero are truncated to zero length and freed. When fsck completes, it releases its snapshot. The complete details of how background fsck is implemented can be found in McKusick [11, 12].

2003: Multi-Terabyte Support

The original BSD fast file system and its derivatives have used 32-bit pointers to reference the blocks used by a file on the disk. At the time of its design in the early 1980s, the largest disks were 330 Mbytes. There was debate at the time whether it was worth squandering 32 bits per block pointer rather than using the 24-bit block pointers of the file system it replaced. Luckily, the futurist view prevailed, and the design used 32-bit block pointers.

Over the 20 years since it has been deployed, storage systems have grown to hold over a terabyte of data. Depending on the blocksize configuration, the 32-bit block pointers of the original file system run out of space in the 1-to-4-terabyte range. Although some stopgap measures can be used to extend the maximum-size storage systems supported by the original file system, by 2002 it became clear that the only long-term solution was to use 64-bit block pointers. Thus, we decided to build a new file system, one that would use 64-bit block pointers.

We considered the alternatives of trying to make incremental changes to the existing file system versus importing another existing file system such as XFS [27] or ReiserFS [20]. We also considered writing a new file system from scratch so that we could take advantage of recent filesystem research and experience. We chose to extend the original file system, because this approach allowed us to reuse most of its existing code base. The benefits of this decision were that the 64-bit-block–based file system was developed and deployed quickly, it became stable and reliable rapidly, and the same code base could be used to support both 32-bit-block and 64-bit-block

filesystem formats. Over 90% of the code base is shared, so bug fixes and feature or performance enhancements usually apply to both filesystem formats.

At the same time that the file system was updated to use 64-bit block pointers, an addition was made to support extended attributes. Extended attributes are a piece of auxiliary data storage associated with an inode that can be used to store auxiliary data that is separate from the contents of the file. The idea is similar to the concept of data forks used in the Apple file system [1]. By integrating the extended attributes into the inode itself, it is possible to provide the same integrity guarantees as are made for the contents of the file itself. Specifically, the successful completion of an fsync system call ensures that the file data, the extended attributes, and all names and paths leading to the names of the file are in stable store.

2004: Access-Control Lists

Extended attributes were first used to support an access control list, generally referred to as an ACL. An ACL replaces the group permissions for a file with a more specific list of the users who are permitted to access the files. The ACL also includes a list of the permissions each user is granted. These permissions include the traditional read, write, and execute permissions, along with other properties such as the right to rename or delete the file [21].

Earlier implementations of ACLs were done with a single auxiliary file per file system that was indexed by the inode number and had a small fixed-sized area to store the ACL permissions. The small size kept the size of the auxiliary file reasonable, since it had to have space for every possible inode in the file system. There were two problems with this implementation. The fixed size of the space per inode to store the ACL information meant that it was not possible to give access to long lists of users. The second problem was that it was difficult to atomically commit changes to the ACL list for a file, since an update required that both the file inode and the ACL file be written in order to have the update take effect [28].

Both problems with the auxiliary file implementation of ACLs are fixed by storing the ACL information directly in the extended-attribute data area of the inode. Because of the large size of the extended attribute data area (a minimum of 8 kbytes and typically 32 kbytes), long lists of ACL information can be stored easily. Space used to store extended attribute information is proportional to the number of inodes with extended attributes and the size of the ACL lists they use. Atomic updating of the information is much easier, since writing the inode will update the inode attributes and the set of data that it references, including the extended attributes in one disk operation. Although it would be possible to update the old auxiliary file on every fsync system call done on the file system, the cost of doing so would be prohibitive. Here, the kernel knows whether the extended attribute data block for an inode is dirty and can write just that data block during an fsync call on the inode.

2005: Mandatory Access Controls

The second use for extended attributes was for data labeling. Data labels provide permissions for a mandatory access control (MAC) framework enforced by the kernel. The kernel's MAC framework permits dynamically introduced system-security modules to modify system security functional-

ity. This framework can be used to support a variety of new security services, including traditional labeled mandatory access control models. The framework provides a series of entry points that are called by code supporting various kernel services, especially with respect to access control points and object creation. The framework then calls out to security modules to offer them the opportunity to modify security behavior at those MAC entry points. Thus, the file system does not codify how the labels are used or enforced. It simply stores the labels associated with the inode and produces them when a security module needs to query them to do a permission check [29, 30].

2006: Symmetric Multi-Processing

In the late 1990s, the FreeBSD Project began the long hard task of converting their kernel to support symmetric multi-processing. The initial step was to add a giant lock around the entire kernel to ensure that only one processor at a time could be running in the kernel. Each kernel subsystem was brought out from under the giant lock by rewriting it to be able to be executed by more than one processor at a time. The vnode interface was brought out from under the giant lock in 2004. The disk subsystem became multi-processor–safe in 2005. Finally, in 2006, the fast file system was overhauled to support symmetric multi-processing, completing the giant-free path from system call to hardware.

Further Information

For those interested in learning more about the history of BSD, additional information is available from http://www.mckusick.com/history/.

REFERENCES

[1] Apple, "Mac OS X Essentials, Chapter 9 Filesystem, Section 12 Resource Forks" (2003): http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/FileSystem/chapter_9_section_12.html.

[2] D. Chamberlin and M. Astrahan, "A History and Evaluation of System R," *Communications of the ACM* (24, 10) (1981), pp. 632–646.

[3] S. Chutani, O. Anderson, M. Kazar, W. Mason, and R. Sidebotham, "The Episode File System," *USENIX Winter 1992 Technical Conference Proceedings* (January 1992), pp. 43–59.

[4] G. Ganger and Y. Patt, "Metadata Update Performance in File Systems," *First USENIX Symposium on Operating Systems Design and Implementation* (November 1994), pp. 49–60.

[5] J. L. Griffin, J. Schindler, S.W. Schlosser, J.S. Bucy, and G.R. Ganger, "Timing-accurate Storage Emulation," *Proceedings of the USENIX Conference on File and Storage Technologies* (January 2002), pp. 75–88.

[6] R. Hagmann, "Reimplementing the Cedar File System Using Logging and Group Commit," *ACM Symposium on Operating Systems Principles* (November 1987), pp. 155–162.

[7] J. S. Heidemann and G.J. Popek, "File-System Development with Stackable Layers," *ACM Transactions on Computer Systems* (12, 1) (February 1994), pp. 58–89.

[8] D. Hendricks, "A Filesystem for Software Development," *USENIX Summer 1990 Technical Conference Proceedings* (June 1990), pp. 333–340.

[9] D. Korn and E. Krell, "The 3-D File System," *USENIX Summer 1989 Technical Conference Proceedings* (June 1989), pp. 147–156.

[10] C.R. Lumb, J. Schindler, and G.R. Ganger, "Freeblock Scheduling Outside of Disk Firmware," *Proceedings of the USENIX Conference on File and Storage Technologies* (January 2002), pp. 275–288.

[11] M.K. McKusick, "Running Fsck in the Background," *Proceedings of the BSDCon 2002 Conference* (February 2002), pp. 55–64.

[12] M.K. McKusick, "Enhancements to the Fast Filesystem to Support Multi-Terabyte Storage Systems," *Proceedings of the BSDCon 2003 Conference* (September 2003), pp. 79–90.

[13] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems* (2, 3) (August 1984), pp. 181–197.

[14] M.K. McKusick and T.J. Kowalski, "Fsck: The UNIX File System Check Program," in *4.4BSD System Manager's Manual* (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 3, pp. 1–21.

[15] L. McVoy and S. Kleiman, "Extent-like Performance from a UNIX File System," *USENIX Winter 1991 Technical Conference Proceedings* (January 1991), pp. 33–44.

[16] J. Moran, R. Sandberg, D. Coleman, J. Kepecs, and B. Lyon, "Breaking Through the NFS Performance Barrier," *Proceedings of the Spring 1990 European UNIX Users Group Conference* (April 1990), pp. 199–206.

[17] J. Ousterhout, "Why Aren't Operating Systems Getting Faster as Fast as Hardware?" *USENIX Summer 1990 Technical Conference* (June 1990), pp. 247–256.

[18] J. Pendry and M.K. McKusick, "Union Mounts in 4.4BSD-Lite," *USENIX 1995 Technical Conference Proceedings* (January 1995), pp. 25–33.

[19] J. Pendry and N. Williams, "AMD: The 4.4BSD Automounter Reference Manual," in *4.4BSD System Manager's Manual* (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 13, pp. 1–57.

[20] H. Reiser, "The Reiser File System" (January 2001): http://www.namesys.com/res_whol.shtml.

[21] T. Rhodes, "FreeBSD Handbook, Chapter 3, Section 3.3 File System Access Control Lists" (2003): http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/fs-acl.html.

[22A] M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer System* (10, 1) (February 1992): 26–52.

[22B] D. Rosenthal, "Evolving the Vnode Interface," *USENIX Winter 1990 Technical Conference Proceedings* (June 1990), pp. 107–118.

[23] J. Schindler, J.L. Griffin, C.R. Lumb, and G.R. Ganger, "Track-aligned Extents: Matching Access Patterns to Disk Drive Characteristics," *Proceedings of the USENIX Conference on File and Storage Technologies* (January 2002), pp. 259–274.

[24] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Proceedings of the USENIX Winter 1993 Conference* (January 1993), pp. 307–326.

[25] K. Smith and M. Seltzer, "A Comparison of FFS Disk Allocation Algorithms," *Proceedings of the USENIX 1996 Annual Technical Conference* (January 1996), pp. 15–25.

[26] M. Stonebraker, "The Design of the POSTGRES Storage System," *Very Large DataBase Conference* (1987), pp. 289–300.

[27] A. Sweeney, D. Doucette, C. Anderson, W. Hu, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," *Proceedings of the 1996 USENIX Annual Technical Conference* (January 1996), pp. 1–14.

[28] R. Watson, "Introducing Supporting Infrastructure for Trusted Operating System Support in FreeBSD," *Proceedings of the BSDCon 2000 Conference* (September 2000).

[29] R. Watson, "TrustedBSD: Adding Trusted Operating System Features to FreeBSD," *Proceedings of the FREENIX Track at the 2001 USENIX Annual Technical Conference* (June 2001), pp. 15–28.

[30] R. Watson, W. Morrison, C. Vance, and B. Feldman, "The TrustedBSD MAC Framework: Extensible Kernel Access Control for FreeBSD 5.0," *Proceedings of the FREENIX Track at the 2003 USENIX Annual Technical Conference* (June 2003), pp. 285–296.

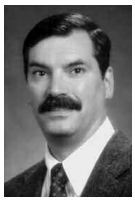
[31] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (October 1994), pp. 86–97.

PAWEL JAKUB DAWIDEK AND
MARSHALL KIRK MCKUSICK

porting the Solaris ZFS file system to the FreeBSD operating system

Pawel Jakub Dawidek is a FreeBSD committer. In the FreeBSD project he works mostly in the storage subsystems area (GEOM, file systems), security (disk encryption, opencrypto framework, IPsec, jails), but his code is also in many other parts of the system. Pawel currently lives in Warsaw, Poland, running his small company.

pjd@FreeBSD.org



Dr. Marshall Kirk McKusick writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues, particularly those related to operating systems and file systems. While at the University of California at Berkeley, he implemented the 4.2BSD fast file system and was the Research Computer Scientist at the Berkeley Computer Systems Research Group (CSRG) overseeing the development and release of 4.3BSD and 4.4BSD.

mckusick@mckusick.com

THE ZFS FILE SYSTEM MADE REVOLUtionary (as opposed to evolutionary) steps forward in filesystem design, with its authors claiming that they threw away 20 years of obsolete assumptions to design an integrated system from scratch. In this article, we describe the porting of ZFS to FreeBSD, along with describing some of the key features of the ZFS file system.

Features of ZFS

ZFS is more than just a file system. In addition to the traditional role of data storage, ZFS also includes advanced volume management that provides pooled storage through a collection of one or more devices. These pooled storage areas may be used for ZFS file systems or exported through a ZFS Emulated Volume (ZVOL) device to support traditional file systems such as UFS.

POOLED STORAGE MODEL

File systems created by ZFS are not tied to a specified device, volume, partition, or disk but share the storage assigned to a pool. The pool may be constructed from storage ranging from a single partition up to farms composed of hundreds of disks. If more storage is needed, new disks can be added at run time and the space is automatically made available to all the file systems sharing the pool. Thus, there is no need to manually grow or shrink the file systems when space allocation requirements change. There is also no need to create slices or partitions. When working with ZFS, tools such as fdisk(8), bsdlabel(8), newfs(8), tunefs(8), and fsck(8) are no longer needed.

COPY-ON-WRITE DESIGN

File systems must be in a consistent state to function in a stable and reliable way. Unfortunately, it is not easy to guarantee consistency if a power failure or a system crash occurs, because most file system operations are not atomic. For example, when a new hard link to a file is created, the file system must create a new directory entry and increase the link count in the inode. These changes usually require writing two different disk sectors. Atomicity of disk writes can only be guaranteed on a per-sector basis. Two techniques have been used to maintain filesystem consistency when multiple sectors must be updated:

- Checking and repairing the file system with the fsck utility on boot [11], a technique that has lost favor as disk systems have grown. Starting with FreeBSD 5.0, it is possible to run the fsck program in the background, significantly reducing system downtime [9].
- To allow an immediate reboot after a crash, the file system uses soft updates to guarantee that the only inconsistency the file system might experience is resource leaks stemming from unreferenced blocks or inodes [5, 10].

McKusick added the ability to create snapshots to UFS, making background fsck possible [12]. Unfortunately, filesystem snapshots have a few disadvantages, because during one step of a snapshot all write operations to the file system are blocked. Luckily, this step does not depend on filesystem size and takes only a few seconds. However, the time of the step that sets up the snapshot grows linearly with the size of the file system and generates heavy I/O load. So even though the file system continues to operate, its performance is degraded while the snapshot is being prepared.

Once a snapshot is taken, all writes to the file system must be checked to see whether an older copy needs to be saved for the snapshot. Because of the design of snapshots, copies are rarely needed and thus do not appreciably slow down the system. A slowdown does occur when removing many small files (i.e., any file less than 96 kilobytes whose last block is a fragment) that are claimed by a snapshot. In addition, checking a file system in the background slows operating system performance for many hours because of its added demands on the I/O system. If the background fsck fails (usually because of hardware-based disk errors) the operating system needs to be rebooted and the file system must be repaired in the foreground. When a background fsck has failed, it means that the system has been running with an inconsistent file system, which implies undefined behavior.

The second technique used requires storing all filesystem operations (or only metadata changes) first in a special journal. Once the entire operation has been journaled, filesystem updates may be made. If a power failure or a system crash occurs, incomplete entries in the journal are removed and partially completed filesystem updates are finished by using the completed entries stored in the journal. Filesystem journaling is currently the most popular way of managing filesystem consistency [1, 17, 18].

The ZFS file system needs neither fsck nor journals to guarantee consistency. Instead it takes an alternate copy-on-write (COW) approach. COW means that ZFS never overwrites valid data. Instead, ZFS always writes data into a free area. When the data is safely stored, ZFS switches a single pointer in the block's parent. With this technique, block pointers never point at inconsistent blocks. This design is similar to the WAFL file system design [6].

END-TO-END DATA INTEGRITY AND SELF-HEALING

Another important ZFS feature is end-to-end data integrity. All data and metadata undergoes checksum operations using one of several available algorithms (fletcher2, fletcher4 [4], or SHA256 [14]). ZFS can detect silent data corruption caused by any defect in disk, controller, cable, driver, or firmware. There have been many reports from Solaris users of silent data corruption that has been successfully detected by ZFS. If the storage pool has been configured with some level of redundancy (RAID-Z or mirroring) and data corruption is detected, ZFS not only reconstructs the data but also writes valid data back to the component where corruption was originally detected.

SNAPSHOTS AND CLONES

Snapshots are easy to implement for file systems such as ZFS that store data using a COW model. When new data are created, the file system simply does not free the block with the old data. Thus, snapshots in ZFS are cheap to create (unlike UFS2 snapshots). ZFS also allows the creation of a clone, which is a snapshot that may be written. Finally, ZFS has a feature that allows it to roll back a snapshot, forgetting all modifications introduced after the snapshot was created.

ZFS supports compression at the block level. Currently, Jeff Bonwick's variant of the Lempel-Ziv compression algorithm and the gzip compression algorithm are supported. Data encryption is also a work in progress [13].

Porting ZFS to FreeBSD

We describe work done by Pawel Jakub Dawidek in porting ZFS to FreeBSD in the remainder of this article. This task seemed daunting at first, as a student had spent an entire Summer of Code project looking at porting ZFS to Linux and had made little progress. However, a study of the ZFS code showed that it had been written with portability in mind. The ZFS code is clean, well commented, and self-contained. The source files rarely

include system headers directly. Most of the time, they include only ZFS-specific header files and a special zfs_context.h header where system-specific includes are placed. Large parts of the kernel code can be compiled in a user process and run by the ztest utility for regression and stress testing.

So, Dawidek felt a fresh start on doing a port seemed appropriate, this time taking the approach of making minimal changes to the ZFS code base itself. Instead, Dawidek built a set of software compatibility modules to convert from the FreeBSD internal interfaces to those used by Solaris and expected by ZFS. Using this approach, he had an initial port up and running with just ten days of effort.

SOLARIS COMPATIBILITY LAYER

When a large project such as ZFS is ported from another operating system, it is important to keep modifications of the original code to a minimum. Having fewer modifications makes porting easier and makes the importation of new functionality and bug fixes much less difficult.

To minimize the number of changes, Dawidek created a Solaris-compatible application programming interface (API) layer. The main goal was to implement the Solaris kernel functions that ZFS expected to call. These functions were implemented by using the FreeBSD kernel programming interface (KPI). Many of the API differences were simple, involving different function names, slightly different arguments, or different return values. For other APIs, the functionality needed to be fully implemented from scratch. This technique proved to be quite effective. For example, after these stubs were built, only 13 files out of 112 of the core ZFS implementation directory needed to be modified.

The following milestones were defined to port the ZFS file system to FreeBSD:

1. Create a Solaris-compatible API using the FreeBSD API.
2. Port the user-level utilities and libraries.
3. Define connection points in ZFS where FreeBSD makes its service requests. These service requests include:
   - ZFS POSIX Layer, which has to be able to communicate with the virtual filesystem (VFS) layer
   - ZFS Emulated Volume (ZVOL), which has to be able to communicate with the FreeBSD volume-management subsystem (GEOM)
   - /dev/zfs, a control device that communicates with the ZFS user-level utilities and libraries
4. Define connection points in ZFS where the storage pool virtual device (VDEV) needs to make I/O requests to FreeBSD.

ZFS POSIX LAYER

The ZFS POSIX layer receives requests from the FreeBSD VFS interface. This interface was the hardest part of the entire port to implement. The VFS interface has many complex functions and is quite system-specific. Although the Solaris and FreeBSD VFS interfaces had a common heritage twenty years ago, much has changed between them over the years. VFS on Solaris seems to be cleaner and a bit less complex than FreeBSD's.

ZFS EMULATED VOLUME

A ZFS VDEV managed storage pool can serve storage in two ways, as a file system or as a raw storage device. ZVOL is a ZFS layer responsible for exporting part of a VDEV-managed storage pool as a disk device.

FreeBSD has its own GEOM layer, which can also be used to manage raw storage devices either to aggregate them with RAID or by striping, or to subdivide them using partitioning. GEOM can also be used to provide compression or encryption (see [12], pp. 270–276, for details on GEOM).

To maximize the flexibility of ZVOL, a new ZVOL provider-only GEOM class was created. As a GEOM provider, the ZVOL storage pool is exported as a device in /dev/ (just like other GEOM providers). So, it is possible to use a ZFS storage pool for a UFS file system or to hold a swap-space partition.

ZFS VIRTUAL DEVICES

A ZFS VDEV-managed storage pool has to use storage provided by the operating system [16]. The VDEV has to be connected to storage at its bottom layer. In Solaris there are two types of storage used by VDEVs: storage from disks and storage from files. In FreeBSD, VDEVs can use storage from any GEOM provider (disk, slice, partition, etc.). ZFS can access files by making them look like disks using an md(4) device.

Rather than interfacing directly to the disks, a new VDEV consumer-only GEOM class was created to interface ZFS to the GEOM layer in

FreeBSD. In its simplest form, GEOM just passes an uninterpreted raw disk to ZFS. But all the functionality of the GEOM layer can be used to build more complex storage arrangements to pass up to a VDEV-managed storage pool.

EVENT NOTIFICATION

ZFS has the ability to send notifications on various events. Those events include information such as storage pool imports as well as failure notifications (I/O errors, checksum mismatches, etc.). Dawidek ported this functionality to send notifications to the devd(8) daemon, which seemed to be the most suitable communication channel for those types of messages. In the future, a dedicated user-level daemon to manage messages from ZFS may be written.

KERNEL STATISTICS

Solaris exports various statistics (mostly about ZFS-cache and name-cache usage) via its kstat interface. This functionality was directed to the FreeBSD sysctl(9) interface. All statistics can be printed using the following command:

```
# sysctl kstat
```

ZFS AND FREEBSD JAILS

ZFS works with Solaris zones [15]. In our port, we make it work with FreeBSD jails [7], which have many of the same features as zones. A useful attribute of ZFS is that once it has constructed a pool from a collection of disks, new file systems can be created and managed from the pool without requiring direct access to the underlying disk devices. Thus, a jailed process can be permitted to manage its own file system since it cannot affect the file systems of other jails or of the base FreeBSD system. If the jailed process were permitted to directly access the raw disk, it could mount a denial-of-service attack by creating a file system with corrupted metadata and then panicking the kernel by trying to access that file system.

ZFS fits into the jail framework well. Once a pool has been assigned to a jail, the jail can operate on its own file system tree. For example:

```
main# zpool create tank mirror da0 da1
main# zfs create tank/jail
main# zfs set jailed=on tank/jail
main# zfs jail 1 tank/jail
jail# zfs create tank/jail/home
jail# zfs create tank/jail/home/pjd
jail# zfs create tank/jail/home/mckusick
jail# zfs snapshot tank/jail@backup
```

FreeBSD Modifications

There were only a few FreeBSD modifications needed to port the ZFS file system.

The mountd(8) program was modified to work with multiple export files. This change allows the zfs(1) command to manage private export files stored in /etc/zfs/exports.

The vnode-pointer-to-file-handle (VPTOFH) operation was switched from one based on the filesystem type (VFS_VPTOFH) to one based on the vnode type (VOP_VPTOFH). Architecturally, the VPTOFH translation should always have been a vnode operation, but Sun first defined it as a filesystem operation, so BSD did the same to be compatible. Solaris changed it to a vnode operation years ago, so it made sense for FreeBSD to do so as well. This change allows VPTOFH to support multiple node types within one file system. For example, in ZFS the v_data field from the vnode structure can point at two different structures (either znode_t or zfsctl_node_t). To be able to recognize which structure it references, two different vop_vptofh functions are defined for those two different types of vnodes.

The lseek(2) API was extended to support the SEEK_DATA and SEEK_HOLE operation types [2]. These operations are not ZFS-specific. They are useful on any file system that supports holes in files, as they allow backup software to identify and skip holes in files.

The jail framework was extended to support "jail services." With this extension, ZFS can register itself as a jail service and attach a list of assigned ZFS datasets to the jail's in-kernel structures.

User-level Utilities and Libraries

User-level utilities and libraries communicate with the kernel part of ZFS via the /dev/zfs control device. We needed to port the following utilities and libraries:

- zpool: utility for storage pools configuration
- zfs: utility for ZFS file systems and volumes configuration
- ztest: program for stress testing most of the ZFS code
- zdb: ZFS debugging tool

- libzfs: the main ZFS user-level library used by both the zfs and zpool utilities
- libzpool: test library containing most of the kernel code, used by ztest

To make it work, we also ported libraries (or implemented wrappers) they depend on: libavl, libnvpair, libutil, and libumem.

Testing FileSystem Correctness

It is quite important and very hard to verify that a file system works correctly. The file system is a complex beast and there are many corner cases that have to be checked. If testing is not done right, bugs in a file system can lead to application misbehavior, system crashes, data corruption, or even security failure. Unfortunately, we did not find freely available filesystem test suites that verify POSIX conformance. Instead, Dawidek wrote the fstest test suite [3]. The test suite currently contains 3438 tests in 184 files and tests all the major filesystem operations including chflags, chmod, chown, close, link, mkdir, mkfifo, open, read, rename, rmdir, symlink, truncate, and unlink.

Filesystem Performance

Below are some performance numbers that compare the current ZFS version for FreeBSD with various UFS configurations. Note that all file systems were tested with the atime option turned off. The ZFS numbers are measured with checksumming enabled, as that is the recommended configuration.

Untarring src.tar archive four times one by one:

UFS	256s
UFS+soft-updates	207s
UFS+gjournal+async	127s
ZFS	237s

Removing four src directories one by one:

UFS	230s
UFS+soft-updates	94s
UFS+gjournal+async	48s
ZFS	97s

Untarring src.tar by four processes in parallel:

UFS	345s
UFS+soft-updates	333s
UFS+gjournal+async	158s
ZFS	199s

Removing four src directories by four processes in parallel:

UFS	364s
UFS+soft-updates	185s
UFS+gjournal+async	111s
ZFS	220s

Executing dd if=/dev/zero of=/fs/zero bs=1m count=5000:

UFS	78s
UFS+soft-updates	77s
UFS+gjournal+async	200s
ZFS	111s

Status and Future Directions

After about six months of work, the ZFS port is almost finished. About 98% of the functionality is ported and tested. The primary work that remains is to tune its performance.

Here are some missing functionalities:

- ACL support. Currently ACL support has not been ported. ACL support is difficult to implement because FreeBSD has only support for POSIX.1e ACLs, whereas ZFS implements NFSv4-style ACLs. Porting NFSv4-style ACLs to FreeBSD requires the addition of system calls, updating system utilities to manage ACLs, and preparing procedures on how to convert from one ACL type to another.
- Allowing ZFS to export ZVOLs over iSCSI. At this point there is no iSCSI target daemon in the FreeBSD base system, so there is nothing with which to integrate this functionality.
- Code optimization. Many parts of the code were written quickly but inefficiently.

ZFS was recently merged into the FreeBSD base system. Indeed, it may be ready for version 7.0 release. There are no plans to merge ZFS to the RELENG_6 branch.

The UFS file system supports system flags—chflags(2). There is no support for those in the ZFS file system, but it would be easy to add support for system flags to ZFS.

There is no encryption support in ZFS itself, but there is an ongoing project to implement it. It may be possible to cooperate with Sun developers to help finish this project. With a properly defined interface within ZFS, it would be easy to integrate encryption support provided by the opencrypto framework [8].

ACKNOWLEDGMENTS

We would like to thank the ZFS developers, who created a great file system, and the FreeBSD Foundation (www.FreeBSDFoundation.org) for their support. The machines from the FreeBSD Netperf Cluster (www.freebsd.org/projects/netperf/cluster.html)were used for much of the development work. Pawel Jakub Dawidek would like to thank Wheel LTD (www.wheel.pl). He was able to do this work during his day job. Finally, we would like to thank the FreeBSD community for their never-ending support and warm words.

REFERENCES

[1] S. Best, "JFS Overview" (2000): http://www-128.ibm.com/developerworks/linux/library/l-jfs.html.

[2] J. Bonwick, "SEEK_HOLE and SEEK_DATA for Sparse Files" (2005): http://blogs.sun.com/bonwick/entry/seek_hole_and_seek_data.

[3] P. Dawidek, "File System Test Suite" (2007): http://people.freebsd.org/~pjd/fstest/.

[4] J. Fletcher, "Fletcher's Checksum" (1990): http://en.wikipedia.org/wiki/Fletcher's_checksum.

[5] G. Ganger, M.K. McKusick, C. Soules, and Y. Patt, "Soft Updates: A Solution to the Metadata Update Problem in File Systems," *ACM Transactions on Computer Systems* 18(2) (2000): 127–153.

[6] D. Hitz, J. Lau, and M. Malcolm, "File System Design for an NFS File Server Appliance," *USENIX Association Conference Proceedings* (Berkeley, CA: USENIX Association, 1994), pp. 235–246.

[7] P. Kamp and R. Watson, "Jails: Confining the Omnipotent Root," *Proceedings of the Second International System Administration and Networking Conference (SANE)* (2000): http://docs.freebsd.org/44doc/papers/jail/.

[8] S. Leffler, "Cryptographic Device Support for FreeBSD," *Proceedings of BSDCon 2003* (Berkeley, CA: USENIX, 2003), pp. 69–78.

[9] M.K. McKusick, "Running Fsck in the Background," *Proceedings of the BSDCon 2002 Conference*, pp. 55–64.

[10] M.K. McKusick and G. Ganger, "Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem," *Proceedings of the FREENIX Track at the 1999 USENIX Annual Technical Conference* (Berkeley, CA: USENIX Association, 1999), pp. 1–17.

[11] M.K. McKusick and T.J. Kowalski, "Fsck: The UNIX File System Check Program," in *4.4BSD System Manager's Manual* (Sebastopol, CA: O'Reilly & Associates, 1994), vol. 3, pp. 1–21.

[12] M.K. McKusick and G. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System* (Reading, MA: Addison-Wesley, 2005).

[13] D. Moffat, "ZFS Encryption Project" (2006): www.opensolaris.org/os/project/zfs-crypto/files/zfs-crypto.pdf.

[14] NIST, "SHA Hash Functions" (1993): http://en.wikipedia.org/wiki/SHA-256.

[15] D. Price and A. Tucker, "Solaris Zones: Operating System Support for Consolidating Commercial Workloads," *Proceedings of LISA '04: 18th Large Installation System Administration Conference* (Berkeley, CA: USENIX Association, 2004), pp. 241–254.

[16] Sun Microsystems, "ZFS Source Tour" (2007): http://www.opensolaris.org/os/community/zfs/source/.

[17] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS File System," *USENIX 1996 Annual Technical Conference Proceedings* (Berkeley, CA: USENIX Association, 1996), pp. 1–14.

[18] S. Tweedie, "EXT3, Journaling Filesystem," Ottawa Linux Symposium (2003): http://ssrc.cse.ucsc.edu/PaperArchive/ext3.html.

AVANTIKA MATHUR, MINGMING CAO, AND ANDREAS DILGER

ext4: the next generation of the ext3 file system

Avantika Mathur is a Linux kernel developer in the IBM Linux Technology Center. Her primary focus is ext3 and ext4 filesystem development and testing.

mathur@us.ibm.com

Mingming Cao has been a Linux kernel developer at the IBM Linux Technology Center for 7 years, mainly in the areas of IPC, block IO, and the ext3 filesystem. Her recent focus has been on bringing up the ext4 filesystem.

cmm@us.ibm.com

Andreas Dilger is a Principal Systems Software Engineer for Cluster Filesystems, Inc., designing and developing the Lustre distributed file system on some of the largest computers in the world. He started programming more than 25 years ago and has spent much of the last 10 years focused on Linux filesystem development.

adilger@clusterfs.com

LAST YEAR, A NEW LINUX FILE SYSTEM was born: ext4. A descendant of the ext3 file system, ext4 will soon replace ext3 as the "Linux file system." Ext4 provides greater scalability and higher performance than ext3, while maintaining reliability and stability, giving users many reasons to switch to this new file system. The primary goal for ext4 is to support larger files and file systems. Once it is mature, the developing ext4 file system will be suitable for a greater variety of workloads, from desktop to enterprise solutions.

Why Ext4

Among the many file systems Linux offers today, ext3 is the most popular, with the largest user base and development community. Having been designed with stability and maintenance in mind makes ext3 a very reliable file system with relatively good performance. Thus it has been the default file system in many commercial Linux distributions for many years.

However, the conservative design of ext3 limits its scalability and performance. One of the hard limits faced by ext3 today is the 16-TB filesystem size maximum. This limit has already been reached in large installations and will soon be hit by desktop users. Today, 1-TB external hard drives are readily available in stores, and disk capacity can double every year.

Last year, a series of patches were sent to the Linux kernel mailing list, to address filesystem capacity and to add extents mapping to ext3. The extent patches would cause intrusive changes to the on-disk format and break forward compatibility for file systems that used them. In order to maintain the stable ext3 file system for its massive user base, it was decided to fork the ext4 file system from ext3 and address performance and scalability issues in this new file system.

Why not use a file system such as XFS for this? The answer is that the XFS code in Linux is very complex, being burdened with an extra layer of compatibility code for IRIX. Even with the addition of the features described here, ext4 is still much smaller and more understandable than XFS (25k lines vs. 106k lines). Also, there is a considerable investment in the stability and robustness of the ext3 and e2fsck code base, most of which will continue to be used for ext4.

What's New in Ext4

Ext4 was included in mainline Linux version 2.6.19. The file system is currently in development mode, titled ext4dev, explicitly warning users that it is not ready for production use. There are many new features in the ext4 road map under development and testing. Some of the features in progress may continue to change the filesystem layout. Any future changes, as with the current ones, will be protected by appropriate feature flags so that existing kernels and e2fsprogs will be able to know whether it is safe to mount a given ext4 file system. Once the layout is finalized, ext4 will be converted from development to stable mode. At that point, ext4 will be stable and available for general use by all users in need of a more scalable and modern version of ext3.

The initial version of the ext4 file system includes two new key features: extent support and 48-bit block numbers. Combined, these features support larger file systems and better performance on large files.

EXTENT SUPPORT

The ext3 file system uses the traditional indirect block mapping scheme, which is efficient for small or sparse files but causes high metadata overhead and poor performance when dealing with large files, especially on delete and truncate operations. To address this issue, ext4 uses extent mapping as an efficient way to represent large contiguous files.

An extent is a single descriptor that represents a range of contiguous blocks. A single extent in ext4 can represent up to 128 MB. Four extents can be stored directly in the inode structure. That is generally sufficient for small to medium contiguous files, but for very large or highly fragmented files, an extents tree is created to efficiently look up the many extents required.

Extents mapping improves performance on large files, such as mp3, DVD, video, or database files, as it is tuned toward allocating large contiguous blocks. Extents bring about a 25% throughput gain in large sequential I/O workloads when compared with ext3. A similar performance gain was also seen on the Postmark benchmark, which simulates a mail server, with a large number of small to medium files. With extents the metadata is more compact, causing greatly reduced CPU usage.

Although the indirect block and extents mapping schemes are incompatible, both are supported by ext4, and files can be converted between the two formats. This is discussed further in the migration section.

LARGE FILE SYSTEM SUPPORT

The first issue addressed in ext4 is the filesystem capacity limit. Because ext3 uses 32 bits to represent block numbers and has a default 4k block size, the file system is limited to a maximum of 16 TB. Ext4 uses 48-bit block numbers. In theory this allows it to support 1-EB (1-million-TB) file systems. This change was made in combination with the extents patches, which use 48-bit physical block numbers in the extents structure. Other metadata changes, such as in the super-block structure, were also made to support the 48-bit block number.

In order to support more than 32-bit block numbers in the journaling block layer (JBD), JBD2 was forked from JBD at the same time that ext4

was cloned. There is also work underway to add checksumming to the journal in JBD2 to validate this critical metadata at recovery time. Currently, JBD2 is only used by ext4, but eventually both 32-bit and 64-bit Linux file systems will be able to use JBD2 for journaling support.

One may question why we chose 48-bit block numbers rather than the round 64 bits. Although it is possible to design for 64-bit ext4 file systems, this is impractical today because of reliability and serviceability problems. Having full 64-bit physical and logical block numbers would have meant that fewer extents could fit within the inode (2 vs. 4) for only very theoretical gains. It would take 119 years at today's speeds to run e2fsck on even a 2^{48}-block file system. The ext4 developers will focus their efforts on improving the reliability aspects before worrying about a theoretical size limit.

What's Next

The increased file system capacity created by the 48-bit block numbers and extent mapping features that are currently in ext4 will provide many new features in the road map. These features are focused on enhancing ext4 scalability, reliability, block placement, and performance.

An ext4 git tree is hosted at git://git.kernel.org/pub/scm/linux/kernel/git/tytso/ext4. The tree contains the series of patches in line for ext4. Up-to-date information on new ext4 features, patch sets, and development discussion can be found at the ext4 wiki page, http://ext4.wiki.kernel.org/.

BLOCK ALLOCATION ENHANCEMENTS

Fragmentation is a key factor affecting filesystem performance. The following features in the ext4 road map attempt to address performance by avoiding or decreasing fragmentation. This is essential for the efficient use of extents and maximizing performance on modern high-bandwidth storage.

PERSISTENT PREALLOCATION

Applications such as large databases often write zeros to a file for guaranteed and contiguous filesystem space reservation. Persistent preallocation in ext4 allocates a contiguous set of blocks for a file without the expensive zero-out. The preallocated extents contain a flag specifying that the blocks are uninitialized. These uninitialized extents are protected from undesired exposure of their contents through read operations. A new system call, sys_fallocate, is planned to be added to the Linux kernel, and the existing posix_fallocate library is being modified. These interfaces can be used by users to specify the portion of the file to preallocate.

DELAYED ALLOCATION AND MULTIPLE BLOCK ALLOCATION

The ext3 block allocation scheme is not very efficient for allocating extents, as blocks are allocated one by one during write operations, so ext4 will use delayed allocation and multiple block mechanisms to efficiently allocate many blocks at a time and contribute to avoiding fragmentation. With delayed allocation, block allocations are deferred to page flush time, rather than during the write operation. This avoids unnecessary block allocation for short-lived files and provides the opportunity to queue many individual block allocation requests into a single request.

The multiple block allocation feature uses a buddy data structure to efficiently locate free extents and allocates an entire extent referencing multiple blocks, rather than allocating one at a time. Combined, delayed allocation and multiple block allocation have been shown to significantly reduce CPU usage and improve throughput on large I/O. Performance testing shows a 30% throughput gain for large sequential writes and 7–10% improvement on small files (e.g., those seen in mail-server-type workloads).

ONLINE DEFRAGMENTATION

Even though there are techniques in place to attempt to avoid file fragmentation, with age, a file system can still become highly fragmented. The online defragmentation tool is designed to defragment individual files or an entire file system. The defragmentation is performed by creating a temporary inode, using multiple block allocation to allocate contiguous blocks to the inode, reading all data from the original file to the page cache, then flushing the data to disk and migrating the newly allocated blocks over to the original inode.

SCALABILITY ENHANCEMENTS

Besides enlarging the overall file system capacity, there are many other scalability features planned for ext4.

Although ext3 has support for different inode sizes, the default inode structure size is 128 bytes, with little extra room to support new features. In ext4, the default inode size will be enlarged to 256 bytes. This will provide space for the new fields needed for the planned features, nanosecond time stamps, and inode versioning. The latter is a counter incremented on each file modification that will be used by NFSv4 (network file system) to keep track of file updates.

By using a larger inode size in ext4, the EA-in-inode feature can be enabled by default. This feature is also available in ext3, but because of the smaller default inode size it is not widely used. EA-in-inode stores extended attributes (EAs) directly in the inode body, making EA access much faster. The faster EA access can greatly improve performance, sometimes by 2–3 times, for those using SELinux, ACLs, or other EAs. Additional EAs, which don't fit in the inode body, are stored in a single filesystem block. This limits the maximum EA capacity per file to 4k. In ext4, there is a plan to remove this limit by storing large EAs in a file.

To address directory scalability, the directory indexing feature, available in ext3, will be turned on by default in ext4. Directory indexing uses a specialized Btree-like structure to store directory entries, rather than a linked list with linear access times. This significantly improves performance on certain applications with very large directories, such as Web caches and mail systems using the Maildir format. Performance improvements were often by factors of 50–100, in particular for directories with more than 10,000 files. An additional scalability improvement is eliminating the 32,000 subdirectory limit in ext4.

Support for larger files, extending beyond the 2-TB limit, are in the road map. There is interest in efficiently supporting large numbers of files, surpassing the 4-billion limit, which implies 64-bit inode numbers and dynamic inode tables. Because of the potential intrusive on-disk format changes involved, plans and design aspects are still on the drawing board.

FASTER REPAIR AND RECOVERY

A file system is not very useful if it cannot be repaired or recovered in a reasonable amount of time. As the filesystem size grows, the e2fsck time becomes unbearable, taking from minutes to years depending on the file-system size. This issue is more prevalent for ext4, as it can support larger file systems. Although the ext4 journaling support tries to avoid the need for file system recovery as much as possible, in the event of a disk error e2fsck is still necessary.

The e2fsck tool scans the whole file system, regardless of whether only a small part of it is being used. With a little more information about the file system, e2fsck could skip those unused block groups or inode structures. The uninitialized block groups feature uses a flag in the block group to mark whether it is uninitialized. Once it is written to, watermarks are used to keep track of the last used inode structures. Any uninitialized block groups or inodes are not scanned by e2fsck, which greatly reduces the run time. This feature can speed up e2fsck dramatically, reducing run times to 1/2 to 1/10 of the original run time, depending on how the file system is used. The flags marking a block group uninitialized and the high water-mark are checksummed. If there is ever corruption, e2fsck will fall back to the slower, but safer, full scan for that block group. As new features are added to ext4, the user tools, e2fsprogs, will be updated correspondingly.

Migration from Ext3 to Ext4

Compatibility between ext4 and existing ext3 file systems has been maintained as much as possible. Even though ext4 has changed some parts of the on-disk format, it is possible to take advantage of many of the ext4 features, such as extents, delayed allocation, multiple block allocation, and faster e2fsck, without requiring a backup and restore.

There is an upgrade path from ext3 that allows existing file systems to start using ext4 immediately, without requiring a lengthy downtime. Users can mount an existing ext3 file system as ext4, and all existing files are still treated as ext3 files. Any new files created in this ext4 file system will be extent-mapping-based. There is a flag in each individual file to indicate whether it is an ext3 file (indirect mapped) or ext4 file (extent mapped).

There is also a way to do a systemwide migration. A set of tools is under development to migrate an entire file system from ext3 to ext4, which includes transferring indirect files to extent files and enlarging the inode structure to 256 bytes. The first part can be performed online in combination with online defragmentation. This tool will perform the conversion from ext3 to ext4 while simultaneously defragmenting the file system and files, and it will have the option of converting only part of the file system. Resizing the inodes must be performed offline, and this can be done in conjunction with converting files to extent mapping. In this case the whole file system is scanned, and proper backup is done to prevent data loss if the system crashes during the migration.

Users who are hesitant to migrate to ext4 immediately can optionally format their ext3 file system with large inodes (256 bytes or more) to take advantage of the EA-in-inode feature today and nanosecond timestamps if they migrate to ext4. This would avoid the need to do an offline migration step to resize the inodes.

There is also a downgrade path from ext4 to ext3, with a method to convert the extent files back to indirect mapping files. In the case that users prefer to go back to ext3, they can mount the ext4 file system with the "noextents" mount option, copy the extent-based ext4 files to new files, rename these over the old extents, use tunefs to clear the INCOMPAT_EXTENTS flag, and then remount as an ext3 file system.

Conclusion

As we have discussed, a tremendous amount of work has gone into the ext4 file system, and this work is ongoing. As we stabilize the file system, ext4 will become suitable for production use, making it a good choice for a wide variety of workloads. What was once essentially a simple file system has turned into an enterprise-ready modern file system, with a good balance of scalability, reliability, performance, and stability. Eventually ext4 will replace ext3 as the default Linux file system.

LEGAL STATEMENT

DAN GEER

a quant looks at the future

Dan Geer is a security researcher with a quantitative bent. His group at MIT produced Kerberos, and a number of startups later he is still at it—today as Chief Scientist at Verdasys. He writes a lot, and sometimes the output gets read, such as the semi-famous paper on whether a computing monoculture rises to the level of a national security risk. He's an electrical engineer, a statistician, and someone who thinks truth is best achieved by adversarial procedures.

dan@geer.org

THE FUTURE IS ALREADY HERE—IT'S just unevenly distributed [1]. To see some of security's future we do trend analysis on what we already know. This essay demonstrates what can be gotten from open-source intelligence of the most general sort and how it may apply to looking at the near- to medium-term future of security. As with any such work, there are limitations to the method and the results can, if one insists, be pushed too far.

Security is not an end, it is a means. As a technique, it is a form of risk management and a subset of reliability. Real risk management means good decisions, good decision-making requires good decision support, and good decision support requires ordinal scale ($X > Y$) metrics—often no more than ordinal scale.

Where does trend analysis come into this? Trend analysis is what a statistician will recommend when the underlying topic of interest is new and/or changing rapidly and where the method of measuring it is uncertain. In such a circumstance, and so long as the measurement you do have can be applied consistently, the trend data from the measurement can be relied upon even if the raw numbers the measurement returns are suspect. As trends are generally sufficient for decision support ($X > Y$), we've explained why we are here. By analogy, a street cop may never know how much crack is for sale, but he or she can tell a lot from the rise and fall of the street price—enough to make decisions.

Making decisions early is often regarded as something valuable. In the present context, it is good to remember that early decision-making is itself a tradeoff: Making decisions early is more expensive in decision cost than making them later, because early on the choice set is larger and the uncertainty around that choice set is higher. Making decisions later generally comes with fewer workable options, so decision cost per se is less. Trend analysis can thus help you decide not only what decision to make but also when to make it. These are Good Things.

Gather Ye Numbers Where Ye May

There are two sources of numbers: reports from instrumented collection points and surveys. Both may be done by others, and so we must hope that

the ways this data is collected is consistent over time. Surveys are hard to do really right, as they are subject to lots of biases, but the biases are not terribly relevant to trend analysis if those biases are consistent over time. Let's start with the well-known CSI/FBI annual survey [2] and look at the question, "Did your organization experience unauthorized use of computer systems in the last 12 months?" The question has been asked for several years, so there is something to look at (see Figure 1).

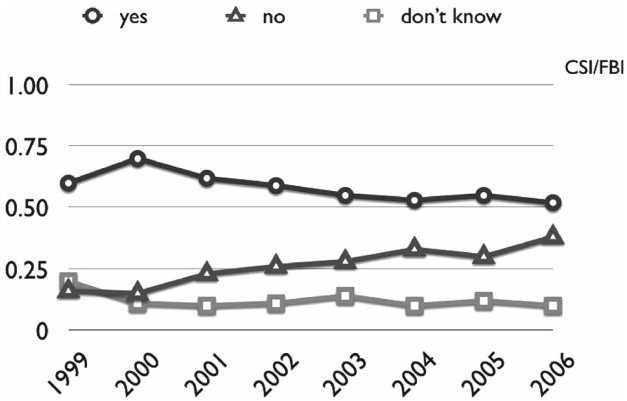


FIGURE 1: UNAUTHORIZED USE IN THE PAST 12 MONTHS

This first trend immediately shows that careful interpretation is part of the effort. In this case, you could say that people who've no idea whether they had or did not have an episode of unauthorized use actually did or did not have such an event. On the one hand, we can say that unless you know you had an event then you did not ("optimistic"), while, on the other hand, we can say that unless you know you did not have an event then you did ("pessimistic"). This is illustrated in Figure 2.

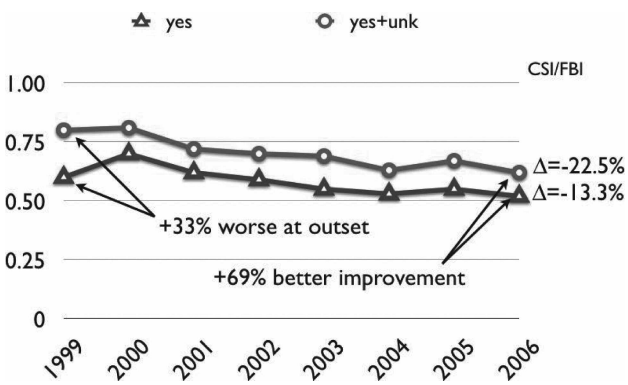


FIGURE 2: OPTIMISTIC V. PESSIMISTIC

This interpretation allows us to think about the problem a little bit deeper, since we now have the upper and lower bounds of assumption, given the data we do have, and we're reminded that the student who gets all As is never the student who gets the "Most Improved" award.

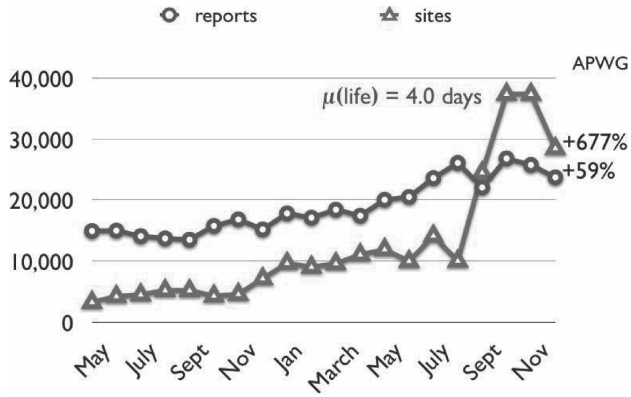


FIGURE 3: NEW PHISHING MESSAGES AND SITES PER UNIT INTERVAL

Let's look at something different: phishing (Figure 3). Data from the Anti-Phishing Working Group [3] shows a 19-month increase of 59% in the monthly reports of phishing email received but a 677% increase in the number of URLs used by phishers in those emails and that the lifetime of those URLs is 4.0 days. This tells us that the supply of URLs is no problem for our opponents and that our opponents cycle the URLs just fast enough to outrun the combined protection bureaucracy response (of consumer to fraud complaint to ISP to hosting center). That tells us that we have lost the supply-side battle, and we should plan accordingly.

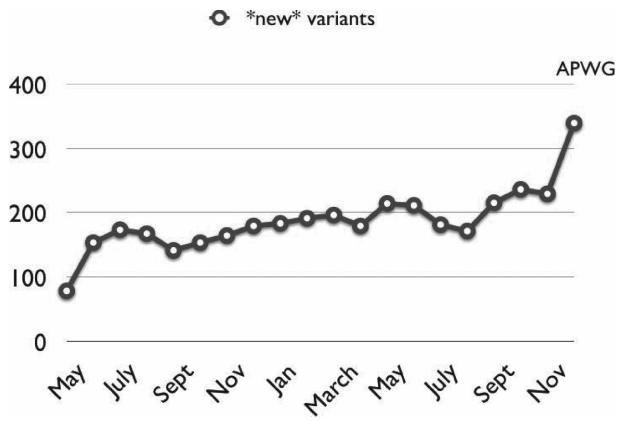


FIGURE 4: NEW DATA THEFT MALWARE VARIANTS IN PHISH EMAILS PER UNIT TIME

These days, phish email often comes with malware attached, and that is certainly a trend worth watching (Figure 4). Although those numbers are not sky-high, it is important as a future-of-security planning exercise to remember that if you are trying to recognize these malware-carrying phish-

ing emails on sight, then your workfactor is the integral of all the phish mails to date, whereas the opposition's workfactor is the price of creating new ones. That, in turn, means that Figure 5 is more like what your force planning exercise has to contend with.

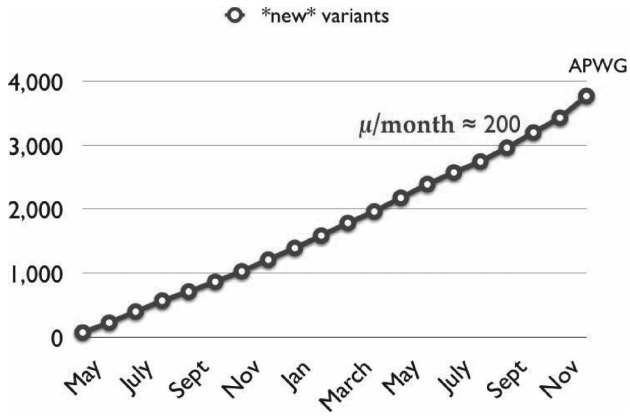


FIGURE 5: CUMULATIVE NEW MALWARE VARIANTS IN PHISH EMAILS

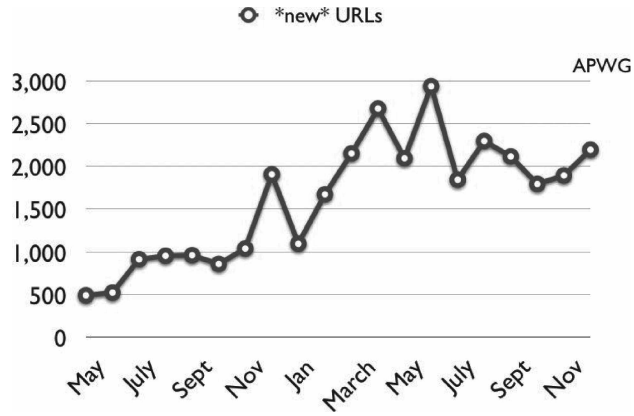


FIGURE 6: NEW DATA THEFT IN URLS IN PHISH EMAILS PER UNIT TIME

Of course, the same thing is true when we look at the URLs that the data theft malware will use if and when that malware succeeds. The month-to-month rate looks like that shown in Figure 6. Figure 7 shows the cumulative effect of Figure 6's rate.

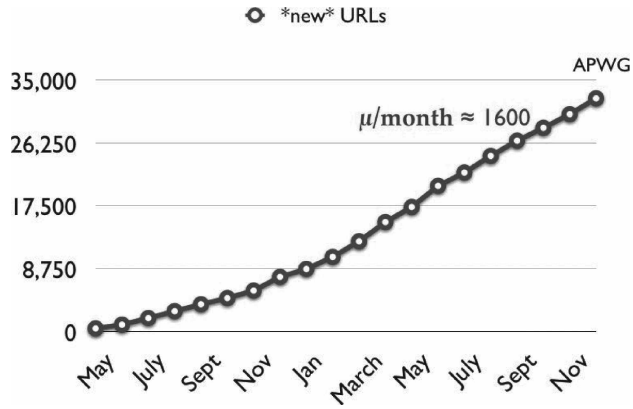


FIGURE 7: CUMULATIVE NEW DATA THEFT IN URLS IN PHISH EMAILS

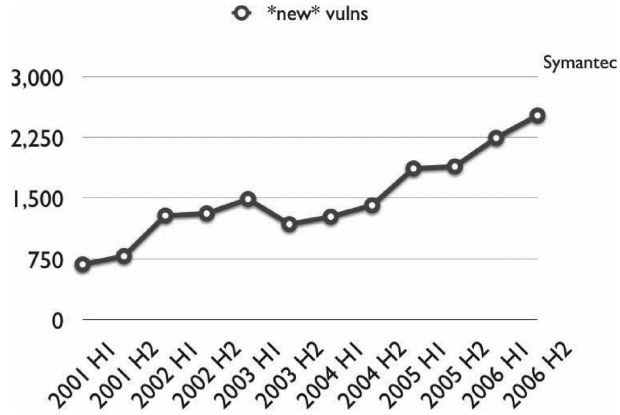


FIGURE 8: NEWLY REPORTED VULNERABILITIES PER UNIT TIME

Let's look at something different—vulnerabilities—and let's switch to Symantec data. Let's also remember that every software vendor is working harder and harder to keep vulnerabilities out of its code. In Figure 8, we can nevertheless see that in the most recent six-month reporting period a new high for identified vulnerabilities was reached. Now Symantec has only been publishing this in its Internet Security Threat Report [4] since 2001, and there have certainly been vulnerabilities around since before that. Even so, if we said that only Symantec hears about vulnerabilities and that there weren't any before 2001, we would have, between then and now, a 26-fold increase since record-keeping began (see Figure 9). Cumulative vulns may not be at the top of anyone's agenda but, in truth, vulns never really go away (they just get rarer, like car owners who never answer recall notices).

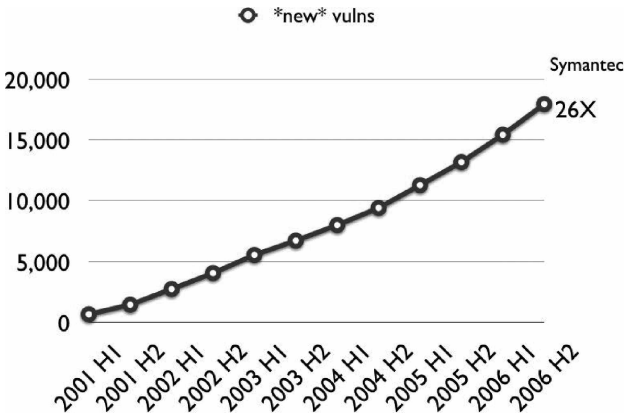


FIGURE 9: CUMULATIVE NEWLY REPORTED VULNERABILITIES

	2005	2004	2003	2002
OS	19	140	163	213
Net Stack	1	6	6	18
Non-Server App.	229	393	384	267
Server App.	88	345	440	771
Hardware	0	20	27	54
Protocol	12	28	22	2
Crypto	0	4	5	0
Other	0	10	16	27

TABLE 1: REMOTE VULNS REPORTED TO/BY NIST

But perhaps you are more interested not in total vulnerabilities but merely in remotely exploitable ones ("remotes"). In that case, NIST has some data for you [5], as summarized in Table 1. Table 1 is exactly as it was reported originally, but as a table it is not as informative as it might be. (Nonserver apps are, by the way, client tools such as Web browsers and email readers.) A better presentation is that of Figure 10 (in which the timeline goes from left to right and mass is displayed as area).

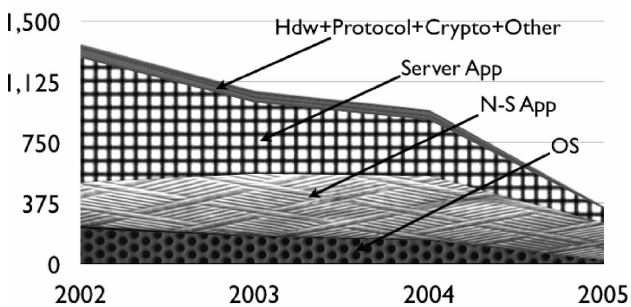


FIGURE 10: REMOTE VULNS BY SOURCE OVER TIME

Hardware	–73.5%
Other	–66.7%
Net Stack	–61.8%
OS	–55.3%
Server App.	–51.5%
Non-Server App.	–5.0%
Protocol	81.7%
Crypto	n.a.
Overall	36.0%

TABLE 2: COMPOUND ANNUAL GROWTH RATE (CAGR) BY REMOTE VULN TYPE

But although the display is more informative, it still isn't good enough. Perhaps it would be better to compute a compound annual growth rate for the various kinds of remote vulns, as listed in Table 2. Now that is more informative, especially as it tells you where progress is being made and where it is not. This might tell you how to redeploy your efforts, for example, but there is yet one more way to look at this, and that is as market share rather than counts. We first construct Table 3 and then use Table 3 to construct Figure 11, where it is now apparent that the action is becoming almost entirely about the nonserver applications. This is important for planning purposes.

	2005	2004	2003	2002
OS	5%	15%	15%	16%
Net Stack	0%	1%	1%	1%
Non-Server App.	66%	42%	36%	20%
Server App.	25%	36%	41%	57%
Hardware	0%	2%	3%	4%
Protocol	3%	3%	2%	0%
Crypto	0%	0%	0%	0%
Other	0%	1%	2%	2%
	100%	100%	100%	100%

TABLE 3: COUNTS OF REMOTE VULNS EXPRESSED AS MARKET SHARE

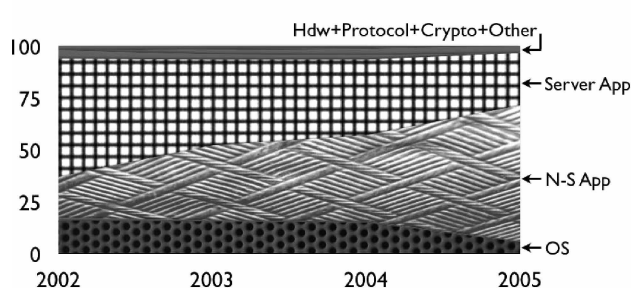


FIGURE 11: REMOTE VULNS BY SOURCE OVER TIME, EXPRESSED AS MARKET SHARE

Let's take a similar look at our very best friend, spam. Because so many people are interested in that topic, we have the luxury of several sources of data. In Figure 12, we have TQM3's take [6] on the volume. In Figure 13, we have Commtouch's take [7] on spam volume and in Figure 14, we similarly have Postini's take [8] on that volume of spam.

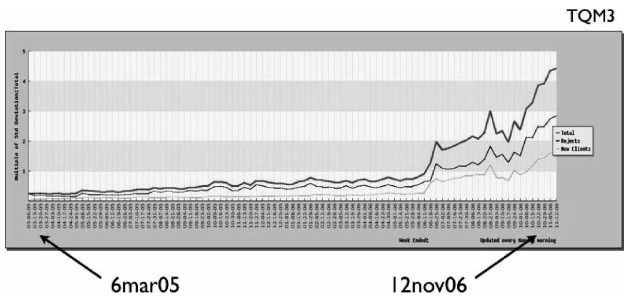


FIGURE 12: ONE ILLUSTRATION OF A SPAM SURGE

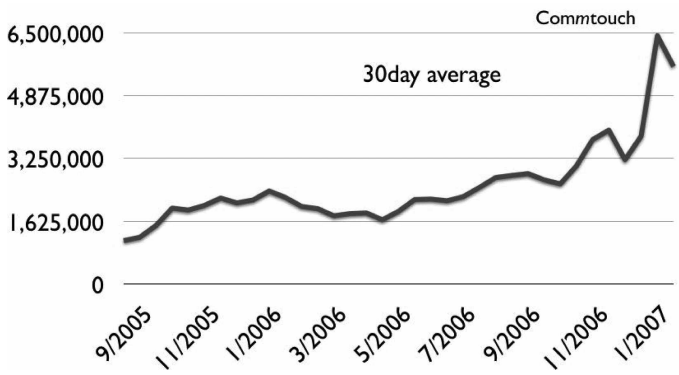


FIGURE 13: ANOTHER ILLUSTRATION OF A SPAM SURGE

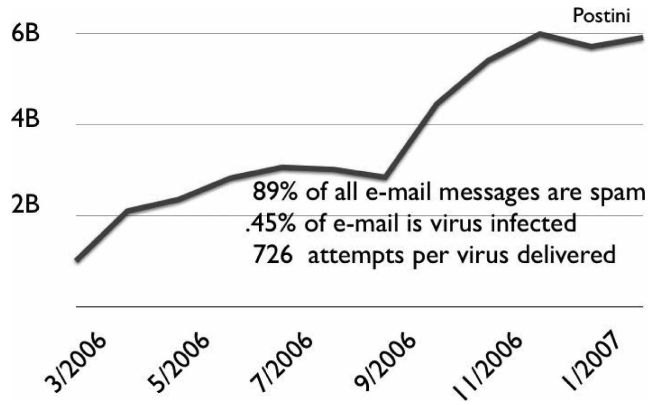


FIGURE 14: ANOTHER ILLUSTRATION OF A SPAM SURGE

It does look as though the trend is upward at not dissimilar rates. Postini's report of additional numbers is itself interesting. For example, it proves that economics lies on the side of the spammer who is trying to get the working attention of the recipient. In the direct mail advertising (junk mail) world, a response rate of 1 in 100 (1%) is considered a success and here we have 1 in 726 for what we can call response rate to virus transmission. The transmitter thus has 1/7 of the direct mail market's definition of success but has that for zero effective cost. The planning information to take from this result is simply that economics favors the opposition, but we also have a metric for how we are doing: whether the 726 number can be made to increase or not.

Incidentally, it is likely that total spam email volume is not rising (despite these three disparate charts) but, rather, that the percentage delivered is rising as template spam (for making individual messages unique) is progressively defeating Bayesian filters.

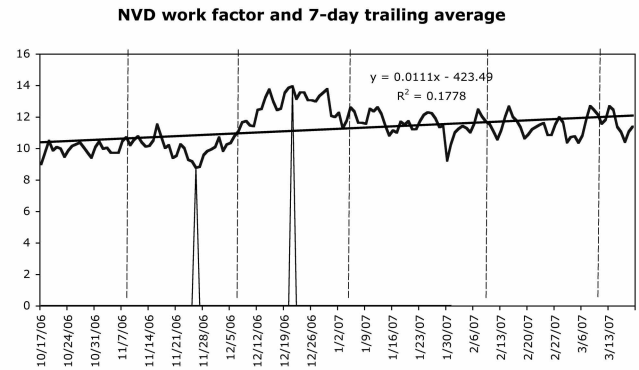


FIGURE 15: WORKFACTOR FOR SECURITY PRACTITIONERS

This is all obviously pointing at work for you, the reader, to do, but how much work? Interestingly,

the National Vulnerability Database folks calculate a daily number—the "workfactor" number. In Figure 15, we see several months' worth as the raw number and a fitted line. Among other things, the fitted trend line is rising, which, as a planning mechanism, says that the workfactor of people dealing with security problems is climbing. The vertical dashed lines are the days on which Microsoft releases its monthly bolus of problems to attend to. The spikes point to the minimum (the Sunday after Thanksgiving) and the maximum (the next-to-the-last shopping day before Christmas). One can be perhaps forgiven for suggesting that this would be consistent with an all-out assault on the Internet Christmas shopper this past season.

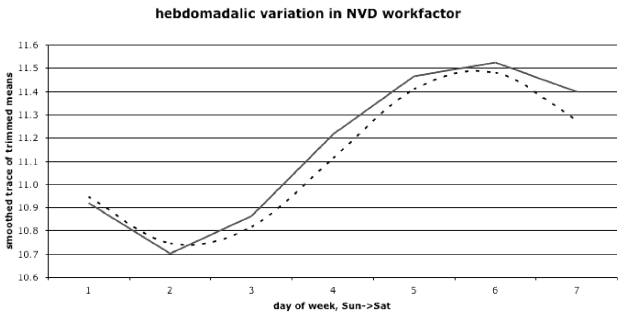


FIGURE 16: EVIDENCE OF A WORKWEEK HIDING IN THE WORKFACTOR DATA

But there is something interesting hiding in these numbers if you look at them a different way: It appears (in Figure 16) that there may be evidence of a conventional workweek. And if the opponent is actually enjoying a conventional workweek, there is perhaps no further need for corroboration that exploiting security problems has become the day job for some number of people. Yes, the dotted line is a sine curve and it does fit pretty well. In fact, we see corroboration of a workweek in Symantec's numbers for the daily appearance rate of unique phishing emails (Figure 17).

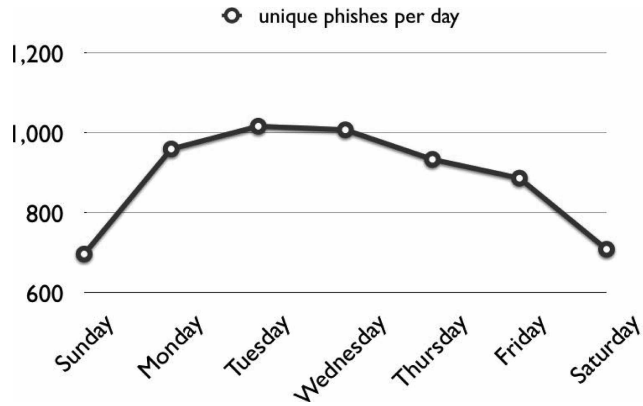


FIGURE 17: NUMBER OF UNIQUE PHISHING EMAILS ON WEEKLY CYCLE

Everyone rightly worries about spyware, trojan horse programs, and especially such nasties as keyloggers. With help from Webroot [9] we can quickly see that if an enterprise PC has any spyware then it probably has more than one example (Figure 18). We can see that trojans are plentiful (Figure 19) and we can see that if an enterprise PC has any trojans then it probably has more than one example (Figure 20) or, even more worrying, that if an enterprise PC has a keylogger then it could well have more than one example (Figure 21).

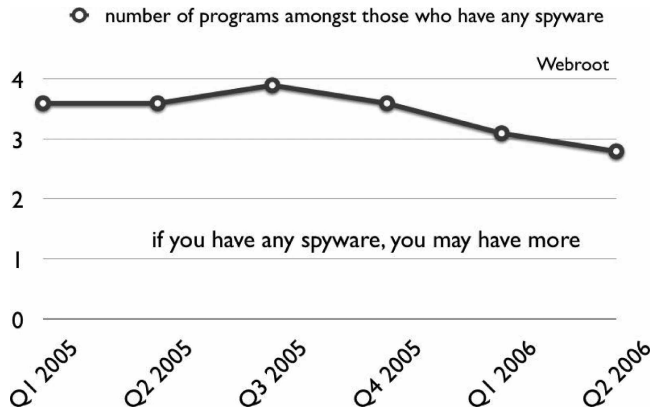


FIGURE 18: NUMBER OF SPYWARE EXAMPLES PER ENTERPRISE PC THAT HAVE ANY

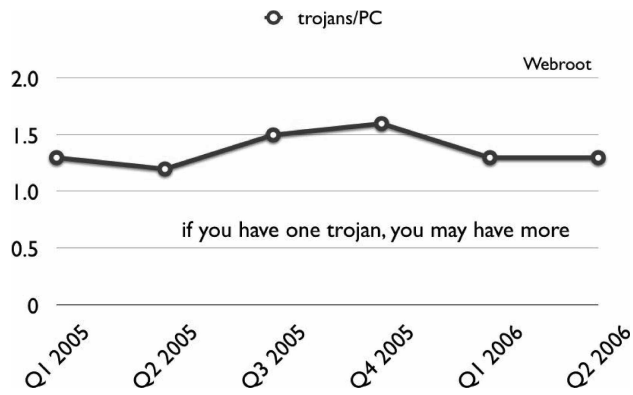


FIGURE 19: PERCENTAGE OF ENTERPRISE PCS WITH A TROJAN

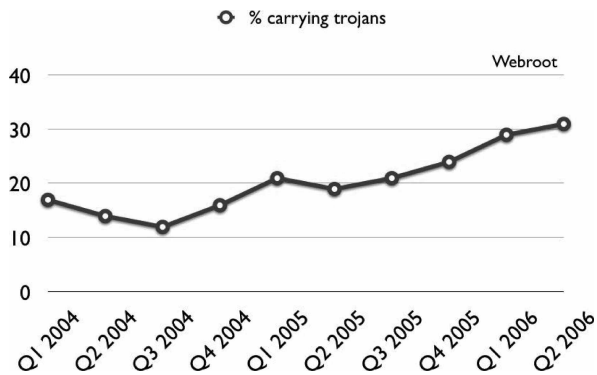


FIGURE 20: NUMBER OF TROJAN EXAMPLES PER ENTERPRISE PC THAT HAVE ANY

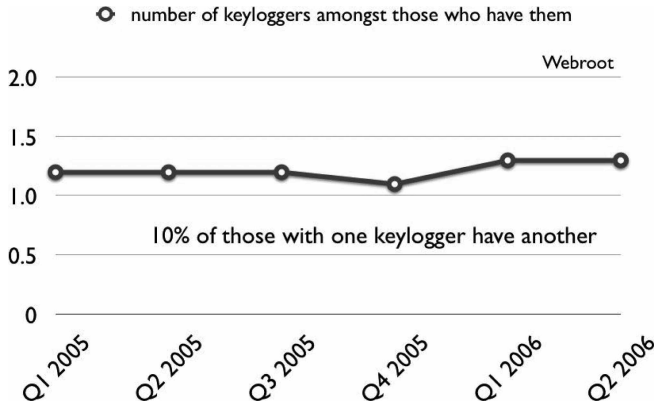


FIGURE 21: NUMBER OF KEYLOGGERS PER ENTERPRISE PC THAT HAVE ANY

This apparent fact makes sense; users who do things that get them in trouble once will probably get themselves in trouble more than once, leading one to concur with Microsoft that having the ability to very quickly re-image a desktop may be an important part of any risk management plan:

> When you are dealing with rootkits and some advanced spyware programs, the only solution is to rebuild from scratch. In some cases, there really is no way to recover without nuking the systems from orbit.
>
> —*Mike Danseglio, Program Manager, Security Solutions Group, Microsoft, April 3, 2006 [10]*

Sometimes, though, you can make better decisions by understanding whether you are a target, *per se*. Using Counterpane's data [11], it is easy to see that where the money is is where the attacks go (Figure 22).

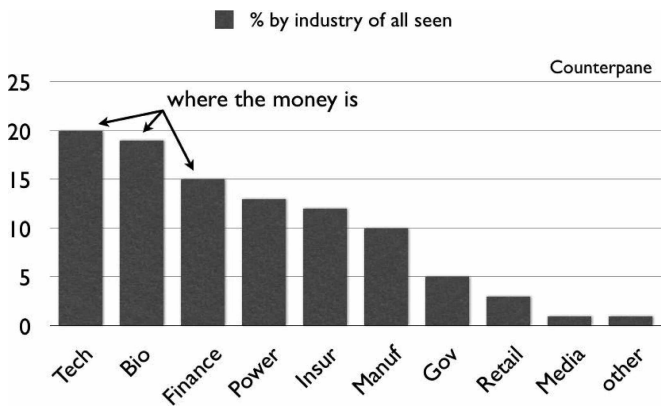


FIGURE 22: WHERE THE ATTACKS ARE IS WHERE THE MONEY IS, AND VICE VERSA

Perhaps your training leads you to think of the great mass of IT in the modern enterprise in the way a public health doctor views infection in a sprawling city. If so, figures like this might make you think:

- 318 new Win32 viruses/week
- 9,163 hosts/day join botnets
- 75% of malware is modular
- 1% of bots show themselves per day
- 5,900 phishing emails/minute

This, too, is part of thinking about the future so as to plan for it. In fact, by this point in this essay, perhaps we can hazard some inferences and identify some implications.

Where This Leads

Security and threat co-evolve, exactly in the same sense that predators are the reason prey diversify. Over time, and as natural immune systems get better, the pathogens that remain are fewer in number but are selected for virulence (the ability to move from host to host), and indeed we've seen that in ever-faster-spreading but ever-rarer epidemics of computer viruses and worms. We've seen that infectious agents rarely cross species boundaries, just like in nature. We know that corruption of the immune system is the worst (think of the "Witty" worm), and we know that parasites co-exist nonlethally with their hosts. (Some wags claim that home machines involved in botnets, except for being 0wned, are better managed than your average home machine.)

We also know that evolution's course is by punctuated equilibria [12] rather than through smooth gradual change. We are living just after such a puncturing of the equilibrium. Public access to the Internet began in 1990, and that access, followed in 1991 by the precursor of the browser, created an irresistible economic force for everyone to connect to this Internet thing. However, doing so suddenly created a world where both prey and predator were and are location-independent. In this new world, force multiplication is proportional to bandwidth, and bandwidth is cheap, almost (but not entirely) too cheap to steal [13]. That opening of the Internet also created the economic driver for commoditization of computers and that commoditization, absent any effective regulatory framework, led inexorably to monoculture and monoculture threat.

For better than three decades, the computing you can buy for a dollar has grown by 1% per week, the storage you can buy for a dollar has grown faster than that, and the transmission capacity you can buy for a dollar has grown even faster still. Over that same interval, total market capitalization, as measured by the Dow Jones Industrial Average, has grown by 1/7% per week, thereby

proving that data comprises a rising fraction of total corporate wealth. Of course, the value thus expressed is a magnitude and, to a large extent, the sign bit is separately determined, in part by security technology and security practitioners. Data has thus become the coin of the realm, being the repository of value for the general economy.

That data is increasingly mobile. The economically optimal computer is changing as we speak. When CPU price/performance doubles every 18 months, storage price/performance every 12, and bandwidth every 9, then for every decade one expects two orders of magnitude in computer power but three orders of magnitude in retained data and four orders of magnitude in data transmission. The implication of spending constant dollars on these three components of a computing infrastructure would thus mean that at the end of a decade the CPU would be only 1/10 as powerful per unit volume of data but the data, despite being 10 times as voluminous, would be able to completely move 10 times as fast. Coupling this with the close embrace of the Internet by commerce at all levels makes it clear that the winners will be those with as much information as possible in play, while the losers will be those who have too much, with security technology and practice providing the fine line between as "much as possible" and "not too much." This is already semi-present, as Gibson would say, with convergence of pure comms (telephony) and data-rich applications.

Data becomes our focus going forward. Security is what distinguishes data that has value from data that does not. Regardless of setting or metaphor, a rising threat requires any defensive perimeter to contract. This is true for the military, for wildebeeste, and for data. A contracted perimeter for data means a shift of focus of our arrayed protection technologies to individual data objects at their point of use. Operationally, data is at risk when it changes from at rest to in motion, a state change akin to evaporation. The point of use is where that state change occurs, and thus monitoring is the first priority because in the electronic world that which escapes your view is that which will escape your grasp (i.e., you cannot control what you cannot see). The single smartest thing any Cabinet Secretary has said in thirty years was Secretary of Defense Donald Rumsfeld's comment that it is the unknown unknowns that will kill you (and every journalist and pundit who made fun of it thus proved beyond doubt that they are innumerate). Security metrics therefore begin with certainty at the point of use.

There are lots of interesting but decidedly losing propositions for how to handle a future that is about data security:

- Perform content inspection. This can be defeated by Pig Latin, much less encryption.
- Use statistical anomaly detection. This defeats itself, as it creates an infeasible work-factor to damp out false positives.
- Look for signatures. Like antivirus programs, this is defeated by any enemy, as it is the Red Queen's own technology, "Around here, it takes all the running you can do to keep in the same place" [14].

No, the trends and the facts tell us that the engineering problem statement now facing us is data protection that is (1) inescapable, (2) invisible, and (3) future-proof. The rules of economics tell us that this is a minimax problem, meaning an optimization tradeoff between preventing trouble (anticipation costs) and cleaning up trouble (failure costs). The National Center for Manufacturing Studies perhaps illustrates this best [15]. Figure 23 shows that near-infinite spending on prevention does get near-zero spending on failure recovery, just as near-zero spending on preventing trouble risks near-infinite spending on failure recovery. The economically optimal point is the sum of the two curves, the minimum cost for the maximum protection—a "minimax" solution. Though not shown, as the degree of electronic collaboration rises, the failure costs at a given level of information assurance will rise, thus pushing the summed cost curve upward and rightward as the essentialness of electronic collaboration grows.

NCMS

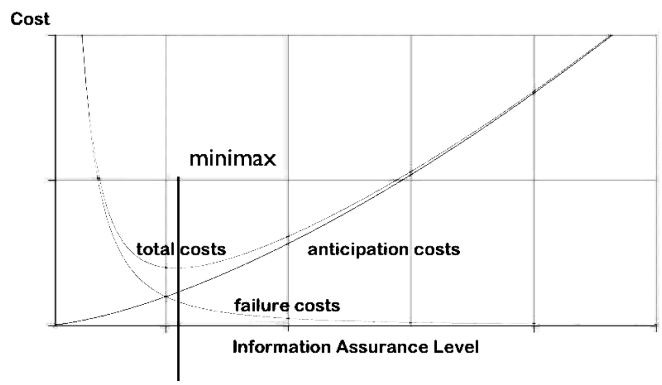


FIGURE 23: BEAR VS. AVOID THEM

Summary

In a sense, this essay should be unsurprising and it should feel unfinished. It is unsurprising, and it is unfinished. The trend data tells us that our opposition is gaining ground in an asymmetric war, a war where our costs accumulate and theirs do not. It tells us that our substantially increased levels of effort in protective armamentarium, in better prevention of vulnerabilities, and in improved detection of all sorts are proving not to be enough, as despite the rise in protective input there is a faster rise still in the capabilities of that which must be protected against. This calls out for the only advice there is: *If you are losing a game you cannot afford to lose, change the rules.* The rules we have to change are what it is we think we are protecting. A lost laptop is economically meaningless besides the data it contains. A single point of failure that must exist for absolute design reasons needs layers of defense-in-depth. Cascade failure cannot be cost-effectively prevented except by diversification when the efficacy of protections is, as these graphs show, falling despite the best efforts of good and honest people. Because data is where the value is, that is where the protections must go. If we are lucky, the worst tradeoffs we get are "DRM and privacy: both or neither." If we are unlucky, we get neither freedom nor security and neither privacy nor convenience, but the unluckiness will be because we failed to make necessity be the mother of invention. The trends are not good, but they are not yet a disaster. All of them have a consistent direction and tilt; what will be a disaster is if that direction and tilt continue, and that disaster will arrive far sooner than global warming.

REFERENCES

[1] William Gibson, author of *Neuromancer*, NPR interview, 30 November 1999.

[2] http://www.gocsi.com/forms/fbi/csi_fbi_survey.jhtml.

[3] http://antiphishing.org/reports/apwg_report_december_2006.pdf.

[4] http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_internet_security_threat_report_xi_03_2007.en-us.pdf.

[5] http://icat.nist.gov/icat.cfm?function=statistics.

[6] http://tqmcube.com/tide.php.

[7] http://www.commtouch.com/Site/ResearchLab/statistics.asp.

[8] http://www.postini.com/stats/.

[9] http://www.webroot.com/pdf/2006-q2-sos-US.pdf.

[10] Mike Danseglio, Program Manager, Security Solutions Group, Microsoft, April 3, 2006; http://www.eweek.com/article2/0,1895,1945808,00.asp.

[11] http://www.counterpane.com/cgi-bin/attack-trends4.cgi.

[12] N. Eldredge and S.J. Gould, "Punctuated Equilibria: An Alternative Tophyletic Gradualism," in *Models in Paleobiology*, edited by T.J.M. Schopf (Freeman Cooper, 1972).

[13] It is actually better to steal that bandwidth, since if you register a block of static addresses, then people will blacklist that block.

[14] L. Carroll, *Through the Looking Glass*, Chapter 2, 1872, replicated in the "Red Queen Hypothesis" in the study of co-evolution of parasites and hosts; for that see L. Van Valen, "A New Evolutionary Law," *Evolutionary Theory* (1973), vol. 1, pp.1–30.

[15] http://trust.ncms.org/pdf/CostInfoAssur-NCMS.pdf.

VASSILIS PREVELAKIS

supporting a security laboratory

Vassilis Prevelakis is assistant professor of computer science at Drexel University in Philadelphia. Over the past 12 years he has been involved in numerous security projects, both as a network administrator and as a researcher; currently, he is leading a project that aims to improve security for home networks.

vp@drexel.edu

MANY YEARS AGO, WHEN I WAS AN undergraduate student, when showing freshmen students around campus we would point to a large crater-like depression in the ground and say, "And this is the site of the old chemistry laboratory" and smile at the allusion to a horrible accident. The trick worked because people (especially freshmen) associate chemistry labs with explosions. However, running a security lab at the undergraduate level can also lead to "interesting" situations. Care must be taken so that the experiments do not disrupt the campus network or, heaven forbid, escape into the Internet.

Another question is what kind of experiments should be run so that the students derive real benefits from these labs. We do not want to teach students to become script kiddies, learning procedures by rote without really understanding what is going on. Moreover, students should have the means to evaluate their proposed solutions to problems that have been set out for them. In this way they reinforce their learning by actually putting into use concepts discussed in class. The labs, thus, do not replace the normal lectures but, rather, augment them. For the labs to be effective we need to ensure that students (a) actually spend time thinking about what they are doing rather than simply following some checklist, (b) learn concepts, rather than being trained in the use of specific programs, and (c) develop their ability to analyze complex situations and arrive at convincing solutions to problems.

At the Computer Science Department of Drexel University we have created a security lab environment and associated course work with the objective of meeting these aims. Our goal was to ensure that students could participate in lab sessions while also giving them the option of working with the security lab environment outside the lab sessions. Either way, students should be able to work independently without interfering with each other.

Experiments

Let us first discuss a number of experiments that we created for the lab and then we can describe the environment we created to run them.

We use two topologies to try out different experiments. In both cases we have three hosts (A, B, and C) that are used for the experiment, plus a fourth host (G), which connects host A to the campus LAN to allow students to exchange files with departmental servers. The first one (bus topology) is the traditional LAN layout, where all the hosts are connected on the same LAN (Figure 1a). The star topology shown in Figure 1b is mostly found in WAN situations, where A, B, and C are routers connecting internal networks together over long-distance point-to-point links

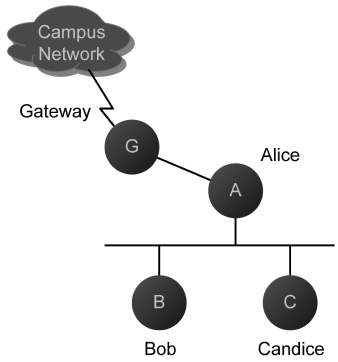


FIGURE 1A: BUS TOPOLOGY, WHERE HOSTS A, B, AND C ARE CONNECTED ON THE SAME LAN

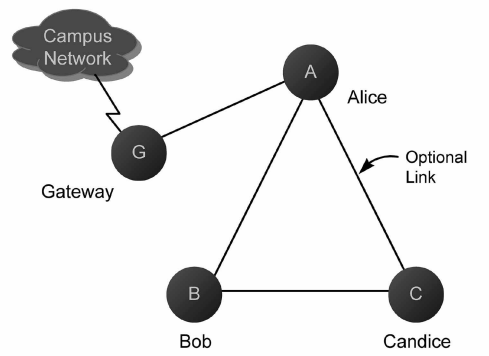


FIGURE 1B: STAR TOPOLOGY, WHERE A, B, AND C ARE EACH CONNECTED TO TWO OTHERS VIA SEPARATE LANS

EXPERIMENT 1: ARP SPOOF

In our first experiment we want to fool Bob into talking to the wrong DNS server and we do this by installing a fake DNS server on Candice and performing an ARP spoofing attack on Bob. The main purpose of this experiment is to show how protocols lacking authentication (such as ARP and DNS) can be subverted, but it also serves to familiarize the students with the ways raw packets can be generated by user-level applications.

This experiment has three stages: installing the fake DNS server, constructing and running the program to carry out the ARP spoof attack, and troubleshooting the fake DNS server in order to complete the attack. Students are provided with a simple DNS server replacement (dproxy) and they have to configure it on both Alice (the "valid" server) and Candice (the malicious, or "fake," DNS server).

We chose dproxy because it is a very simple DNS proxy server. Although dproxy listens for DNS requests in the same way as a usual DNS server (e.g., named), rather than resolving the queries itself, it simply uses the resolver library. The big advantage of this is that dproxy can answer queries by looking at the /etc/hosts configuration file, so we can easily add a new entry (e.g., www.drexel.edu, but even one belonging to a bogus zone such as www.priv) by editing the /etc/hosts file.

Students start the experiment by setting up dproxy on Alice and setting up Bob to use Alice as its DNS server (i.e., adding Alice to the /etc/resolv.conf file). They also configure dproxy on Candice and add two bogus entries pointing to itself: one for Alice and a second using a fictitious domain (www.priv). For example, by assuming that Candice has IP address 192.168.100.3, the new entry in /etc/hosts will look like:

```
192.168.100.3 alice www.priv
```

Students then run a few queries (nslookup and ping) to ensure that they have correctly configured their machines and establish a baseline for obser-

vations. They then activate the ARP spoof program on Candice and carry out the same queries, observing that while Bob now sends its requests to Candice, the request fails because dproxy uses Candice's source IP address and not Alice's. Students have to solve this problem and run the complete spoofing operation, fooling Bob into thinking www.priv really exists.

EXPERIMENT 2: ROUTING/FIREWALL

The triangle topology in Figure 1b is used as the basis for a WAN scenario where A, B, and C are routers connected via point-to-point links.

In the routing experiment students run a routing protocol (we used RIPv2 because it is the easiest to configure) and observe how they can divert traffic to Candice by injecting routes. The fake DNS server from the first experiment is also used here to exploit the redirection.

By removing the link marked "optional" in the diagram, the same topology can be used to create a configuration where B can serve either as "man-in-the-middle" or as a firewall. In the man-in-the-middle scenario, students observe packets going through B to spy on communications between A and C. For example, we ask students to use telnet to log onto A from C, while running tcpdump on B. Then students extract the log-in password from the packet traces.

In the scenario where B is an IP firewall, students design various configurations showing how B can filter packets, perform network address translation, etc.

Making All This Happen

Running these experiments in a safe manner while allowing an entire class of students to work at the same time during the lab session has been a daunting task. About five years ago we installed a rack with about 20 computers interconnected via a large switch with more than 100 ports. Each computer had 3 or 4 Ethernet interfaces, and they were all connected to the switch. By partitioning the switch ports into independent groups (VLANs) we could create various interconnection topologies for the rack machines (Figure 2). On each machine, one of the interfaces was reserved for management, allowing network access to the machine regardless of the configuration of the other port interfaces. As a last resort, serial access to the console ports of each machine was also provided.

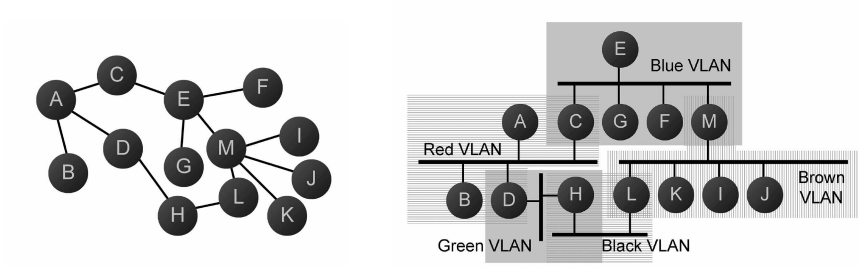


FIGURE 2: VARIOUS TOPOLOGIES (LEFT DIAGRAM) CAN BE REPRESENTED BY CONFIGURING VLANS ON THE ETHERNET SWITCH (RIGHT DIAGRAM)

However, the main problem with this approach was that reconfiguring the switch was extremely laborious and error-prone, and, to make matters worse, we had to provide special configurations for each machine.

Scripts implementing canned configurations for the switch were developed and the rack machines were configured to boot from the network and use NFS for their file systems, but still there were problems. For example, if we allowed students to have access to their home directories on the departmental server they would also have access to the files of other students. Although NFSv4 supports user authentication, the version of NFS we had at the time supported client-side authentication, so it was an all-or-nothing solution. In addition, the number of the machines was inadequate for the size of the class so we had to split students into groups. Finally, the students complained that they did not have access to the machines outside lab hours, so they could not work on their own.

VMWARE COMES TO THE RESCUE

To address the limitations of the hardware solution, about two years ago we started migrating our security lab to VMware. The students use a lab with Linux PCs connected to the campus LAN. There are sufficient PCs in the room for each student to have his or her own workstation.

The topologies described in the previous section were implemented using virtual machines (VMs) linked together via virtual networks (vmnets) that are included in the VMware product. The vmnets may be used to connect virtual machines together, to link them to the host workstation, or even to provide direct access (via a virtual Ethernet bridge) to the physical network. For the lab virtual machines we used exclusively host-only networks that do not allow direct communications with the outside network. For example, to create the layout in Figure 1a we created one host-only vmnet and linked all the virtual machines together. For the layout shown in Figure 1b we created virtual machines with two or three virtual network interfaces each and linked them together via three vmnets (one for each side of the triangle). Students use separate windows for each virtual machine and so can see output from all three VMs at the same time.

Hosts A, B, and C in Figure 1 are instantiated as separate VMs, whereas host G is the workstation hosting the VMware session. Each VM runs a complete installation of OpenBSD 3.8, allowing students to develop programs and test them in the target environment. Previously, programs that required administrator access to run (e.g., used raw sockets, low-numbered ports, etc.) could not be run in the common servers used by the department and many students could not run OpenBSD on their own PCs. This forced students to carry out program debugging during the lab sessions, which distracted them from the actual lab tasks. With VMware, students have the option of running the security lab environment on their own personal computers and can rerun the assignments on their own.

A big problem with running three virtual machines per student is that during the beginning of the class, 60 to 90 VMs are booted. Although the CPU load is not important because students are running VMware on their workstations, the file I/O load on the NFS servers is tremendous. This not only caused serious delays in the initial classes but caused many students to exceed their disk quotas as they started using the disks associated with their VMs. We addressed this problem by using a special feature of VMware called "non-persistent virtual disks." This allows a virtual disk to be read-only, but in a way that does not cause problems with the operating system running in the VM. Normally, an operating system expects its boot disk to be writable, so simply making this disk read-only is bound to cause problems. Instead, the VMware non-persistent disks allow full read/write

capability while the VM environment is running, but once the VM is shut down, all changes are lost. We can even reboot the guest OS and it will still see the modified image, as long as we do not restart the virtual machine environment. More information on non-persistent virtual disks is available on the VMware Web site [1].

Thus, we created three virtual disk images (one for each of the three machines in Figure 1) and we asked students to attach these images to their virtual machines and mark them as non-persistent. Since the volumes are read-only and belong to the teaching assistant, none of the students can attach these disks read/write anyway. Using non-persistent disks in turn necessitates providing some nonvolatile disk space that can be used by students to save their work. We addressed this issue by allowing each student to create another virtual disk, which is stored in the student's home directory and is only big enough to contain the student's personal files. The second virtual disk can be mounted at any place in the file system (both manually and automatically during boot), so its existence can be completely transparent to the student.

Another advantage of using common virtual disks for the operating system is that new VMs or changes to the configuration of existing machines can be added quickly and applied to all students at the same time. This makes it possible to create on short notice new experiments to demonstrate a new technique or to provide an example for something discussed in class. For example, in order to get students to carry out code injection attacks, we created a new VM with an old version of FreeBSD containing a number of vulnerabilities and asked students to come up with attacks. In this case, rather than all the students attacking one machine and thus potentially interfering with one another, we had each student boot a private VM with the vulnerable system and attack it at leisure.

Running the Labs

With the environment ready and the lab sessions created, a key question was whether to run them as homework assignments or as actual lab sessions. The former has the advantage that students can go through the assignments in their own time, and it also reduces the logistics associated with the lab sessions (booking a room with 30 workstations, making sure that VMware works correctly on every station, etc.). Moreover, students prefer to be able to work on the lab sessions outside the (limited) lab hours. Nevertheless, we feel that carrying out the experiments during the lab sessions is very important as students who are stuck can be nudged toward finding the solution. Otherwise, students will simply get the solution from fellow students and apply it blindly just to get on to the next question rather than solving the problem.

The next question is how to structure the experiments and, more important, how to phrase the instructions and questions to ensure that students do not simply look ahead to the next steps in order to deduce the answer to the question. This forced us to think about some way to prevent students from looking ahead before they answered the questions.

Our first approach was to use an overhead projector with slides describing one question at a time and wait till everybody had answered it before moving to the next one. This failed miserably, as students had to wait for the slowest one to finish and hence either students were bored or slow students were hurried along (or just given the correct answer so that the class could go on). Also, students could not go back and look at previous ques-

tions or review information given out earlier. We briefly tried handing out the assignments in printed form, one question at a time, but this meant that the lab assistants were spending more time distributing sheets of paper than answering questions.

Finally, we decided to bite the bullet and use an on-line course work system (WebCT). Each lab exercise is encoded as an online quiz that prevents students from changing submitted answers to questions. In this way students may proceed at their own pace, but since they receive no points for skipped questions, they have a powerful disincentive to peek ahead.

Unfortunately, WebCT is a very temperamental system with a lot of obstacles for casual users. For example, at one time, during the lab we found that the text boxes that students would use to type in their responses were limited to 100 characters. Since the lab was already in progress, we had no way (or clue) how to fix this, so students were forced to be brief. (This is not such a bad thing in retrospect, but one student remarked that the most challenging aspect of some questions was the requirement that the reply should be fewer than 100 characters.) Having used WebCT about 10 years ago, I wish I could go back to that older version, which, while lacking all the bells and whistles, actually let the user be in control.

Lessons Learned

- Using non-persistent virtual disks for booting the VMs and for storing the bulk of the data needed for their operation is a clear winner. However, there have been instances where students lost work when they shut down their VMware session without copying their work to their private persistent partition. We are investigating various techniques for reducing this risk. For example, on virtual machine B (the one used for the firewall and man-in-the-middle experiments) we have moved most of the system configuration normally stored in /etc to the private partition. Initially, students copy an existing partition (with the configuration of B) to their home directory and attach this copy as the second disk drive on host B. (OpenBSD sees this partition as wd1a.) Since they own this partition, they can make changes to it and these changes will persist across VMware sessions. Students can always return to the initial configuration by copying the shared partition over their private copy, thus destroying all the changes they have made. Of course, once students start working on their private copy of the /etc directory, we lose our ability to change virtual machine configurations globally. This is why we provide this capability on only one of the VMs.
- The extremely rich environment supported by UNIX sometimes works against us, as we cannot create a platform that contains all possible editors, shells, development environments, etc., that students are used to working with. Students thus often have difficulties because they are not familiar with a particular utility. For example, when we created the original environment, the default installation of OpenBSD did not include the emacs editor, and some students complained that they did not know how to use vi to make changes to various files. Since the lab environment is currently used only for the security course, students cannot be expected to spend much time customizing their environment or learning how to live with its peculiarities.
- Support for graphical user interfaces (GUIs) is needed. Programs such as Ethereal offer powerful ways of representing and managing captured data using a GUI. We believe that students would benefit from the use of such programs, but the current lab environment only supports char-

acter-based consoles. Allowing the X11 window system to run on the virtual machines is not difficult, but it causes a lot of headaches, such as performance degradation; worse configuration issues than those discussed earlier, as students are forced to live with potentially different window managers, X11 settings, etc.; and more lab assistant time to help students.

- As practically all students now have powerful laptops or home computers, we are considering the possibility of asking students to install the entire environment on their own computers and use it to carry out their regular assignments (i.e., use the security lab environment for all security course homework). Unfortunately, VMware currently runs only under Windows and Linux, which means that some students (e.g., Apple users) will not be able to use the virtual environment. Staff limitations make supporting multiple VM environments difficult, but we hope to be able to support Parallels on the Mac in the near future.
- Despite the WebCT-related setbacks, we find the on-line quiz format the best solution so far, but we are looking for alternatives to WebCT.

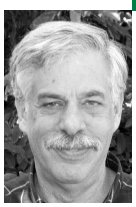
The security lab environment is a work-in-progress, as we always find things that can be improved and we constantly add new experiments or fine-tune existing ones. Despite the numerous issues we have had to address over the past four to five years, students enjoy the labs and we find (through exams and continuous assessment) that their understanding of security concepts has been improved by the lab experience. We believe that the same environment can be adapted for use in other systems courses, such as computer networks and operating systems, and we are planning to support such courses in the future.

REFERENCE

[1] http://www.vmware.com/support/gsx25/doc/disks_modes_gsx.html.

DANIEL L. APPELMAN

spam and blogs



PART 2: BLOGS, FOR GOOD OR ILL

Dan Appelman is legal counsel for the USENIX Association and practices technology law as a partner in the Silicon Valley office of Heller Ehrman LLP.

dan@hewm.com

This is the second part of a two-part article based on a tutorial I gave on spam and blogs at the LISA '06 conference in Washington, D.C., in December 2006. The first part, on spam, appeared in the April 2007 issue of *;login:*.

THE EVER-INCREASING USE OF THE Internet creates new challenges for the system administrator. Many of these challenges have legal dimensions. This is particularly true of spam and blogs. By some estimates, over 75% of all email traffic is spam. In the United States and abroad, laws have been enacted to regulate spam with various remedies and varying success in encouraging compliance. Blogs are increasingly used not just for personal expression but also for commercial purposes. It often falls to the system administrator to design and enforce company policies to protect against spam, to limit personal use of blogs using company facilities, and to ensure that the company is in full compliance with all applicable laws and regulations.

Blogs have become increasingly popular and easy to use just in the past few years. Unlike spam, however, no body of laws has arisen that specifically addresses or attempts to regulate blogging. This may be because blogs, unlike spam, do not inherently impose themselves on unwilling recipients. Blogs are more passive: One can read them or ignore them. Readers must search them out, whereas spam arrives in one's email inbox unsolicited.

To a great extent, blogs are simply publications like any other. The innovation in blogging is that the Internet reduces the cost of publication to virtually nothing while it enables distribution to the widest possible audience. Blogging enables everyone to be a publisher and provides everyone with a means of reaching an audience with their ideas and opinions.

Blogs raise the same legal issues that are common to all other publications. This article focuses on several of the most important of these: defamation, intellectual property infringement, and employer-employee relations.

Defamation

Defamation is a civil law cause of action where one person sues another because some statement made by the other has caused damage to his or her reputation. To be actionable, the statement

must be false, the person making it must know it is false, and the statement must actually result in injury to the reputation of the person suing.

Often, defamation arises in the context of the publication of false statements by the media, such as newspapers or radio or television broadcasting stations. United States Supreme Court decisions shield the media against suits by public figures unless they can show that the defamatory statements about them were made with malicious intent. However, plaintiffs who are not public figures only have to show that the injurious statements were made with a knowledge of their falsity.

There is every reason to believe that the same standards applied to the media would apply to bloggers. The constitutional right of free speech will usually protect bloggers, even if they make false statements about others. But the balance tips toward protecting the reputations of others against those false statements where the blogger knows those statements are false and, in the case of public figures, where the blogger has malicious intent.

Intellectual Property Infringement—Copyrights

Federal copyright law protects authors against unlawful copying of their articles or other works. Obviously, copying an entire article written by someone else and republishing it as one's own would be a blatant case of copyright infringement. Even republishing it and giving the true author appropriate attribution would still constitute infringement unless the author gave his or her consent.

The more difficult cases involve the "fair use" doctrine. That doctrine balances the author's right to control the publication of his or her work with the importance of a free press. For instance, journalists are permitted to republish portions of copyrighted works without obtaining their authors' consent, in the interests of serving this social objective.

However, the fair use doctrine has its limits. Even journalists cannot republish whole works without their authors' consent; and the availability of the privilege will be balanced against other considerations, such as the purpose and character of the copying, the nature of the copyrighted work, the amount and substantiality of the copying, and whether it will deprive the author of the value of the original publication.

Blogs are publications, and the fair use doctrine should be as available to them as to the publications of the mainstream media. Still, the rights of bloggers to copy and reproduce works of others is not absolute. Publishing without attribution, copying whole or major portions of articles, republishing for commercial gain, and republishing that deprives the author of income are all highly suspect.

A common practice among bloggers is to link to third-party content instead of copying it. A few lawsuits have been filed alleging that linking to such content without the consent of the author of that content is a copyright infringement. In one of those lawsuits, Ticketmaster sued Tickets.com and lost. Ticketmaster also sued Microsoft on the same issue, and the suit was settled before a decision could be reached. Thus far, no court in the United States has ruled that linking to another's Web site content without the other's consent is illegal.

Incidentally, some readers may wish to know how to "copyright" their blogs. Since copyrights arise from the creation of the work, blogs are protected by copyright without doing anything at all. No registration is required; neither is the inclusion of a copyright notice. Nevertheless, I advise my clients to put a copyright notice prominently on their blog home pages.

Intellectual Property Infringement—Trade Secrets

Another kind of intellectual property right that blogs call into question is the right of trade secrecy. In the United States, state laws make it illegal to misappropriate confidential information that gives its owner a commercial advantage from not being known to competitors. Publishing such information without the consent of its owner can certainly constitute a violation of these state laws and may even result in the loss of that information to the public domain.

Bloggers must be very careful not to reveal any trade-secret information to which they have access. The fair use doctrine will not shield them from liability for publishing this category of information, particularly if they were or should reasonably have been aware of the confidential nature of this information prior to its publication. Violation of the state laws protecting trade-secret information can result in substantial fines and in some cases the infringer can be required to reimburse the plaintiff's legal fees as well.

Employer-Employee Relations

No area has generated as much interest among bloggers as the issues that arise in the context of employer-employee relations. Bloggers write about their jobs and their employers. They also blog about unrelated topics during working hours, thus detracting from the attention they are supposed to be giving to their workplace responsibilities. And increasingly, employers are using blogs to promote their products and services and to communicate with their customers.

What rights do employees have to blog about their employers? Are there any limits to the free speech rights of employees? To what extent do employers have the right to prohibit personal blogging by their employees while on the job? And when can an employee refuse an employer's request to contribute to company blogs?

It is clear that employees can be terminated and even sued for revealing confidential information about their employers if that confidential information rises to the status of trade secrets. Information rises to the status of trade secrets when it gives the employer a commercial advantage in not being known by its competitors. Employees have a duty to safeguard the confidential information of their employers. Often, that duty is made explicit in a nondisclosure agreement that the employee must sign. But even without a nondisclosure agreement, that duty is imposed by law.

Individuals who come upon trade secrets without breaching any obligation to keep that information confidential may not be in violation of the law if they publish that information in blogs. However, it is difficult for employees to take advantage of this exception to liability. Employees who blog are particularly vulnerable to allegations of trade-secret misappropriation because of the access they have to the confidential information of their employers and the presumption that their duty to keep it secret is absolute.

Employees charged with trade-secret infringement by their employers often point to the "whistle-blowing" effect of publication of the illegal or unethical activities of their employers. There are laws that reward rather than penalize employees for disclosing their employers' secret, but illegal, activities. But these do not constitute legitimate trade secrets. Often, activities that the employee thinks are illegal are not, so the employee makes that determination at his or her own risk. The better advice for employees is to consult a lawyer before disclosing any confidential information about their employers.

The balance between employer and employee rights in the workplace differs greatly from state to state. Some states, such as California and New York, are very protective of employees and many others, such as Texas, are much more protective of employers. Generally, however, employers have a right to expect their employees to devote complete attention to their duties and responsibilities during working hours. Employees who engage in personal blogging during the working day can be terminated after they receive reasonable notice that those activities are inappropriate. The right to pursue free speech in the workplace is trumped by the employers' interest in maintaining a profitable and efficient business.

Some laws impose affirmative obligations on employers to regulate certain kinds of speech. Examples include content that may constitute harassment of other employees or that contributes to an unsafe work environment. Employers can also curtail actions by their employees to the extent necessary to protect themselves from suit by third parties for unlawfully revealing the third party's trade secrets and confidential information. But the employer's powers in these circumstances stem from laws of a more general purpose rather than laws aimed specifically at blogging.

Implications for System Administrators

Many system administrators are aware of increasing blogging activity by employees using their employers' systems. It is clear that the constitutional rights of free speech and free press are not unlimited. Those rights are always balanced against countervailing rights and interests, such as the right not to have one's reputation injured by false statements, the right to protect one's intellectual property, and an employer's right to the undiluted loyalty and focus of its employees.

Employers should develop policies that address employee blogging. At a minimum, these policies should prohibit publishing content that harasses other employees or damages their reputations and content that jeopardizes the employer's intellectual property rights. Further, these policies should address whether blogging using the employer's facilities is even a permitted activity.

It is often useful for employer blogging policies to be developed with the input of the employer's system administrators, since they are the ones who

are frequently tasked with monitoring compliance with those policies. At a minimum, system administrators need to be familiar with those policies and given the tools and authority to enforce them. The system administrator should question any policies that seem to be overreaching or that do not adequately address any of the issues described in this article.

As blogging becomes more prevalent in the workplace and outside of it, system administrators will often be expected to inform their employers of any activities that appear to be illegal or that may violate established employer policies. Often it will not be easy to make the initial determination. Clear policies will help, but the system administrator will always be working in an environment of substantial ambiguity and legal uncertainty. Asking for help from the employer or the company's legal counsel at appropriate times will relieve the system administrator from the full burden of determining which blogging activities are appropriate and which are not.

LinuxConf Europe 2007

UKUUG

German Unix User Group

At the start of September 2007, a series of Linux events will take place in Cambridge, UK. From Sunday, Sept. 2, to Tuesday, Sept. 4, there'll be a significant new technical conference, incorporating the best elements from the UK Unix User Group's Linux Developers' Conferences (2006) and the German Unix User Group's Linux-Kongress (2006).

This event will be followed by the invitation-only 2007 Linux Kernel Developers Summit, sponsored by USENIX, on Sept. 4–6. Python enthusiasts may also wish to attend PyCon UK in Birmingham on Sept. 8–9.

ALEXANDER MUENTZ

script kiddies with briefcases

Alexander Muentz is both a sysadmin (since 1999) and a lawyer (admitted to the bar in Pennsylvania and New Jersey). He'd love to be a hacker public defender but has to earn his living helping law firms do electronic discovery. When he's not lawgeeking, he tries to spend time with his wife and his motorcycle.

lex@successfulseasons.com

THE LEGAL SYSTEM AS THREAT

SYSADMINS HAVE TO KNOW MORE than just their systems to do their jobs well. You have to understand people, finance, and your organization's business to spend its money wisely and protect it from all sorts of threats. IT people tend to think of threats using a preparation /attack/response framework, even when the threats are not malicious. You can then compare risks and allocate resources rationally. If threats can affect your systems, your users, or their data, they should be on your mind, even if most of the responsibility rests on someone else.

So why not think about legal processes in the same way? Some resemble existing attacks (having your systems seized is like a DoS attack), whereas others are unique. I'll discuss search warrants, wiretaps, subpoenas, and discovery orders and their effects on users, systems, and data. Because this is a significantly deeper subject than can be explained in a small book, let alone a single article, I'll just be glossing over important nuances to give you an overview of the issues. This article only discusses U.S. federal law, but many state laws resemble federal ones. This isn't legal advice so much as it is the start of a conversation that you can finish with your IT staff or legal counsel. Let's talk about a few threats.

The Search Warrant: Noisy, Disruptive, and Potentially Destructive

A search warrant execution to an IT defender is like an invasion and a DoS attack wrapped up together. Law enforcement officers (LEOs) enter the area with enough strength to control the area and to prevent evidence or people of interest from leaving. Intimidation is a second benefit to such overwhelming force—scared people often unknowingly waive their rights.

A search warrant is a legal document issued by a neutral judicial officer such as a judge and specifies both the area to be searched and what is to be searched for and taken or seized [1]. There are some specifics about how the warrant is obtained, but I'll not delve into that, for two reasons: (1) they're rather complex (even to lawyers); (2) you can't stop or interfere with a search while it's being executed. Your concern should be limited to two

potentially conflicting interests: preventing disruption to your organization, and not waiving your rights.

It helps to focus on the three distinct phases of such a legal attack:

- Pre-attack: Defenses include redundant systems in multiple places, good backups, and an attorney on retainer who is familiar with your operations.
- During the attack: Either be quiet or be helpful. Protect your rights. Don't interfere.
- Post-attack cleanup: Transfer over to untouched systems and restore from backups. Attack the warrant and file suit for damages and/or return of equipment or data.

During the Search

LEOs with a warrant to search and seize electronic evidence have some discretion about the execution. They can take copies of the evidence, media, or entire systems that they believe contain what they're looking for [2]. Although letting them take forensic copies of your systems may be annoying, wholesale removal of several servers is far worse. There's an obvious temptation to assist the LEOs so that they don't feel the need to truck your entire server room back to their office. Generally, I'd recommend keeping the conversation to a minimum, both to stay safe and to prevent expansion of the scope of the search. LEOs can expand the search if you grant permission, which you may do during the course of the discussion. You may also unintentionally admit knowledge of or control over evidence, which may make you "of interest" to the LEOs. That's not a good thing. Ever.

Imagine the following hypothetical situation: A LEO arrives with a warrant to search system A1 for Alice's email. Bob is the sysadmin for A1 and A2. The LEO, while debating on whether or not to put A1 on a hand truck, asks Bob if he can look at Alice's files on A2 or Abe's files on A1. If Bob doesn't clearly say no, the LEO may start looking. Or, imagine that the LEO's warrant includes A3, a system on which only Alice has a login. If Bob, attempting to be helpful, knows Alice's password on A3 and gives it to the LEO, he's now opened himself up to possessing whatever is on A3.

I want to end this section with two final ideas of what to do during the search. First off, *do not interfere with the search*. Such behavior may subject you to several criminal charges, in addition to charges related to whatever the search was about. Second, have a witness or two available to watch. You may want an additional person who can say what happened during the search, in case your recollection disagrees with the LEO's.

After the Search

If you have good backups or alternate sites, you can recover or cut over, minimizing your total outage. If the LEO took any systems, your lawyer can sue for their return [3] and also attempt to exclude the evidence from trial if there's a flaw in the search warrant (evidence gained from a incorrectly obtained or executed warrant can be suppressed prior to its admission in court).

Wiretaps

IF YOU'RE THE TARGET

The attack profile: Quiet and incriminating.

- Pre-attack: Defenses include point-to-point encryption under your control.
- During the attack: There's no defense. If the wiretapping is done correctly, you won't know until it's too late.
- Post-attack cleanup: Suppress the evidence in court.

Wiretaps aren't just for phones anymore. LEOs may intercept IP traffic with a valid wiretap warrant or with the permission of the intended recipient [4]. I'm not going to discuss Foreign Intelligence Surveillance Act (FISA [5) wiretaps because I think that's still a moving target. I hope it will be rare for readers to be the target of a regular LEO wiretap (also known as a Title III [6]), but I think it's helpful to understand them.

A LEO with a warrant can request that a wiretap be placed somewhere on a network where it can acquire all the traffic sent and received from the target, without the target's knowledge. It is unlawful for the wiretap to acquire "innocent" traffic, and this responsibility falls to the provider of the network, not the LEO. Encryption can shield the communications as long as it takes to decrypt the traffic or acquire the key. If the service provider has the key, the service provider can be forced to divulge it with a court order. Although the key may be acquired with a search warrant or subpoena, at least the target is informed of the lack of privacy.

IF YOU'RE THE PROVIDER OF THE SERVICE

The attack profile: Confusion, stress, and expense.

- Pre-attack: Defenses include the ability to carve out traffic to any host or user.
- During the attack: Have network staff ready to assist the LEO.

Providers of electronic communication services must accommodate valid wiretap warrants, allowing LEOs to remotely acquire targeted traffic while ignoring innocent traffic [7]. The provider must carve out any requested IP traffic or communications and forward them to law enforcement. If the provider encrypts the traffic, it must also decrypt it or make the keys available as well. There's some controversy about who is a "provider" under this legislation, as well as whether or not the wiretap can be remotely activated without the intervention or knowledge of the provider. But such controversy is the subject of another article.

SUBPOENAS AND DISCOVERY

The attack profile: Slow but invasive.

- Pre-attack: Defenses include a data retention/destruction policy. You need to be able to search all of your storage for relevant documents.
- During the attack: Work closely with legal representation.
- Post-attack cleanup: Expect repeat requests and be ready to explain what you did.

Although there are some legal differences between subpoenas and discovery, the two are similar from an IT defender's point of view. They're both legal orders to provide information, and since more information is stored electronically, you're going to be asked to help out if your organization is served with a subpoena or is a party to a lawsuit. Being able to retrieve information quickly and to honestly claim that you've searched all your files will make you golden.

First off, a sensible data retention policy is important. Talk to your counsel about what you have to keep and for how long. Be willing to explain how backups work in layman's terms. Once you have a retention policy that you can live with, follow it. If it says keep data for no longer than three years, make sure you've erased or destroyed older media. This dovetails into the second half of your pre-attack defenses. Knowing what you have prevents expensive mistakes.

A short war story is worth recounting here: When I was a backup admin, I was helping an outside law firm in searching through our archives. After searching the backups that I knew about, I gave them all the documents matching the search terms the form provided. Imagine my surprise when someone found a crate of DLTs from before my time, but within the scope of the request. The outside counsel almost had a heart attack. Lucky for us, the tapes had been stored above a Nuclear Magnetic Resonance machine, and they were all blank. Discovering relevant information later makes you look dishonest, and hiding it would have made us dishonest.

SUBPOENAS AND DISCOVERY: WHERE THEY DIFFER

Subpoenas do the heavy lifting in legal investigations. Grand juries, regulatory agencies, and courts can all issue them. The two basic types of subpoenas are *Duces Tecum* (bring us stuff) and *Ad Testificandum* (come and testify under oath). You can also get one if your organization has information necessary to resolve a lawsuit between other people. Dealing with a subpoena is less disruptive than a search warrant—at the minimum, you usually have a few days to respond. If the request is difficult to comply with, counsel may be able to narrow the scope to make it easier. (Don't you wish you could do that with your other projects?) Unfortunately, not too much is protected from a subpoena other than trade secrets and some client-attorney communications (for example, Rule 45 of the Federal Rules of Civil Procedure protects trade secrets and communications normally under some privilege). As long as it isn't abusive, overly burdensome, or not likely to lead to relevant evidence, it may have to be brought to the issuer.

Discovery entails sharing of relevant information between parties. Simply put, if you are in a lawsuit, you have to give the other side any information (discovery) you have that may help your adversary's case. You also have a duty to preserve any of that information when litigation becomes likely. So does your adversary. The only relevant information that you can withhold are some attorney-related files. Under the new Federal Rules of Civil Procedure, litigants must either hand over discovery that is in electronic storage or divulge the nature and location of that discovery fairly soon after the lawsuit is filed [8].

WORKING CLOSELY WITH YOUR LEGAL COUNSEL

When your organization has been served with a subpoena or discovery request, someone has to collect all the information that "responds" to the request. You'll be collecting a lot of it, and counsel will be reviewing it to see whether it's responsive. Your lawyers will be pulling long hours looking through each file you bring them. They'll need help with viewing, sorting, and interpreting the mountain of data. This may require them to hire temp workers or to export the files to their outside law firm. Being helpful here will let your organization save money and effort.

You may be called on to explain what you did in collecting the information, either to opposing counsel or at a grand jury or trial. If this is a discovery request, there are a few additional ways you can help. First, you can identify information that you hold that is very time-consuming or expensive to deliver, such as data on obsolete or failing media [9]. Counsel can go back to the court and exclude or reduce the amount of such discovery.

Conclusion

In closing, knowing what I just told you might allow you to be proactive when dealing with your organization's legal department, instead of waiting for the lawyers to come and impose difficult rules upon you. Let them know that you want to coordinate defenses and protect your users and the organization as a whole. They may be pleasantly surprised.

REFERENCES

[1] U.S. Constitution, Amendment 4.

[2] "Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations," CCIPS, Department of Justice (available at www.cybercrime.gov/s&smanual2002.htm).

[3] Federal Rules of Criminal Procedure, 41(e).

[4] 18 U.S.C. §§ 2518, 2511(2)(c).

[5] FISA (50 U.S.C § 1801 et seq).

[6] Title III of the 1968 Omnibus Crime Control Act. It's been modified by subsequent legislation, including the Stored Communications Act (18 U.S.C. §§2701-2722), Electronic Communications Privacy Act (18 U.S.C. §§ 2501 et seq) and a few others you may have heard of.

[7] CALEA, 108 Stat 4729.

[8] See FRCP 26(a)(1)(B).

[9] See FRCP 26(b)(2)(B).

DAVID BLANK-EDELMAN

practical Perl tools: impractical Perl tools

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the book *Perl for System Administration* (O'Reilly, 2000). He has spent the past 20 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

I'M WRITING THIS COLUMN RIGHT around the unofficial U.S. holiday of April Fool's Day. I realize you won't be reading it until several months after this day has passed, but pretend the U.S. Congress was so enamored with how well the daylight-saving-time rules changes went that they began moving around other dates on the calendar willy-nilly.

Given that you are now reading this column on the new date for April Fool's Day, I now have the license to look at some of the least practical of the Perl modules available. However, despite my best attempts to the contrary, I fear you'll probably learn something practical from this exploration (but shh, don't tell anyone!).

The mother lode of impracticality in the Perl world is the Acme:: namespace. In the gaggle of modules whose name begins with Acme:: you can find modules that extend the language in weird and wacky ways or play around with various programming ideas. These modules range from the exceptionally clever to the downright stupid. But sometimes even stupid code examples can teach us something.

Cleaner, Brighter Code

The progenitor of the Acme:: namespace is the module Acme::Bleach (originally just called Bleach.pm) by Damian Conway. Written in slightly cryptic code (there's a good explanation of it at http://www.perlmonks.org/?node_id=270023), the Acme::Bleach module takes a script that consists of perfectly ordinary looking Perl code like:

```
use Acme::Bleach;

open my $EXAMPLE, '<', 'example.txt' or die
"Can't open example: $!\n";
   while (<$EXAMPLE>){
     print;
   }
   close $EXAMPLE;
```

and transforms the file into:

```
use Acme::Bleach;
```

(plus another 107 lines of carefully selected whitespace characters)

That file, believe it or not, actually runs and does the same thing as the original program. This idea of a program that could rewrite itself into an obfuscated form that is self-deobfuscating on the fly quickly caught on in the Perl community as a fun thing to try. Now there are a number of modules like this available. Some of them are actually bordering on being useful.

Let's look at two examples. (1) Acme::PerlTidy runs the Perl::Tidy cleanup process on itself every time it is run, thus assuring your code is always as readable as possible. (2) Acme::PerlML takes your perl code and translates it to an XML representation. For example, the sample code above (substituting Acme::PerlML for Acme::Bleach) becomes:

```
use Acme::PerlML;
```

```
<document><token_whitespace></token_whitespace>
<statement><token_word>open</token_word><token_whitespace></token_whitespace>
<token_word>my</token_word><token_whitespace></token_whitespace>
<token_symbol>$EXAMPLE</token_symbol>
<token_operator>,</token_operator><token_whitespace></token_whitespace>
<token_quote_single>&apos;&lt;&apos;</token_quote_single>
<token_operator>,</token_operator><token_whitespace></token_whitespace>
<token_quote_single>&apos;example.txt&apos;</token_quote_single>
<token_whitespace></token_whitespace>
<token_operator>or</token_operator><token_whitespace></token_whitespace>
<token_word>die</token_word><token_whitespace></token_whitespace>
<token_quote_double>&quot;Can&apos;t open example:$!\n&quot;</token_quote_double>
<token_structure>;</token_structure>
</statement><token_whitespace></token_whitespace>
<statement_compound><token_word>while</token_word><token_whitespace></token_whitespace>
<structure_condition><token_structure>(</token_structure>
<statement_expression><token_quotelike_readline>&lt;$EXAMPLE&gt;</token_quotelike_readline>
</statement_expression><token_structure>)</ token_structure>
</structure_condition>
<structure_block><token_structure>{</token_structure>< token_whitespace>
. . .
```

(plus more lines of a rather ugly XML representation of the Perl code).

You could imagine someone finding this transformation to be actually helpful, perhaps in conjunction with an XML database or XML acceleration appliance.

Before we leave the Acme::Bleach family to look at more useful modules, I think it is worth pointing out that the general concept of messing with the source code of a script right before it is executed is an interesting one that opens up many possibilities.

Perl has had a feature to do this for some time (although it isn't used by the Acme::Bleach module) called "source filtering." With source filtering you can write code that processes the source code being read into the Perl interpreter *before it is executed*. If you can fiddle with source like this before handing it to Perl to interpret it gives you the power to write your source in any form you'd like (just as long as it eventually can be transformed back to basic Perl syntax). Just to show the power of the concept, Damian Conway has written a module that allows you to program in Perl using Latin. See the perlfilter man page if you are interested in this concept.

Not Just Fun and Games

The set of Acme:: modules I tend to respect the most are those that play with various language concepts or attempt to solve real problems while still maintaining a sense of humor. An example of the latter is Acme::RemoteINC, which describes itself as the "Slowest Possible Module Loading." That's being overly modest. What it really does is fetch a module using FTP from some repository in a transparent way if it isn't available when the program runs. Even if this turns out to be a slow operation, you have to admit the concept is cool and ripe for further exploration.

Similarly strange but very clever in its own way is Acme::Scripticide, which allows you to write scripts that delete themselves. Why is this useful? The author explains this in the documentation:

> Believe it or not this is handy if you have a one time job to execute:
>
> ```
> # $script uses Acme::Scripticide
> system $script if -e $script;
> ```
>
> or say to create static files from a database:
>
> ```
> # in flowers.pl (copy this to whatever names you want and execute:)
> use Acme::Scripticide qw(good_bye_cruel_world);
> good_bye_cruel_world('.html', get_html($0));
> ```
>
> now flowers.pl does not exist and flowers.html is there.
>
> You could have a directory full of those types of scripts and glob() them in and execute each one; once that is done, you have a directory of corresponding static html files.

I get excited about stuff like this because it opens up a whole new avenue of thinking for solving certain kinds of problems.

Mucking about with language constructs using Acme:: modules has a similar expanding action on one's brain. For example, the Acme::BottomsUp module lets you order your really long compound Perl statements closer to the way you might explain the statement to someone. Once again I'm going to quote from a module's documentation, because it has a superb example. It shows that a code fragment like this:

```
my @arr = (1..10);

   print                  # lastly, display result
       join ":",          # and glue together
       map { $_**3 }      # then cube each one
       grep { $_ % 2 }    # then get the odd ones
       @arr               # first, start with numbers
;
```

"reads better" if you use the Acme::BottomsUp module like so:

```
my @arr = (1..10);

use Acme::BottomsUp;
@arr                      # first, start w/ numbers
    grep { $_ % 2 }       # then get the odd ones
    map { $_**3 }         # then cube each one
    join ":",             # and glue together
    print                 # lastly, display result
;
no Acme::BottomsUp;
```

If you've worked with other programming languages that use a different statement order, you won't find this idea to be particularly revolutionary. But for someone who has only written code like the "before" sample here, this module might provide a welcome ponderable about language design.

Let me give you one last titillating example in the language vein before we move on: Acme::use::strict::with::pride. This module is designed (and I quote) to "enforce bondage and discipline on very naughty modules." As soon as you load this module, all subsequent modules loaded by the script via "use" or "require" will find themselves running with use strict and use warnings turned on (and I quote again) "whether they like it or not."

A::u::s::w::p provides us with two things:

1. The chance to enforce the same level of discipline on the modules you import from someone else that you might impose on yourself when writing code.
2. Another good ponderable about what other sorts of context or manipulation could be applied to these external modules as they are loaded.

Getting More Entertaining

For a column with "impractical" in the name we're drifting perilously close to abject seriousness. Let's get a little lighter by looking at two modules that solve a "problem" that we may not have considered easily solvable.

The first module is actually not necessary as of this writing but it is nice to know it exists. Acme::DNS::Correct was written to correct for a condition that afflicted the Internet for a brief while back in 2003. This was the great VeriSign SiteFinder debacle. At some point VeriSign decided it would help the Internet by making sure that all domain names in the .COM and .NET top-level domains would resolve to something when queried, even the ones that *didn't exist*. This broke all sorts of things and so modules such as Acme::DNS::Correct were developed.

Acme::DNS::Correct lets you do all of the standard Net::DNS resolution stuff but is smart enough to remove all references to the $ROOT_OF_EVIL, VeriSign's SiteFinder server, when it encounters them. Luckily that "service" was quickly run out of town. Earthlink pulled a similar stunt earlier this year (http://kb.earthlink.net/case.asp?article=187117), so it is good to know that modules like this are still available should this idea rear its ugly head in any substantial way again.

A second Acme:: problem-solver module is the Acme::MetaSyntactic breed of modules. Acme::MetaSyntactic is dedicated to the problem of finding good example variable names when "$foo" and "@bar" ceases to cut it. I tend to use "$fred", "$barney", and "@betty" when teaching but thanks to this module it is clear that I've been limiting myself. The module has many, many themes (there are 104 as of this writing) from many different sources. It ships with a helper script called meta that allows you to say:

```
$ meta teletubbies 3 # give 3 example variable names using this theme
Noo_Noo
Laa_Laa
Tinky_Winky

$ meta sins 3
laziness
gluttony
pride
```

```
$ meta thunderbirds 3
Brains
Gordon_Tracy
Parker
```

You'll never be without interesting example variable names again.

OK, I Lied; It Is All Fun and Games

As a way of ending this month's column with a smile, let me stick to my guns and show you three modules that are legitimately of dubious practical value but are amusing nonetheless.

The first is Acme::Test::Weather. Acme::Test::Weather is meant to be like the other testing-oriented modules (Test::More, Test::Simple, etc.) I first wrote about almost precisely a year ago in this column. The difference is that instead of providing testing primitives that perform comparisons such as "Is the result of subroutine() eq to this string?" it provides tests such as:

```
is_cloudy()
isnt_snowing()
eq_fahrenheit()
lt_humidity()
```

The module allows you to write tests based on the current weather for the machine running this test. Seriously. To make this happen it first attempts to look up the IP address's location using CAIDA::NetGeo::Client. With this location it calls Weather::Underground to find the current weather for that location. Why does it do this? The doc says, "Because, you know, it may be important to your Perl module that it's raining outside."

(As a related aside, I have to confess that in one of my classes I show people Perl code that behaves a certain way based on the current phase of the moon. Maybe I need to package this into its own Acme:: module?)

The second of our closing modules will mostly amuse the computer science readers. Let me let it speak for itself:

```
Acme::HaltingProblem - A program to decide whether a given program
halts
```

I would show you some sample code that uses this module, but the documentation lists the following bug:

```
This code does not correctly deal with the case where the machine does
not halt.
```

And finally, there is Acme::Morse::Audible. Like Acme::Bleach, the first time you run it it rewrites the script containing your original source code. In this case the source code becomes a real MIDI file containing the original source code: the original source code translated into Morse code, that is. Once you strip out the leading "use Acme::Morse::Audible;" line anything that can play back MIDI files will let you listen to your Perl code as it would be rendered in dots and dashes. And yes, if you leave the first line intact the obfuscated script still runs fine.

At best the idea of translating your programs into audible representations may inspire some new great ideas (or some nostalgia for the days when listening to relays could help debug programs). At worst this module's very existence tickles me pink. On April Fool's Day that's good enough for me. Take care, and I'll see you next time.

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.

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

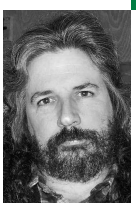
You own the copyright to your work and grant USENIX permission to publish it in ;login: and on the Web. USENIX owns the copyright on the collection that is each issue of *;login:*.
You have control over who may reprint your text; financial negotiations are a private matter between you and any reprinter.

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.

ROBERT G. FERRELL

/dev/random



Robert G. Ferrell is a chronically underemployed information security geek who enjoys surfing (the Internet), sashimi (it makes great bait), and long walks (to the coffee machine and back).

rgferrell@gmail.com

OVER TOO MUCH FINE BELGIAN ALE one night deep in my largely fictitious past, I decided to create my own file system. It wouldn't be anything fancy, I thought, just create a good, basic, reliable workhorse for the OS I would never get around to writing, either. I wanted to call it (the operating system) OreOS because I had the munchies at the time. Copyright issues seemed inevitable, however, so, to avoid legal entanglements and strike a blow for truth in advertising, I next named it ZenOS. An operating system that doesn't really operate anything is, after all, something of a paradox. The logical name for this new file system would therefore have been ZFS, but, sadly, Sun had nabbed that one out from under me. I finally settled on RGFS, on the supposition that no one else would want to name a file system using my initials.

Shortly thereafter I discovered that the German company Actum had a product called *Zenos*, although any durn fool can see this isn't the same thing as ZenOS. Their site was also in German, unfortunately, and since all the Deutsche I know I learned from *Hogan's Heroes*, I wasn't able to make heads or tails of what Zenos was supposed to do. In the interest of clarity and because I could already feel the hot, weaselly breath of intellectual property lawyers wilting my 70% post-consumer fiber shirt collar, I decided that discretion was the better, or at least less litigious, part of valor.

Most vendors solve this branding conundrum by just tacking "nix" onto the first syllable of some company-related word in the established evolutionary tradition of organisms mined from the Mother Code. My OS wasn't likely to resemble UNIX in any way, however, owing to the fact that I really don't understand how operating systems work, so I decided to eschew this nomenclatural methodology for something more novel and fitting. Besides, who wants to run something called "Robnix" (which sounds like a bad Web comic or a direct-to-DVD family action movie, an oxymoron if ever I've seen one)?

"What," I therefore inquired of myself, "would one reasonably call an operating system like mine—one still in the, um, *formative* stages?" The answer came out of the blue, as epiphanies are

wont to, like unto the arrival of a sinus headache shortly after stumbling into a field of blooming goldenrod: EmbryOS! As a name this had it all: It was descriptive, topical, vaguely biological, and relatively easy to print diagonally across a box in large, colorful letters. I had obviously missed my Madison Avenue calling long ago, at an early stage (probably between pupa and larva). Happily I sat back and envisioned the Web 3.0 HD 1080i 64-bit multimedia extravaganza (I figured 2.0 would be passé by the time I get around to an actual release), the brochures, the full-page ads in slick magazines, the IPO, and my eventual six-bedroom cabaña in the Bahamas, complete with fire-engine-red Porsche Carrera GT in the garage. Must . . . not . . . hyperventilate.

I put at least two full beers' worth of thought into the design of RGFS, and I came to the conclusion that journaling is just *so* late '90s. It was time to move file system architecture into the twenty-first century, I decided, so RGFS should keep track of changes not in a journal, but in a blog. That way, the system itself can add comments and attach still images, Flash presentations (what do you call Flash contained in flash memory? HyperFlash?) and even YouTube movies or streaming multimedia to the usual boring ol' file information. I might need to up the block size to, say, a terabyte or so to accommodate this new architecture, and that could perhaps slow down seek times a wee bit, but progress requires sacrifice, right?

File Blogging, or Flogging, might well be the file system wave of the future. It will render slack space obsolete, since every last bit will now be at a premium. Entire drives will need to be dedicated, not to the actual primary data itself, but to the metadata attached to it. We might even do away with the primary data altogether. RAID protocols will have to be adjusted accordingly, as well. Heck, let's just drop the RAID nonsense and ensure data redundancy using BitTorrent. Every storage device in the network neighborhood can host a stripe or mirror partition of every other one. SAN will now stand for *Symbiotic Area Network*.

Once computers are free—nay, required—to keep their own flogs, who knows what interesting and useful intel we can gain from plumbing their innermost ruminations? Will the dual-core processors consider themselves superior to their monolithic brethren? Will those employing LDAP have DNS envy? Will they list their favorite bands and complain endlessly about how we don't understand them?

While I'm on a roll, innovation-wise, I may as well implement some other ideas I've been kicking around, such as spanning the superblock across several volumes and encrypting the magic number. (Speaking of which, I'm thinking of incorporating a separate hardware data path just for passing around file system labels. I'll call it the Magic Bus.) The whole inode thing is hopelessly old-fashioned, too. I'm updating mine to iN0d3z and giving them their own MySpace page. Any machine that wants to add content to a file system can download them there.

EmbryOS/RGFS will of course be Open Source. In fact, I'm going to take this one step further and declare it to be Pro-Am Invitational Source, which means that I invite any programmer, be he or she professional or hobbyist, actually to create the source *for* me in the first place. You'll receive full credit for your input, naturally, in the End User License Agreement just below "EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES."

I ask for help not because I'm lazy and incompetent, but merely to protect the consumer. If I have to do the coding myself it may end up a little wonky. Inline-embedding GWBasic in Perl is just so exacting . . .

Riddle me this: Why is coding an operating system like being the parent of a young child on Christmas Eve?

Some assembly required.

book reviews

ELIZABETH ZWICKY, CHAOS GOLUBITSKY, SAM STOVER, AND RIK FARROW

WHAT TO DO WHEN YOU HAVE NO BOOK REVIEW

ELIZABETH ZWICKY

Periodically, I find myself in need of a technical book on a subject I know almost nothing about (I wouldn't need a book if I knew what I was doing) when reviews are not available to me for one reason or another. There I am, standing in a bookstore, in front of a row of books of unknown quality. What do I do then?

Well, reviewing means I read a lot of books, and not all of them are books I'd pick out myself. I wish I could say they were all winners, but even the books that I am unenthused about in print are the winning tip of an iceberg. I often think with great sympathy about the editors who have to read the manuscripts out of which these are winnowed. The books I hate were selected out of thousands of even worse books.

This experience with the good, the bad, and the ugly has led me to some rules I can use to make an educated guess about which is the best of the available books on a subject. These rules work best with physical books, where you can actually read bits that you select, and of course they don't guarantee excellence. But I find they're pretty reliable.

1. Start with the cover. It is a good thing if it is a second edition or later; enough people bought the first edition to make it worth doing again. It is a bad thing if the number of authors (or editors, in the case of an anthology) is greater than the edition + 1 (so a first edition can have two authors, a second edition three, and so on). Books work best if there's a strong unifying force, and you can't achieve that kind of unity well with more than two people. However, extra authors added for later editions usually mean that one or more of the original authors dropped out. If a first edition has enough authors to make up a sports team, I'm usually ready to move on right there.
2. While you're still judging the book by its cover, look at who published it. How do you feel about that publisher? The fact is that there are good books out there by every publisher, even the ones that periodically make me wonder whether they do actually reject books, but if you're making guesses, it's fair to predict based on your previous experience. If you've never heard of a publisher before, that's not necessarily a bad thing; I'd prefer an unknown publisher to one I've tried and hated. Good new presses do show up.
3. Open the book up. Stay superficial for another moment, and flip through it. Look at the print; is it a reasonable size? If you care about fonts, now would be a moment to rise above your prejudices. Some lovely books come in terrible, terrible fonts. But a book that's not marked "large type" and is printed in big fonts and/or with big margins is a flashing warning sign saying "We wanted this book to look big and important and it isn't."
4. OK, let's actually think about the content now. Start by looking for a section in the preface that tells you who the audience for the book is. There should be one. It should not say the moral equivalent of "Everyone on earth, or at least everyone who ever touches a computer, should read this book." You do not always need to be in the target audience, although it's a very nice sign if you are. But there needs to be a target audience, or the book itself can't be very coherent.
5. Look at the illustrations. Once again, ignore any graphic design tendencies you might have. Some otherwise sane presses let authors do their own illustrations, which tends to leave them one step up from being drawn in crayon on paper napkins. Regardless of what they look like, and whether they are easily legible, think about whether they have interesting, comprehensible content. If most of the illustrations are graphs with no X and Y axis labels, or screen shots of menus, this is a very bad sign.
6. Temporarily unleash any copy-editor instincts you have, and look at 10–20 pages. If you can find grammatical errors, spelling errors, or typos on more than one-quarter of the pages you look at (and I'm not talking forgivable ending-sentences-with-a-preposition–type errors, I'm talking "Excuse me, but that's three sen-

tences held together with randomly placed commas"–type errors), you will go insane trying to read the book. Please note that all books go to press with some errors; the first one you find might be your own good luck. But if there's a regular pattern, the editors were slack.

7. Find a section, any section, that you know something about. Read that section, and see if it gets the bits you already understand right. This can occasionally be misleading if the section you know is one that is truly very tangential to the main content to the book but, in general, coverage is pretty even. If it's incoherent or wrong in deep fundamental ways about the stuff you know, it's probably not right about the stuff you don't.

These rules are of course no substitute for reviews and recommendations from people you know and trust, or even from published reviewers (which I've got to say come a poor second to personalized advice from sensible friends). But they'll usually find you the least horrible available option.

CODE QUALITY: THE OPEN SOURCE PERSPECTIVE

Diomidis Spinellis

Addison Wesley, 2006. 521 pages. ISBN 0-321-16607-8

I've enjoyed this book a lot, but can I blame it for the shortage of reviews? It's a very dense book, with something to think about in every sentence. If you carefully absorb everything it has to say and manage to implement it, you will be a programming wizard. If you try to skim it you will have a very bad headache. (At all costs avoid the "Advice to Take Home" sections, which at worst run to more than five pages of bullet points, more than the human brain can possibly take home.) Fortunately, the chapters are relatively independent, so you could just gnaw on whichever subject you're interested in at the moment and be assured of getting as much out of it as you put in.

You will, however, need to be putting energy into it. The author is willing to put all the dots down, but it's up to you to connect them. Sometimes I suspect that this is because he doesn't realize the connections aren't intuitively obvious to every reader; other times it seems intentional. Frankly, with this many pages, there isn't the space to spell everything out. It's amazing that he manages to do a good job on such a wide range of topics as it is.

This is, unfortunately, a good example of a book you should not try to judge by the usability of its illustrations, which are heavy on content and light on pixels. Some of them ought to come with a magnifying glass.

BACKUP & RECOVERY: INEXPENSIVE BACKUP SOLUTIONS FOR OPEN SYSTEMS

W. Curtis Preston

O'Reilly, 2007. 729 pages. ISBN 0-596-10246-1

REVIEWED BY CHAOS GOLUBITSKY

The problem with backups is that you can't afford to ignore them, but your knowledge about how to do them is probably out of date. Backups are a subproblem of data storage, and storage options (as your users will tell you) change rapidly. This book is designed to make sure you have the information you need to perform your corporate backups effectively and without risk of terrifying data loss. Curtis Preston's new edition is a rewrite and significant expansion of the 1999 *Unix Backup and Recovery* (but this version covers Windows as well). The book's secondary goal is to focus on free and inexpensive backup solutions.

Preston is extremely knowledgeable about the world of backups, and the book shines most when he is showing off his experience. The overview chapters are full of information on backup strategies and considerations, on features to think about when shopping for commercial backup products, and on backup hardware. Highlights of these sections range from neat tricks such as how to use the Towers of Hanoi problem to improve incremental backup scheduling, to great descriptions of how several types of tape and optical media work.

About a third of the book is dedicated to general and specific commentary on database backup and recovery, including blueprints for designing and debugging instances of database servers from Oracle to PostgreSQL. These chapters may contain more about database implementation than you wanted to know—a stated goal is to help sysadmins and DBAs communicate with each other around the subject of backups—but the information you need to get your server running again after a disaster is almost certainly in there.

A number of chapters and sections are contributed by other authors or specialists, including most of the chapters on Open Source backup systems. The best of these chapters, such as Leon Towns-von Stauber's fast and usable blueprint for performing bare-metal recovery of a Mac OS X machine, have "invaluable reference during some future catastrophe" written all over them. Unfortunately, the quality of the contributed chapters is not consistent. I found it particularly frustrating that the chapters on Amanda and Bacula focused on different design features, preventing a head-to-

head comparison of the two products. Bacula's job scheduling and volume management capabilities got only a paragraph mention, whereas Amanda's had multiple pages. Having used Bacula, I know that it can do some job scheduling, and that its volume expiration facilities are sometimes confusing. Those features would have benefited from fuller treatment, and more careful editing would have improved the book as a whole.

A related word of warning is in order: This is not your grandmother's backup book. I have friends and relatives who don't do home backups (or whose backup schedule is "once every three years whether it needs it or not"), and I hoped to augment my advice for them with some of the free or cheap backup tools discussed in this book. Some programs that could be used on home networks, including rsync and BackupPC, are discussed, but most configuration examples pertain to managed networks. I would need to do other research to decide whether these tools would be feasible for non-savvy home users.

Unless your only data protection requirements fit neatly into one of the scenarios for which blueprints are provided, *Backup & Recovery* is not the last backup book you'll ever have to read. However, by reading this book, you will learn something you didn't know about the environment you are backing up and the tools you already use and gain perspective on what to think about when designing or improving your backup process.

BOTNETS: THE KILLER WEB APP

Craig Schiller, Jim Binkley, Gadi Evron, Carsten Willems, Tony Bradley, David Harley, and Michael Cross

Syngress, 2007. 464 pages. ISBN: 978-1-59749-135-8

REVIEWED BY SAM STOVER

I wasn't sure how this book was going to play out. I thought it might end up not being technical enough, but with the two names I did know from the author list (Evron and Willems) I had high hopes. I was not disappointed: I think this book is an *excellent* read for just about anyone interested in bots and botnets. I differentiate between bots and botnets because there is ample focus on both aspects of this new threat.

The book starts out with a fairly comprehensive overview of bot evolution, which I found to be the right mix of theory, history, and technical detail. This sets the stage for Chapter Two, which lays out the life cycle of a botnet. Since individual bots comprise a botnet, it's interesting to see how the technologies between the two differ. Creating a bot requires mobility and exploit code, while controlling a botnet requires a Command and Control (C&C) mechanism that's intuitive and easy to use (those poor, overworked bot-herders).

The next two chapters deal with C&C mechanisms and how different bots function in different botnets before we get to Chapter 5, which deals with detecting botnets. The first four chapters deal with understanding bots and botnets; the remaining eight deal with detection and mitigation. One of the things that I really like about this book is that the authors spend a lot of attention on two very different response/detection mechanisms. The first centers around the ourmon tool, which monitors network traffic and produces graphs to give you insight into how bots and botnets are impacting your network. There's just enough installation instruction to get you started; I'm running ourmon now, and it's pretty painless if you are running FreeBSD or Ubuntu—they have tips for installing on both. Think of ourmon as MRTG with a focus on detecting characteristics inherent to bots and botnets. Even with my little honeynet, I've found this tool to be very interesting and can easily see how beneficial it would be in an enterprise environment.

The second mechanism is called sandboxing, which focuses on interacting with the bot directly and performing analysis on the actions and capabilities. This is the main reason I was interested in this book. I've used Nepenthes (which is mentioned briefly several times), but I wanted to learn more about how sandboxing works—specifically, CWSandbox. Since Willems wrote CWSandbox, I figured he'd be the person to write the chapter on sandboxing, and I was right. Chapter 10 deals exclusively with sandboxing, with a heavy emphasis on CWSandbox. This is probably the most technical chapter in the book, and it's quite a read.

The remaining two chapters deal with resources available to learn more and communicate with other concerned citizens. There is a good sampling of various types of organizations and tools to help the bot-stricken administrator.

There were a couple of spelling and grammatical errors, as seems to be the trend in Syngress books, but nothing to get spun up over. A fair bit of overlap between chapters makes it pretty evident that multiple authors wrote the different chapters, but this was only mildly annoying. The chapters on ourmon and CWSandbox are worth the price of admission alone, and while I would have liked to see a bit more emphasis on Nepenthes, I can always hope that will follow in the next edition.

Overall, this was a great read. If you want to get a jump on the botnet learning curve, I'd recommend starting with this book.

BUILD REAL-WORLD, END-TO-END, NETWORK MONITORING SOLUTIONS WITH NAGIOS

David Josephsen

Prentice Hall, 2007. 230 pages. ISBN 978-0-13-223693-5

REVIEWED BY RIK FARROW

I came away from the 2006 LISA conference determined to learn more about monitoring, and Josephsen's book seemed like a good way to get my feet wet. I knew that Josephsen has a clear and easy writing style from his articles in *;login:*, and I had hoped that would help me make the plunge into learning about Nagios. I was right.

Nagios is a complex topic. Like any popular example of Open Source Software, there is a lot of information available online. What Josephsen adds to the online documentation is a thorough approach that starts by getting readers to consider their goals in monitoring, not just what should be monitored but how, as well as interrelationships that affect monitoring. Josephsen applies his personal experience in monitoring a large, distributed infrastructure to how he introduces the concepts, a touch that he carries throughout his book.

His introduction to the software itself is gentle, so that I am quickly convinced I can write my own Nagios plug-ins if need be. He explains that Nagios is really a scheduling and notification scaffold, and what it can do is largely up to you. After a short but mandatory chapter on installation, he gets into the nitty-gritty of configuration. This is, I expect, the place where most people who try using Nagios soon give up, as there are many files that must be edited. Although Josephsen does focus on the editing of files, he also covers the GUIs that can be used for configuring the Nagios Web interface. Along the way, he provides practical hints for making the management of the Nagios configuration itself easier.

Josephsen concludes with chapters on visualization and the new Nagios Event Broker interface. We humans are very good at extracting data quickly from visual information, and Josephsen spends a lot of his page budget explaining how to add useful graphs to Nagios, something that you don't get from just installing Nagio alone. Again, using clear explanations and an appropriate dose of humor, he explains how to start using RRDTool to produce graphs that best utilize your mental capabilities and impress managers everywhere.

The final chapter describes the Event Broker, a method that allows you to extend Nagios by adding in your own callbacks. As Josephsen explained this, I quickly understood that the Event Broker allows programmers to add to existing event handling by exposing internal global variables and data structures. Although you can't create logical service groups, for example, a collection of the hosts and services that provide a Web farm with a database backend, you can customize how Nagios deals with its events.

Three appendices round out the book: buildtime options, the many configuration options in two keys files (nagios.cfg and cgi.cfg), and command-line and plug-in arguments.

I did have some problems with the typesetting of this book. I find it amazing that Prentice Hall still can't typeset single back-quotes after all these years (having screwed this up in my first book 18 years ago), so what looks like a single-quoted string (and makes no sense) is really a command executed within a subshell, for example. That aside, I can recommend this book to you if you want to get started with Nagios or learn more about how to best set up a monitoring system.

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, VoIP, 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/.

ACCESS TO PAPERS from USENIX conferences online: www.usenix.org/publications/library/proceedings/

THE RIGHT TO VOTE on matters affecting the Association, its bylaws, and election of its directors and officers.

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. For details, see www.usenix.org/membership/specialdisc.html.

TO JOIN SAGE, see www.usenix.org/membership/classes.html#sage.

FOR MORE INFORMATION regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

USENIX BOARD OF DIRECTORS

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

PRESIDENT
Michael B. Jones,
mike@usenix.org

VICE PRESIDENT
Clem Cole,
clem@usenix.org

SECRETARY
Alva Couch,
alva@usenix.org

TREASURER
Theodore Ts'o,
ted@usenix.org

DIRECTORS
Matt Blaze,
matt@usenix.org

Rémy Evard,
remy@usenix.org

Niels Provos,
niels@usenix.org

Margo Seltzer,
margo@usenix.org

EXECUTIVE DIRECTOR
Ellie Young,
ellie@usenix.org

NOTICE OF ANNUAL MEETING

The USENIX Association's Annual Meeting with the membership and the Board of Directors will be held during the 2007 USENIX Annual Technical Conference, June 17–22, 2007, Santa Clara, California. The time and place of the meeting will be announced on-site and on the conference Web site, www.usenix.org/usenix07.

IN MEMORIAM: JOHN W. BACKUS, 1924–2007

Alex Aiken
Professor of Computer Science, Stanford University

Computing has been such an integral part of everyday life for so long that many people alive today don't remember a time when it wasn't so. In the 1950s, digital computers were only a few years old, and very little of what we take for granted about modern computers had been invented. One thing was already apparent, however: the new, powerful hardware needed equally powerful software if it was to be of use to anyone, and software was turning out to be surprisingly difficult, time-consuming, and expensive to write.

A big part of the problem was that software at the time was written directly in the machine's native assembly language. As anyone who has ever written in assembly knows, it takes a lot of error-prone coding to get anything done that way.

John Backus, who was working at IBM at the time, had the idea that programming could be greatly improved if the programmer could write more abstract and less machine-specific commands and if the program could be organized in a way that seemed more natural for humans. A special program called a compiler would take care of translating

from the higher-level language into the machine's language.

The obvious problem with this idea, a difficulty that led some to predict it would fail, was the concern that the translation would end up being much less efficient than a carefully written assembly program; since machines were very expensive, computer time was a valuable commodity.

But John was literally a visionary; he had a vision of what could be. He led a team that spent three years (1954–1957) designing and implementing the programming language and its compiler.

Within a few months of its first release, FORTRAN (for FORmula TRANslation) was a huge success for IBM, dramatically improving software productivity. FORTRAN was an even greater intellectual triumph, for it changed forever the way people thought about programming computers and led directly to a surge of interest in the design and implementation of high-level computer languages.

John was also a prototype of the modern computer science research manager, an oxymoron if there ever was one. John always claimed that his contribution to the FORTRAN project was only breaking up the chess games after lunch, that really he hadn't done anything himself.

His characteristic modesty aside, his management style was one of almost no management at all, taking only the role of articulating the vision of where the project should go, both to inspire those working with him and to convince his backers to continue funding the project.

The key was to let the project be guided by the technical constraints of the problem they wanted to solve, which was poorly understood at the time, and not by a manager's early guess at what the solution should be. It's a great trick, but really quite difficult to pull off, since projects of FORTRAN's ambition succeed only after many failures. It requires both great technical ability and a gift for connecting with people to keep an effort like that on track, but John made it look easy.

After FORTRAN, John went on to invent BNF (or Backus-Naur Form), the notation used universally today to describe the syntax of programming languages. He also was a major force behind the early development of functional languages, championing even higher-level programming as necessary to move beyond languages of the FORTRAN era.

For his work he won the Turing Award and the Draper Prize, among many other awards, but he never stopped being a researcher, always interested in new ideas and ways to make programming easier and more useful.

conference reports

THANKS TO OUR SUMMARIZERS

FAST '07: 5th USENIX Conference on File and Storage Technologies

San Jose, CA
February 13–16, 2007

INVITED TALK

■ ***A Crash Course on Some Recent Bug Finding Tricks***
Dawson Engler, Stanford University
Summarized by Brandon Philips (brandon@ifup.org)

Storage systems have a simple and important contract to keep: Given user data, they must save that data to disk without loss or corruption even in the face of system crashes. Dawson Engler and his students at Stanford have created eXplode, a systematic approach to finding bugs in storage systems, to help root out the bugs that can break this contract.

eXplode systematically explores all the possible choices that can be made at each choice point in the code to make low-probability events, or corner cases, just as probable as the main running path. And it does this exploration on a real running system with minimal modifications.

This system has the advantage of being conceptually simple and very effective. Bugs were found in every major Linux file system, including an fsync bug that can cause data corruption on ext2. This bug can be produced by doing the following: Create a new file, B, which recycles an indirect block from a recently truncated file, A, then call fsync on file B and crash the system before file A's truncate gets to disk. There is now inconsistent data on disk, and when e2fsck tries to fix the inconsistency it corrupts file B's data. A discussion of the bug has been started on the linux-fsdevel mailing list.

EXE (EXecution generated Executions) is another useful testing tool and was also briefly discussed. A developer using the tool would annotate the application to mark unrestrained input data. In the case of file systems, the unrestrained input data is a disk to mount. The annotated code is then run through exe-cc, which instruments the code, and then is compiled using gcc.

The added instrumentation will track all constraints on the input data to discover inputs that can cause termination by a call to exit(), crash, assertion failure, or error. The inputs that can lead the code to terminate are recorded and used to create an "input of death" or, in the case of a file system, a "disk of death" that can crash the system and uncover exploitable bugs.

Input sanitation on mounts is becoming more important as USB flash drives become pervasive and nonroot users get the ability to mount drives.

MEASURE THRICE

Summarized by Avishay Traeger (atraeger@cs.sunysb.edu)

■ ***Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?***

Bianca Schroeder and Garth A. Gibson, Carnegie Mellon University

Awarded Best Paper!

Bianca Schroeder began by stating that understanding disk failure frequencies has become increasingly important as server clusters become larger. Disk replacement data shows that what system administrators see in the real world is far different from what we statistically predict based on manufacturer specifications. Better knowledge about the statistical properties of storage failure processes, such as the distribution of time between failures, can empower researchers and designers to develop new, more reliable and available storage systems.

The authors draw their conclusions from seven supercomputing and ISP data sets, which include more than 100,000 drives. Some factors for why disks are replaced more frequently than predicted may be that administrators and disk manufacturers may disagree on the definition of a disk failure and that real-world operating conditions do not match those used by the manufacturers in their accelerated stress tests.

Some of the interesting observations are that the MTTF (Mean Time to Failure) is always much lower than the observed time to disk replacement, that SATA is not necessarily less reliable than FC and SCSI disks, and that, contrary to popular belief, hard drive replacement rates do not enter steady state after the first year of operation, but in fact steadily increase over time. In addition, early onset of wear-out has a stronger impact on replacement than does infant mortality. The authors also show that the common assumptions that times between failures follow an exponential distribution and that failures are independent are not correct. In addition to analyzing the data, they are creating a public failure data repository and hope that their results will help build better systems—for example, by predicting the probability of a RAID failure.

With regard to RAID failures, where a second drive failed during reconstruction, one questioner asked whether the data includes information about failures in specific arrays. The reply was that it does not, but the data that was used should provide a conservative estimate. To the question of whether the authors had a hypothesis about the correlation of disk replacements, their reply was that the disks are in the same physical environment and are probably under similar loads. Another question involved the accuracy of the data, given that the authors used disk replacements in their metrics, rather than actual failures. The authors said that even if they assumed that half of the replacements were unnecessary, the failure rate is still twice as much as the MTTF indicates. Someone asked how disk batches affected the data, and if it could account for all of the results, which would indicate that the data should be discarded. Since customers do not have data regarding bad disk batches, the authors said that this is difficult to estimate, but it is unlikely that it had much effect. Another person noted that since bad batches exist in the real world, the data should not be discarded.

■ ***Failure Trends in a Large Disk Drive Population***

Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso, Google Inc.

Although magnetic media are used to store most of the world's information, there is little published work on the failure patterns of disk drives and the key factors that affect their lifetime. Eduardo Pinheiro and the authors presented failure statistics from Google and analyzed the correlation between failures and several parameters generally believed to impact longevity. This analysis is made possible by a new highly parallel health data collection and analysis infrastructure, along with the ample amount of data provided by the authors' computing deployment. The hope is that this analysis will help predict failures so that administrators can act preemptively, help diagnose problems, and improve fault-tolerant techniques.

The authors found that there was no consistent correlation between higher temperature drives (for the available data, between 20 and 50 degrees Celsius) or drives at higher utilization levels (normalized per drive model) with failure rates. This indicated that other effects may be more prominent in affecting disk drive reliability. They confirm that some of the SMART parameters, such as scan errors, are well-correlated with higher failure probabilities. However, failure prediction models based on SMART parameters alone are likely to be limited in their accuracy, since many drives failed with no SMART errors.

Randal Burns (Johns Hopkins University) asked how this research has affected disk replacement policies at Google. The reply was they have not achieved high enough accuracy yet to base purchasing decisions on the data. Someone asked about the kind of accuracy one could get by modeling the SMART data. It was explained that this would result in many false positives and not enough good predictions, but it depends on the modeling techniques. In response to whether it was worth replacing disks as soon as an error was seen, the reply was that it would not be cost-effective. Someone later noted that this is true for Google since they have very good replication, but it is not true for the common user. Rik Farrow (*;login:* editor) asked whether multiple SMART errors are a better indicator for disk

failure, and the presenters admitted that this is in fact the case. Someone inquired as to whether the data that was collected included disk access patterns. Although this was examined, no significant differences were found. Ying Wang asked whether the type of disk failure was taken into consideration (i.e., bad blocks, head crashes, etc.). Eduardo replied that they did not include such data in their analysis, and they used disk replacements to model failures. Noting the correlation between SMART errors and failures, another questioner asked why this can't be used for prediction, but it was explained that although the drives will be more likely to fail after error, the rates are not so much more likely that a prediction is possible. More SMART data may help form predictions. Finally, the question of whether the correlation between disk failures and variation in temperatures was investigated came up. Although it was, there was no strong correlation.

■ ***A Five-Year Study of File-System Metadata***

Nitin Agrawal, University of Wisconsin, Madison; William J. Bolosky, John R. Douceur, and Jacob R. Lorch, Microsoft Research

Nitin Agrawal explained that metadata from over 60,000 Windows PC file systems at Microsoft collected over five years was used to study how various file characteristics changed over time. This data can prove useful for designers of file systems and related software, testing hypotheses, driving simulations, and validating benchmarks. The authors also provided a generative, probabilistic model for how directory trees are created, in terms of the depth of the namespace tree and the distribution of subdirectories. The authors plan to make their data set publicly available.

Some of their observations: (1) Both the mean file size and the number of files have increased. (2) A few filename extensions account for a large percentage of files and storage and have not changed much with time. (3) Less filesystem content is created or modified locally over time. (4) Directory size distribution has not changed significantly, although directory size is steadily increasing. (5) The fraction of documents in the namespace subtree meant for user documents and settings has increased in every year of the study. (6) Although filesystem capacity has increased dramatically, filesystem fullness has only slightly decreased. (7) Large files (e.g., binaries, movies, database files) are contributing to an increasing fraction of filesystem usage, but most files are still 4KB or smaller. (8) The median file age is between 80 and 160 days, and this has not changed with time.

Erez Zadok (Stony Brook University) asked whether there were trends in how people were organizing their files. The reply was that most files were being stored in the Windows, Documents and Settings, and Program Files directories. Another question was about the types of systems and workloads that were analyzed. Agrawal said that almost all were Windows PCs and were mostly from developers. Geoff Kuenning (Harvey Mudd College) asked how they considered machines where capacity was added, or the situation when a file system was migrated to a new machine. The study considered these to be new file systems. There was an inquiry about the presenter mentioning that most files were less than 4KB but also saying that the median size was 4KB. Agrawal joked that, in this case, most means half, and the audience laughed. In response to whether differences between developer and nondeveloper distributions were considered, the reply was that they considered temporal trends and not workload types, but the data is available for future analysis. Eduardo Pinheiro (Google) inquired about the reason for 4KB being the median file size, wondering whether this had something to do with block sizes or application behavior. The presenter was unsure of the cause. Finally, John Wilkes (HP Labs) asked about the amount of unique data present in the file system, since all probably had similar installs. While admitting that this data may be useful to analyze, they said they had not done so.

WHO PUT THEIR NETWORK IN MY STORAGE?

Summarized by Michael Mesnier (michael.mesnier@intel.com)

■ ***Proportional-Share Scheduling for Distributed Storage Systems***

Yin Wang, University of Michigan; Arif Merchant, HP Laboratories

Yin Wang began by saying that the advantage of distributed storage is the ability to scale out using cheap commodity hardware (storage bricks). HP's Federated Array of Bricks (FAB) is one example of such a distributed architecture. However, the drawback is often unavoidable resource contention, which is complicated by the fact that data is typically distributed across multiple storage bricks and accessed by clients using different I/O patterns.

The goal of this work is to provide proportional-share scheduling of storage resources based on a client's "weight," which can be assigned by an administrator. No previous work in this area has addressed multiple schedulers with multiple resources (storage bricks). Of course, if a scheduler is simply placed at every storage coordinator or back-end storage device, no one scheduler will ever see all of the I/O requests. For example, a scheduler at a storage brick will not see requests to other storage bricks, and a scheduler at a coordinator will not see requests to other coordinators. Thus it would be difficult to determine a client's total service across all resources.

One naive solution is to broadcast all I/O requests to all bricks or coordinators, but this of course introduces significant network overhead. However, the authors show that only the service cost of each I/O needs to be broadcasted to each coordinator. Such a broadcast includes a "delay value" that allows a storage coordinator to delay I/O requests from a particular client in order to provide propor-

tional-share service across clients. In their architecture, coordinators maintain FCFS queues and can therefore easily incorporate the broadcasted delay values.

The authors present a new proportional-service scheduling framework suitable for use in a distributed storage system that uses such a delay-broadcast approach. Their approach is an extension of SFQ(D), called Distributed Start-time Fair Queuing (DSFQ). Two approaches are evaluated: Total-DSFQ and Hybrid-DSFQ. The primary difference is that Hybrid-DSFQ guarantees a minimum amount of service to each client. In their evaluation, the authors showed that Total-DSFQ works in the single-brick case, across multiple bricks, and across multiple bricks with dependencies among the I/O requests. They also showed that Hybrid-DSFQ, unlike Total-DSFQ, guarantees a minimum share for each client stream for fluctuating workloads.

David Black (EMC) asked whether there was any negative result with respect to disk striping; he believed that total service was at odds with striping. Arif Merchant said that the degree of proportional sharing can be adjusted to work with striping. Wenguang Wang (Apple) then asked what would happen if the client issued one stream with small random I/O and another with sequential. Yin said that that you can still maintain total proportional-service sharing.

■ ***Argon: Performance Insulation for Shared Storage Servers***

Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger, Carnegie Mellon University

Matthew Wachs stated that benefits of shared storage include the simplicity of a single infrastructure and the ability to better load-balance workloads across storage servers. However, these benefits come at the cost of interference among different workloads. When workloads share storage resources without any regard to efficient I/O scheduling, storage efficiency can plummet. Whereas processor and network sharing is well understood, it is unclear how to best share storage resources such as disk and cache. Moreover, it is unclear what a "time slice" even means for a resource such as a cache.

The goal of Argon is to provide better performance insulation. In the ideal case, sharing a storage server among *n* processes will result in each process receiving 1/*n* of its stand-alone performance. Because this ideal may not always be achievable, Argon uses an "R-value" to force a bound on lost efficiency. Once an R-value, such as 0.9, is set, each of the processes will then have performance within that fraction of the ideal. In general, typical disk scheduling policies do not grant sequential workloads long enough blocks of time, resulting in too much interference from seeks. Therefore, one goal of Argon is to amortize the cost of each seek by having larger request sizes for sequential workloads (using prefetching or write coalescing). For a given R-value, Argon automatically determines at system startup the optimal request size for a given storage server.

Seek-cost amortization solves only part of the performance insulation problem. The other part is cache pollution. To address this, Argon statically partitions cache space among *n* competing workloads, such that each workload is given an amount of reserved cache space. The amount each workload is given is determined through simulation: Block-level traces are replayed through a cache simulator in order to determine how a workload's performance improves with increased cache allocation. Again, the goal is to achieve within an R-value of the ideal performance by balancing the cache allocations for each workload.

Wenguang Wang (Apple) asked about the cache replacement policy, believing something smarter than LRU could be used. Matthew agreed, noting that although LRU was used for the experiments, one could use different policies for different workloads. Matthew further noted that for certain combinations of workloads, more intelligent policies (e.g., ARC) may obviate the need to statically partition the cache. Another attendee asked whether Argon yields the remainder of a time slice to scheduled workloads. Matthew replied by saying that Argon does not currently implement yielding. The challenge is determining the "right" time to yield, as application think time may make it appear that a request stream is idle, when in fact more I/O is soon to be issued. Yin Wang (University of Michigan) asked whether a trade-off exists between the length of the time slice (request size) and the fairness. Matthew said no, as fairness can also be achieved with larger time slices. More strictly, fairness can be achieved with any length time slice so long as the same length is given to each workload. The purpose of longer time slices is to increase efficiency, not change fairness. An attendee from IBM suggested that better write scheduling (e.g., request coalescing) could improve performance, and Matthew agreed. Another attendee asked what would happen if the cache were not large enough to store prefetched data for a large number of sequential workloads. Matthew said that the I/O accesses would degrade into random with a shared cache and that Argon does nothing to change this fact. Finally, an attendee from NetApp mentioned that a large variance in response time might be unacceptable for certain applications. Matthew agreed, but he also noted that a low mean response time is often more important than the variance for many applications. Matthew further noted that applications that need low variance in response times might need to have their own systems.

■ ***Strong Accountability for Network Storage***

Aydan R. Yumerefendi and Jeffrey S. Chase, Duke University

For network storage to be trustworthy, Adan Yumerefendi declaimed that strong accountability is required for both clients and servers. Servers must be able to track client accesses and guarantee nonrepudiation. Clients must be able to verify that their actions against a server have been committed and verify that a server is faithful.

The authors presented CATS, a rudimentary network storage service with strong accountability properties. CATS takes a "trust but verify" approach. The properties of their threat model include authenticity and undeniability, freshness and consistency, and completeness and inclusion. Confidentiality is also desired but is outside the scope of this work. The mechanisms used by CATS to ensure strong accountability include digital signatures, client action history, broadcasts of digital signatures of server memory state, challenges and proofs, and auditing.

CATS is an object storage service and a state storage toolkit. Each stored object is annotated with action records that reflect all state changes to the object. Versioning preserves multiple versions of each object, and version stamps allow clients and servers to agree on the ordering of updates. Such stamps also make request replay/reordering detectable. Servers must commit to their state and broadcast a digital signature of their internal memory state to the clients. Thus, a server can deny service but never subvert the system. Moreover, servers can make provable statements about their internal state. Freshness prevents a server from reverting writes, as each accepted write is represented in the digest of a server's internal memory state, so a committed request cannot be removed without detection by the clients.

The CATS storage service presented by the authors is just one example of a strongly accountable service that can be built by using the CATS state storage toolkit. The authors' evaluation shows that the cost of such a service can be high but that the approach is practical when strong accountability is required.

One attendee asked whether clients can exchange digests directly to perform verification. Aydan answered yes, noting that a common framework could be used to publish digests. An attendee from HP labs asked how the systems scales to large numbers of objects, specifically, whether one can keep files from disappearing. Aydan said that a bound on the probability of the file/object's existence can be given, but it depends on the age of the object.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Brandon Philips (brandon@ifup.org)

■ *Secure, Archival Storage with POTSHARDS*

Mark Storer introduced POTSHARDS, a system to securely store archival data. The system attempts to find a better solution to long-term encrypted storage. It uses secret sharing across a number of RAID archives and ensures that no one archive gets enough shards to reconstruct the secret. It also uses approximate pointers to ensure that an attacker has to have access to the data on all archives in order to reconstruct the data set.

■ *SeFS: Unleashing the Power of Full-Text Search on File Systems*

Stergios Anastasiadis proposed SeFS, a file system that is designed to facilitate full-text search. The need for this new file system was defended by the weaknesses in existing search technologies such as electronic journal databases and Web searches. Journal database entries are immutable once they are added to the database and Web search engines index data infrequently, but on most file systems files are modified frequently, making both approaches unacceptable for filesystem search.

■ *On the Scalability of Storage Sub-System Back-end Network*

Yan Li presented progress on research concerning how many disks a Fibre Channel Switched Bunch of Disks (FC SBOD) can efficiently support. The workload will be simulated using the Storage Performance Council SPC-1 benchmark. The initial findings show that a 2Gbps FC SBOD is saturated by 48 disks under RAID5 and 53 disks under RAID6 with a stripe size of 16KB. Future work includes testing 4Gbps versus dual 2Gbps FC and the scalability of FC SBOD with a cache system.

■ *Layout-aware Exhaustive Search*

Aravindan Raghuveer presented the motivation and plan for a layout-aware exhaustive search algorithm. The work is motivated by Jim Gray's keynote at FAST '05, where he projected that future hard disks will take a day to read 10TB of data sequentially and 5 months with random access and 8KB sectors. To get the fastest search possible an exhaustive search algorithm will need two properties: It must use physical layout information to compute a optimal sequential search, and it must have the ability to suspend and resume a search to provide service to real-time requests. Two pieces of metadata will need to be maintained to achieve these goals: location metadata on data object placement and state metadata on a suspended search.

■ *Scaling Security for Big, Parallel File Systems*

Andrew Leung proposed a new protocol, Maat, to scale I/O security to petabyte-scale parallel file systems used for high-performance computing applications. Current systems rely on shared-key cryptography, which can be a weakness in a large system. Maat uses extended capabilities, automatic revocation, and secure delegations. Extended capabilities can authorize I/O for a number of clients and files, which reduces the number of capabilities in the system. The small number of capabilities allows asymmetric cryptography to be used efficiently. Each capability has an automatic revocation timeout and can be extended by using a small, efficient extension token. An implementation is underway on top of the Ceph parallel file system.

■ *CompulsiveFS: Making NVRAM Suitable for Extremely Reliable Storage*

Kevin Green presented CompulsiveFS, a file system that stores persistent metadata directly to an erasure coded log in NVRAM instead of a RAM cache. The file system performs incremental erasure-encoding and signature computations over the contents of the log to create fault tolerance. The current prototype log is an order of magnitude faster than page-protected caches for small writes of 10–50 bytes. CompulsiveFS is in the early stages of research, design, and implementation.

■ *Performance Evaluation of RAID6 Systems*

Yan Li presented a plan for a three-piece study on the performance characteristics of RAID6 under the Storage Performance Council-1 benchmark. The study will simulate the storage using SimRAID, which models and simulates the RAID controller, cache, fibre-channel bus, and disks and has been shown to have a maximum inaccuracy of 5%. The study looks at RAID6 performance under fault-free mode, degraded mode, and recovery mode. Of particular interest is finding the ideal mix of handling requests versus rebuilding a disk.

■ *GANESHA, a Multi-Usage with Large Cache NFSv4 Server*

Philippe Deniel presented work on GANESHA, a user-space NFSv4 server with a large cache and support for a number of backend file systems. GANESHA has a filesystem abstraction layer (FSAL) that makes writing plug-ins for different backing file systems easy. Currently, there is support for HPSS and POSIX backends. A number of interesting backends are under development; these include an NFSv4 client that will allow GANESHA to be a proxy, an SNMP backend that will allow MIBS to be browsed, and the LDAP backend that will allow browsing of trees. Announcements to SourceForge and freshmeat are forthcoming.

■ *FlexiCache: A Flexible Interface for Customizing Linux File System Buffer Cache Replacement Policies*

Pavan Konanki presented the initial work for FlexiCache, an interface in the Linux kernel to accommodate different buffer cache replacement algorithms. The motivation for this work is to test new replacement algorithms, such as ARC, PCC and LIRS, that have been shown to perform better under certain access patterns. Also, this API would accelerate the testing and development of new buffer cache replacement algorithms. The key issue is trying to design the system to support many replacement policies while keeping the cache mechanics hidden. The performance implications of this added generality are also unknown.

■ *Diamonds Are Forever, Files Are Not*

Surendar Chandra talked about storage systems that use an "importance number" to decide how to manage a data store automatically. The experiments were motivated by a storage server for video-recorded lectures where some of the data may be less important than new data coming in and can be removed. To make the system successful, administrators must be able to specify accurate object lifetimes; therefore providing usage feedback is important. Currently a system is being built to prototype this idea.

■ *RBF: A New Storage Structure for Space-Efficient Queries for Multidimensional Metadata in OSS*

Yu Hua presented RBF, an r-tree with a bloom filter at each node, a structure that allows for efficient point and range queries. Point queries ask whether an object is in a data set and a range query grabs the set of objects that match a query. The application of this is to make efficient object-based storage devices that can do point/range operations on object metadata. Currently, a real 10TB storage system has been implemented using a partial implementation of the RBF structure.

■ *Storage Performance Isolation: An Investigation of Contemporary I/O Schedulers*

Sarala Arunagiri presented research on the performance isolation characteristics of a number of modern I/O schedulers. This research is motivated by the quality of service guarantees that large consolidated shared storage must make. The findings suggest that many I/O schedulers do not provide performance isolation in all circumstances.

INVITED TALK

■ *Trends in Managing Data at the Petabyte Scale*

Steve Kleiman, Senior VP and CTO, Network Appliance
Summarized by Michael Mesnier (michael.mesnier@intel.com)

In the first three-quarters of 2006, approximately 900 PB of storage was shipped worldwide by storage systems vendors, including Dell, EMC, Hitachi, HP, IBM, and Network Appliance. Of this total, Network Appliance shipped approximately 200 PB and is currently at a 100 PB/quarter run rate.

The big challenge today is keeping pace with such growth. A 50–100% yearly increase in storage capacity is not uncommon. Most of this growth is from unstructured data (e.g., contracts, letters, memos) and semi-structured data (e.g., email), but structured data (db) is also growing. Managing this growth introduces hidden burdens (costs).

A common strategy is to overprovision, resulting in low disk utilizations (with 25% or less being typical). Of course, CIOs do not want to take chances when it comes to safely storing company and/or customer data. Today's legal burdens (e.g., regulatory compliance) and social burdens (e.g., losing data is bad press) underscore this point. In general, overprovisioning stems from the "build-out" manner in which most storage infrastructures are managed today (i.e., within application-centric "silos"), including

space for an application's primary storage, disaster recovery, testing and development, backup, and archive. Although silos provide good QoS, they can result in different processes for managing data, and they require too many experts.

The key to reducing hidden costs is to separate data from its physical containers and to use virtual copies of data whenever possible (e.g., snapshots, clones, and data mirrors). This can help reduce the physical storage requirements and consolidate storage infrastructure. In turn, this will reduce the number of processes and experts. Of course, a new management paradigm is required to deal with virtual copies. The approach taken by Network Appliance is to consolidate all data and copies associated with a particular application into a "data set." Data sets can have properties such as security, regulatory compliance, QoS, and namespace management. Indeed, future systems will see unified environments with user-specified properties on data sets. As such, properties can remain constant while the storage infrastructure adapts to advances in storage technology.

In summary, much of this is already starting to happen. "It's good to be in storage!" Steve proclaimed. There is another decade of interesting change (megatrends) ahead of us.

Rik Farrow (USENIX) asked whether there will be a reduction in the amount of storage because of virtual copies, and a similar question was posed by Chris Lumb (Data Domain). Steve said yes, but he also pointed out that it will be easier to create copies. Rik then asked if file systems will become more interesting. Steve said yes, as they will be forced to use the new abstractions for storage. Garth Gibson (Carnegie Mellon) made a comment in support of the Aperi open-source storage management framework. David Black (EMC) asked how we can prevent users from turning the QoS dials "all the way to the right." Steve said that, in practice, users will be presented with options that they will have to pay for (e.g., bronze, silver, and gold). Julian Satran asked how we can move forward to content management, as users are not interested in managing storage. Steve replied that although he did not talk about application integration, he expects storage to get tighter with the application (e.g., data sets administered through Oracle). An attendee from Berkeley asked how support will work for a single management infrastructure. Steve said that virtualization will make it difficult, but he expects that most of the deployments will occur in a "two worlds" scenario where applications can share the same physical infrastructure but still be managed separately.

THE LATEST VERSION

Summarized by Raja Sambasivan (rajas@andrew.cmu.edu)

- ***Design and Implementation of Verifiable Audit Trails for a Versioning File System***

  Zachary N.J. Peterson, Randal Burns, Giuseppe Atensiese, and Stephen Bono, Johns Hopkins University

Zach Peterson presented a system capable of creating, managing, and verifying digital audit trails for versioning file systems. The digital audit system he presented involves three components: the file system, an authenticator, and an escrow site. The file system generates new versions of files and, when an audit trail is desired for a particular file version, commits an authenticator of that file version to an escrow site. The authenticator stored by the escrow site is a MAC that includes the authenticator of the previous version of the file and the data contained in the current version. An independent auditor can verify the contents of a given version of a file by first requesting both the version data and the previous version's authenticator and then comparing the auditor-constructed MAC based on this data to the MAC stored on the escrow site.

During his talk, Peterson noted that a central challenge the authors faced lay in finding a way to limit the amount of I/O necessary when computing authenticators. Computing an HMAC for a file, for example, requires that all of the file's data first be read into memory. To address this problem, the authors chose to use an XOR MAC as the authenticator. XOR MACs allow incremental authentication and thus allow the cost of authentication to scale with the size of changes to the data, not the size of the file.

The authors implemented their digital audit system in the ext3cow filesystem. Evaluation was performed using two micro-benchmarks and a trace-driven study, all of which showed significant gains for using an XOR MAC based on SHA-1 versus an HMAC based on SHA-1. Most of these gains were a result of the XOR MAC needing to perform less I/O than the HMAC. Most interestingly, Peterson pointed out that XOR MACs outperform HMACS especially well when the workload seen is dominated by small appends. These workloads can best take advantage of the incremental nature of the XOR MAC. Peterson then showed that write activity is dominated by appends via an analysis of his trace data.

During the Q&A period, Bill Bolosky from Microsoft asked whether the authors had considered using Merkle trees instead of the XOR MAC for their authenticators. Peterson responded by stating that they had considered Merkle trees, but had decided to use the XOR MAC instead because it is more efficient for data that changes via small incremental updates. David Black from EMC then asked whether the XOR MAC reveals information about what was authenticated and whether such transparency matters. Peterson's response was that it is the underlying MAC that

determines if any information is revealed. In this case the underlying MAC, SHA-1, does not reveal any information. Finally, Eduardo Pinheiro from Google wanted to know the size of the trace data used. Peterson stated that the trace data size was 4.2 gigabytes.

■ ***Architectures for Controller Based CDP***

Guy Laden, Paula Ta-Shma, Eitan Yaffe, Michael Factor, and Shachar Fienblit, IBM Haifa Research Laboratory

The ability to roll back to any previous storage state is clearly very useful and is exactly the functionality that continuous data protection (CDP) techniques provide. In this talk, Paula Ta-Shma compared four different storage architectures and provided analytic equations for reasoning about their cost when applied to different types of workloads. Finally, a trace-based study was used to validate the equations and compare the architectures on real workloads.

Paula Ta-Shma started by stating that the cost of using CDP is the number of user data device I/Os per write request in the common case (when no data is being reverted). This cost depends on four variables: the CDP granularity (specifically, the granularity of updates preserved), the workload (specifically, the temporal distance distribution of writes), the controller write cache (specifically, the size and replacement policy), and the CDP architecture in use.

Next, she described four different architectures for CDP: Logging, SlipStream, SplitDownStream, and Checkpointing. The Logging architecture is the simplest—the entire history of writes is stored in a log. Only one user data I/O per write request is incurred. However, although this architecture works well for write-dominated workloads, it does not allow for good read performance.

To remedy the read performance problem of the Logging architecture, the next two architectures split the version data into two volumes—a directly addressable "current store" that holds the current data and a hidden, mapped "history store" that holds all historical data, including the current version. The first of these architectures, SplitStream, splits write data above the cache; one copy of the write data is sent to each volume and thus a maximum of two user data I/Os and a minimum of zero user data I/Os are incurred per write request. Conversely, the SplitDownStream architecture splits write data underneath the cache. This architecture allows cache pages to be shared across current and historical volumes, thereby conserving memory resources. Compared to the Logging architecture, both the SlipStream and SlipDownStream architectures achieve better read performance, but they incur more I/Os and use more resources.

The Checkpointing architecture is similar to that of SplitDownStream, except that the history store only holds previous versions of write data. When a write is evicted from cache, the previous version of that data is copied from the current store to the history store. This architecture requires a maximum of three user data I/Os and a minimum of one.

Next, Paula compared the CDP cost of the various architectures when applied to a trace of the SPC-1 benchmark, which exhibits large temporal write distances, and the Cello99 trace workload, which exhibits smaller temporal write distances. For SPC-1, the results showed that SplitDownStream costs less than Checkpointing for smaller CDP granularities (i.e., less than 30 minutes). The same trend was observed for Cello99, except the cutoff point occurred near 5 minutes instead of 30 minutes. Additionally, the cost of Checkpointing declined to that of Logging at very coarse granularities.

At the end of the talk, an attendee asked whether the cost of SlipStream and SlipDownStream was one, not two, user data I/Os per write, since the I/Os to the current store and to the history store occur in parallel. Keith Smith asked whether different CDP architectures will need different metadata structures and thus differ in the cost of accessing metadata. Paula responded that in her CDP implementation, the underlying metadata structures for different architectures are the same, but they could be different.

■ ***Jumbo Store: Providing Efficient Incremental Upload and Versioning for a Utility Rendering Service***

Kave Eshgi, Mark Lillibridge, Lawrence Wilcock, Guillaume Belrose, and Rycharde Hawkes, HP Laboratories

Many providers would like to provide batch services to customers, in which clients send some data to the provider and the provider performs some large computation on the data (e.g., finite element analysis, data mining) and then sends the results back. However, providing batch services is difficult because the links used to transfer data from the customer to the provider (e.g., over the Internet via ADSL) tend to be very slow. In this talk, Kave Eshgi addresses this problem by describing a new storage system called Jumbo Store, which uses Hash-Based Directed Acyclic Graphs (HDAGs) to provide incremental upload of filesystem snapshots from a Jumbo Store client to a Jumbo Store server. The authors claim that incremental upload is a solution to the transfer problem since, in many cases, new client jobs use data that is only slightly different from previous jobs. Eshgi noted that Jumbo Store had been experimentally evaluated by incorporating it into the HP Labs prototype utility rendering service located in Palo Alto, CA, and used by animators in England to create a number of high-quality animated shorts.

To provide some intuition about HDAGs, Eshgi noted that HDAGs are a generalization of Merkle trees. However, he was adamant that HDAGs not be called Merkle trees, as DAGs, unlike trees, can contain nodes with multiple parents. Also, unlike Merkle trees, nonleaf nodes can contain data in an HDAG.

An HDAG is a directed acyclic graph (DAG) whose nodes refer to other nodes by their hash rather than their location in memory. Each node in a HDAG is comprised of two fields: the pointer field, which is a possibly empty array of hash pointers, and the data field, which is an application-defined byte array. Jumbo Store encodes the directory structure into an HDAG by representing each directory as an HDAG node whose data field contains that directory's metadata and whose hash pointers point to the directory's members. Conversely, a single file is represented as a two-level HDAG; the first-level node's data field contains the metadata for the file and its pointer field points to a node containing the file contents. To avoid having to send the entire file contents every time a small portion is changed, the file contents are broken into chunks via a technique called content-based chunking.

Eshgi next described the utility rendering service (URS) in which Jumbo Store was used. The URS is a batch utility service that performs the calculations required to render a 3D movie. Animators in the U.K. would use the URS client to submit rendering jobs to the URS in Palo Alto. To evaluate Jumbo Store, a subset of the data uploaded by the animators was re-uploaded using rsync (with the compress option turned on) and the difference in actual data transferred was noted. The authors found that Jumbo Store, on average, transferred half as many bytes as rsync.

During the Q&A period, an attendee asked whether the chunk sizes used to split up the file contents were static. Eshgi responded affirmatively, stating that a static chunk size of 4KB was used. Another attendee wanted to know whether Jumbo Store was subject to large latencies when uploading information. Eshgi answered that latency was large, but that latency was a relatively inconsequential metric given the usage scenario of Jumbo Store within the URS. Specifically, the animators would often leave or work on other things as soon as they were convinced that the incremental upload had started. Finally, Keith Smith asked whether the feedback from the animators using the system had been positive. Eshgi responded that the animators liked the system.

SCALABLE SYSTEMS

Summarized by Avishay Traeger (atraeger@cs.sunysb.edu)

■ *Data ONTAP GX: A Scalable Storage Cluster*

Michael Eisler, Peter Corbett, Michael Kazar, and Daniel S. Nydick, Network Appliance; J. Christopher Wagner, Ironport Systems, Inc.

Peter Corbett began by describing Data ONTAP GX as a scalable clustered network file server composed of a number of cooperating filers. The storage of a large number of filers can be presented as a single shared storage pool. The system's key features are scalability (by allowing for the easy addition of filers to the cluster), location transparency of data within the cluster, an extended namespace that can span multiple filers, increased resiliency in the face of failures, and simplified load and capacity balancing.

The system exports both NFS and CIFS protocols to clients via virtual interfaces (VIFs). Requests are initially processed by the networking blade (N-Blade), which terminates incoming NFS and CIFS connections and maintains protocol-specific state. The requests are translated into SpinNP RPCs, which are transmitted over a cluster fabric to the server responsible for the target volume, where a volume is a filesystem subtree and volumes are grouped into aggregates. SpinNP calls are processed by the data blade (D-Blade) on the target server. Two cluster-wide databases are used to route requests and responses. The volume location database (VLDB) tracks the corresponding aggregate for each volume, as well as the D-Blade currently responsible for the aggregate. The VIF manager database tracks which N-Blade is currently hosting each virtual interface, so that D-Blades can send callbacks to clients.

Data ONTAP GX is the first system to achieve one million operations/s on the SPEC SFS benchmark. In addition, it scales linearly on all benchmarks up to 24 nodes. The product is already deployed at customer sites and provides a powerful set of features that go well beyond what a stand-alone file server offers.

Gregory Touretsky (Intel) asked whether performance was measured on single or multiple D-Blades, and the reply was that benchmarks were run across multiple volumes and multiple D-Blades. Someone asked what happens to performance when they go beyond 24 nodes, and the audience laughed. The reply was that this is what is currently being shipped, and they are working on expanding the system further. There was a question about what was used to benchmark performance via the CIFS interface, and if any results were available. The reply was that there is no standard CIFS benchmark, and any results that they have are not publicly available. Another questioner wondered why write throughput was less than half of read throughput. The answer was that it is probably the result of read-ahead.

■ *//TRACE: Parallel Trace Replay with Approximate Causal Events*

Michael P. Mesnier, Intel Research with Carnegie Mellon University; Matthew Wachs, Raja R. Sambasivan, Julio Lopez, James Hendricks, Gregory R. Ganger, and David O'Hallaron, Carnegie Mellon University

Michael Mesnier stated that the goal of //TRACE is to extract traces of parallel applications and replay them in such a way that the I/O behavior is true to the original workload. This is done by discovering internode data dependencies and inter-I/O compute times. Replaying the trace as fast as possible is one classic option, but performance is poor because it assumes an I/O bottleneck, there is no idle

time, and dependencies are ignored. The other classic replaying option is to use the original timing, but with this technique we would not see any changes in performance when replaying on different hardware.

//TRACE is portable and treats the application and storage system as a black box. To find the dependencies, the workload is run multiple times, each time adding delays to the I/O stream of a node. I/O dependencies can be found when nodes block on the I/O that is currently being throttled. This will also find computation times. It then annotates the per-node traces with the dependencies. These annotations allow a parallel replayer to closely mimic the behavior of a traced application across a variety of storage systems. The number of runs and the rate at which I/O is throttled determine how many data dependencies can be discovered, as well as the time necessary to collect the traces. (One can trade off running time and replay accuracy.) Once //TRACE has collected this information, it can adjust with the speed of the storage system while enforcing dependencies. This technique works well for parallel applications with deterministic I/O dependencies.

The causality engine that //TRACE utilizes is implemented as an LD_PRELOAD library, allowing it to capture file-level traces. This allows users to properly evaluate different file and storage systems. Compared to other replay mechanisms, //TRACE offers significant gains in replay accuracy. Overall, the average replay error for the three parallel applications evaluated is below 10% when throttling every node and delaying every I/O. Trading off replay errors with greater time in collecting the traces was explored.

One question was asked about how //TRACE orders events within a node. Mesnier explained that it is actually the threads that are being throttled, and so this is not an issue. Another point raised was that most HPC workloads today are performing nondeterministic I/O, where this technique will not work. The reply was that there are many workloads of both types, and this work targets the deterministic applications. A follow-up point was that many HPC centers have clusters that are being shared by several applications, so even a deterministic application will act in a nondeterministic fashion. Mesnier responded that they are targeting dedicated clusters, which are often used for larger applications. Another question involved how one could deal with bottlenecks within the applications. The answer was that one could perform static analysis on the traces to see if there are too many barriers, for example, or possibly visualize the traces. In response to whether they had tried replaying the traces on systems other than those where the trace was collected, the reply was that this is what they had in fact done in their evaluation.

CACHE PRIZES

Summarized by Raja Sambasivan (rajas@andrew.cmu.edu)

■ *Karma: Know-It-All Replacement for a Multilevel Cache*

Gala Yadgar, Technion; Michael Factor, IBM Haifa Research Laboratories; Asaaf Schuster, Technion

Gala Yadgar introduced Karma, a system for optimizing cache replacement in systems with multiple levels of caching. Yadgar noted that multilevel cache hierarchies introduce three major problems in cache replacement. First, locality information is hidden from lower-level caches by the upper-level caches. Second, cache space is wasted because blocks are often duplicated at multiple cache levels. Finally, contextual information about blocks (e.g., the file to which they belong, the application that issued the I/O request, etc.) is also often hidden from the lower-level caches. Karma addresses all of these problems in concert. First, hints from applications are used in all cache levels to divide disk blocks into ranges based on their expected access pattern and access frequency; each range is allocated its own cache partition (which may span multiple cache levels) and replacement policy. The data blocks contained in each partition are managed by Karma using the READ, READ-SAVE, and DEMOTE operations. READ forces the lower-level cache to delete a block whenever it is read by an upper-level cache. Conversely, READ-SAVE allows a lower-level cache to keep a block read by an upper-level cache. Finally, DEMOTE sends a block evicted from an upper-level cache back to the next lower-level cache. Overall, by using application hints and partitioning the cache, Karma is able to use the optimal replacement policy for each access pattern seen. As a result, it compares favorably to all other cache replacement techniques (e.g., LRU, ARC, MultiQ).

There are two key design decisions in Karma. The first lies in how it partitions the workload it sees into ranges. The second revolves around the mechanism it uses to allocate a cache partition for each range. Workloads are partitioned into ranges based on application hints. Blocks in a given range have similar access frequencies and are accessed in a similar fashion. Application-level hints are propagated to lower-level caches by attaching a range identifier to each cache block. Karma allocates a cache partition for a given range in such a way as to maximize the "normalized marginal gain" for that range. The marginal gain for an access trace is the increase in hit rate that will be seen by this trace if the cache size increases by a single block. For example, for sequential accesses, the marginal gain is always zero; conversely, for looping accesses, the marginal gain is constant until the entire loop fits in cache and is zero afterward. Ranges with higher normalized gains are allocated cache partitions in higher cache levels, since it is likely that blocks in these ranges will be accessed more frequently.

During the Q&A session Jason Flinn from the University of Michigan asked how Karma dealt with access patterns that changed too quickly for Karma to optimize for them. Yadgar explained that such access patterns are not handled by Karma, but their effects are somewhat minimized because Karma does not yet perform any prefetching. John Wilkes from HP Labs then asked whether the authors had thought about using Karma in a multiple host setup. As an example, Wilkes noted that several changes had to be made to DEMOTE to adapt it to work with multiple hosts. Yadgar responded by stating that this was future work and that, for the multiple-host case, the authors would have to look into replacement policies that take into account the fact that blocks are not necessarily discarded when they leave a given host.

■ ***AMP: Adaptive Multi-Stream Prefetching in a Shared Cache***

Binny S. Gill and Luis Angel D. Bathen, IBM Almaden Research Center

In this very entertaining talk, Binny Gill provided an analysis of sequential prefetching algorithms for the case where an LRU cache houses prefetched data for multiple concurrent sequential streams. He first separated current sequential prefetching algorithms into four different classes based on two criteria: whether the prefetching algorithm chooses the prefetch size in a fixed manner or in an adaptive manner and whether the prefetching algorithm prefetches in a synchronous or asynchronous manner. He then showed that, of these classes, Adaptive Asynchronous (AA) prefetching algorithms show the most promise for sequential prefetching in terms of minimizing cache pollution and wasted prefetches. Next, a formal analysis showing how to best adaptively pick the parameters p (the prefetch distance) and g (the distance at which a prefetch is triggered) for AA prefetching algorithms was shown. Finally, Binny described AMP, an implementation of an AA algorithm, and experimentally showed its superiority to other types of prefetching algorithms for sequential workloads.

During the talk, Binny described his taxonomy of different prefetching algorithms by using an animation showing customers reaching for and eating donuts on a table while a waiter strove to "prefetch" more donuts according to the semantics of the various prefetching algorithm classes. His taxonomy classifies prefetching algorithms into four different classes: Fixed Synchronous (FS), Fixed Asynchronous (FA), Adaptive Synchronous (AS), and Adaptive Asynchronous (AA). The first term in this nomenclature refers to whether the prefetching algorithm adaptively chooses the prefetch size. The second term refers to whether prefetching is carried out synchronously or asynchronously. Binny noted that although AA prefetching algorithms show the most promise for sequential streams, they have rarely been implemented in practice.

Next, Binny quickly described a formal analysis showing how best to adaptively choose the prefetch size, p, and prefetch trigger distance, g, for each independent stream when using an AA prefetching algorithm. This information was then used to describe the Adaptive Multi-Prefetching (AMP) algorithm. This algorithm adapts the prefetch size by decreasing the value of p whenever a prefetched page reaches the end of the LRU list unaccessed. To minimize wasted prefetches, such unaccessed prefetched pages are also moved back to the head of the LRU list and marked as "old" when they fall off the end of the LRU list. Conversely, the value of p is incremented whenever the last page read within a given read I/O hits in cache and the cache block hit is not "old." The size by which p is incremented is the size of the read operation in pages. The prefetch trigger distance, g, is incremented whenever a prefetch returns only to find that a read request is already waiting for some page in the prefetch set. The value of g is decremented when p is decremented.

Finally, Binny showed that the AMP algorithm easily outperforms the other classes of prefetching algorithms on several different workloads. These workloads included a sequential stream workload, a workload with many short sequential sequences, the SPC-1 Read workload, and the SPC-2 Video on Demand workload.

During the Q&A session, John Wilkes from HP Labs asked whether the authors had tried to use AMP on mixed workloads. Binny responded that they hadn't explored this space yet, but that the SPC-1 Read workload, on which AMP performed well, is not a strictly sequential workload.

■ ***Nache: Design and Implementation of a Caching Proxy for NFSv4***

Ajay Gulati, Rice University; Manoj Naik and Renu Tewari, IBM Almaden Research Center

Sharing data across wide area networks (WANs) is becoming very common; however, a common problem with WAN file sharing is high latency. A common solution used to reduce latency in WANs involves installing a caching proxy close to the client, but many protocols available for use by the caching proxy and the server, such as CIFS and NFSv2/v3, are optimized for local area networks (LANs), not WANs. As a result, these protocols tend to be "chatty" and result in suboptimal performance. In this talk, Ajay Gulati described why NFSv4 is a good choice for the protocol that should be used between caching proxies and servers and then proceeded to describe the design and implementation of such a NFSv4 caching proxy (Nache).

Gulati stated that NFSv4 is useful in a caching proxy setup because it provides read/write delegations and compound requests. A write delegation for a file issued to a client by an NFSv4 server gives the client the authority to modify that file locally without interacting with the server until the delegation is revoked. Similarly, a read delegation for a file issued to a client allows the client to read the file without continually checking cache consistency. For cases

where sharing of files among clients is uncommon, delegations can greatly improve performance. Compound requests allow multiple related requests to be batched into a single request; such requests are suited for WANs as they generate less overall network traffic and per-command round-trip delays than the case in which each request contained in the compound request is issued separately.

Ajay then described Nache, a caching proxy for NFSv4 that sits between local clients and a remote server. Nache, in some sense, relies on receiving delegations from the server for the files accessed by its clients. If it does receive these delegations, it can proceed to serve the clients locally without communicating with the server.

Nache is implemented via cascaded mounts. Nache mounts the data exported by the remote server and then re-exports this mount to clients. This means that when a client request must be forwarded from the Nache to the remote server, the request must first be translated from an NFSv4 request to a VFS call and then back to an NFSv4 request. This translation works without issue in most cases; however, some stateful NFSv4 requests (e.g., OPEN, CLOSE, LOCK, REQUEST) require special handling.

Ajay then proceeded to provide an evaluation of both Nache and delegation performance in NFSv4. He showed that delegations can greatly reduce the number of operations at the remote server, but that the cost of issuing a delegation is high. Hence, delegations are useful as long as the server does not have to revoke them often (owing to conflicting accesses, etc.). The authors evaluated Nache by using the filebench benchmark, which was used to approximate both a Web-based workload and an OLTP workload, and by executing a software build on the clients. The results showed that as long as more than one client is used, the fraction of total operations generated by clients sent to the remote server was substantially reduced. Finally, the authors measured response time over the WAN when executing the software build. They found that the server response time decreased as the number of clients used increased.

One attendee asked what would happen if the remote server were to revoke the delegations issued to Nache. Ajay responded that Nache would then stop acting as a proxy and start acting as a simple passthrough.

BEYOND THE MACHINE ROOM

Summarized by Andrew Leung (aleung@soe.ucsc.edu)

■ ***TFS: A Transparent File System for Contributory Storage***

James Cipar, Mark D. Corner, and Emery D. Berger, University of Massachusetts, Amherst

Awarded Best Paper!

James Cipar discussed how contributory applications such as Folding@home and Freenet utilize a user's unused local disk space to contribute to a common distributed system. This disk utilization intrudes on users' personal free space as well as impacting normal application performance. Namely, as used disk space increases, the file system's ability to make ideal block allocation decisions decreases. This leads to disk layout fragmentation, which has a very negative impact on disk access times. James continued by saying that for users to want to use contributory applications, these applications must not consume too much disk space.

To address this issue the authors presented the Transparent File System (TFS), an on-disk file system that allows users to contribute disk space with minimal performance impact. James outlined the goals of TFS: to make contributory data transparent, meaning their presence has no impact on filesystem performance or capacity; to allow contributory data to be overwritten when space for user data is needed; and to make such data compatible with legacy contributory applications. To facilitate overwriting of contributory data, TFS uses five block allocation states to specify whether data is contributory, contributory and overwritten by user data, or overwritten contributory data that has been deleted. James evaluates the contributions of TFS with respect to a watermarking approach and using fixed contribution space. The two key metrics are the amount of storage capacity contributed and bandwidth. Their figures show TFS is able to contribute more space, as well as maintaining higher bandwidth. To examine local filesystem performance, the Andrew Benchmark is used against different amounts of data contribution. James presents the TFS run time for several benchmark phases, each of which is better than or comparable to ext2 with or without any contributory application running.

Brent Callaghan from Apple asked whether TFS could be used as a Web browser cache. James replied that in fact they had tried to use TFS as a browser cache but that because browser cache size is generally small the effects are negligible. Another question related to files that are open for extended periods of time. Since TFS does not overwrite contributory data from an open file it is conceivable that a contributory application could cause fragmentation. James confirmed that this is the case but that perhaps some effort should be made to ensure that contributory applications do not hold files open too long. Binny Gill from IBM Research asked whether ext2 had to be modified. James said that indeed ext2 was modified and that porting TFS to another file system would require the source code. A second question was whether movie data from sources such as BitTorrent is affected by TFS. James answered that BitTorrent data is user data, rather than contributory data, and therefore TFS does not affect it; rather, BitTorrent consumes other resources, such as bandwidth during uploads. Finally, James provided a link to more TFS information and source code: http://prisms.cs.umass.edu/tcsm/.

■ *Cobalt: Separating Content Distribution from Authorization in Distributed File Systems*

Kaushik Veeraraghaven, Andrew Myrick, and Jason Flinn, University of Michigan

Kaushik Veeraraghavan presented a solution for secure mobile content access. In addition to accessing digital content from personal devices, Kaushik said that users often wish to access content from other devices to share media with friends or family. He described how these ad hoc clients, which are devices such as MP3 players or laptops that a user does not own or rarely uses, make access to digital data difficult. The current data distribution and authorization model makes accessing data from ad hoc clients difficult because the ad hoc client must be used to explicitly locate and fetch remote content after searching for it over a slow, wide-area link. Kaushik described how digital rights management adds additional complexity by forcing users to authenticate themselves at each ad hoc device, opening the door for possible abuse, and often requires the user to jump through hoops to play back the content. What makes protected content special is that the goals of users and content providers are largely orthogonal. Users desire easy and pervasive access to their data, whereas providers do not want data leaked to any unauthorized device or user.

To address this, Kaushik described a shift in paradigm where people, rather than clients, are authenticated. Their solution, Cobalt, relies on a Cobalt token, such as a cell phone or PDA, which is carried by the user, allowing authentication to be based on proximity. Kaushik said the goal of Cobalt is to improve usability, privacy, and content protection. This relies on trusting the Cobalt token and ad hoc media players, each of which has a Trust Platform Module (TPM) chip that allows content providers to verify the integrity of the Cobalt token. Kaushik described their current implementation on the Blue File System. The Cobalt token manages content acquisition, where the provider forwards encrypted content to either the token or a more powerful helper computer. Playing the content requires the token to identify all media players in range, select a specific media, and send either the data from the token or the IP address of the helper computer to the media player.

Evaluation of Cobalt aims at understanding the overhead of content acquisition and content playback and any new applications that Cobalt may enable. The content acquisition has a fixed 10-second overhead that scales linearly with file size. Kaushik attributes most of this cost to the establishment of a secure connection. Associating a media player with the token and specifying content to share requires about 12 seconds. Finally, Kaushik describes how Cobalt can be used to adaptively merge many users' playlists into a single playlist, enjoyable by all. Each user specifies a query and only songs that appear in all query results are played.

Peter Honeyman from the University of Michigan asked about clarifying whether it is the TPM or the media player that is trusted, since the media player can still steal unencrypted content. Kaushik responded that the media player must also be trusted and that ideally DRM is enforced all the way until it reaches the user's ear but that this obviously cannot be done. Brent Welch from Panasas asked if Cobalt would make it possible to steal media from TiVo, to which Kaushik replied that TiVo does not have a TPM chip, which makes it untrusted. Daniel Ellard from Network Appliance asked about how content can be retrieved should the Cobalt token be lost. Kaushik said content providers could deauthorize the token or one could encrypt the content key and resynchronize with a new token. Finally, Craig Everhart from Network Appliance wondered how Cobalt defined media players in range and whether it would be possible to accidentally access a neighbor's media player. Kaushik responded that it is indeed possible but that simply registering tokens with specific media players could alleviate that problem.

MAKING THE RAID

Summarized by Charles Weddle (weddle@cs.fsu.edu)

■ *PARAID: A Gear-Shifting Power-Aware RAID*

Charles Weddle, Mathew Oldham, Jin Qian, and An-I Andy Wang, Florida State University; Peter Reiher, University of California, Los Angeles; Geoff Kuenning, Harvey Mudd College

Charles Weddle began his talk by discussing the increasing concern of energy consumption in server-class machines. Storage systems account for a significant part of the energy consumption in servers. In fact, storage systems account for 24% of the power usage in Web servers and 27% of the electricity cost for data centers. The authors hypothesize that it is possible to reduce energy consumption in RAID devices without degrading performance, while maintaining reliability. The three main challenges in this problem are finding opportunities to save energy in active servers, preserving peak performance, and maintaining reliability. The existing work mostly trades performance for energy savings directly, for example by varying the speeds of disks. The authors took advantage of three observations to help provide a solution to this problem. First, conventional RAID overprovisions resources: A conventional RAID keeps all disks spinning even under light load. Second, unused storage exists on storage servers because of overprovisioning. This unused storage can be used for energy savings. Third, workloads have a cyclic fluctuation. Infrequent disk power transitions during periods of light load can save energy.

The authors present the power-aware RAID (PARAID) as a solution to this problem. PARAID introduces skewed striping that allows PARAID to power off disks during periods of light load. Skewed striping replicates blocks in unused

storage, allowing disks to be powered off. Skewed striping creates sets of disks of varying sizes, where each disk set is thought of as a gear. Skewed striping allows PARAID to match performance to workload by switching into different gears. PARAID preserves peak performance by operating in the highest gear, with all disks in the array, when the system is under peak load. PARAID maintains reliability by reusing existing RAID levels, for example RAID level 5. The authors implemented a PARAID prototype in Linux 2.6.5 for evaluation. The PARAID components reside between the conventional RAID device driver and the disk device driver. The evaluation was performed by replaying a Web trace and a Cello99 trace and running the Postmark benchmark. For the Web trace experiments the PARAID device, using RAID level 0, was compared to a conventional RAID level 0 device and was found to reduce energy consumption by up to 34%. For the Cello99 experiments the PARAID device, using RAID level 5, was compared to a conventional RAID level 5 device, yielding up to 13% energy savings. PARAID is ongoing work and the authors would like to test PARAID under more workloads. Also, the authors would like to put PARAID into a production environment for live testing.

Bill Bolosky from Microsoft asked why the latency CDF graphs for the Web trace experiments have sustained peaks. Charles responded that this was caused by a small number of Web requests never finishing owing to the way the Web trace playback program handles certain errors. Keith Smith from Network Appliance asked whether RAID level 4 would make disk power transitions easier. Charles responded that it might help alleviate the overhead of cascading parity updates. Carl Waldspurger from VMware asked how the optimal gear configuration is chosen. Charles explained that gear optimization is ongoing work and that iterating over gear configurations using simulation might help with this problem.

■ *REO: A Generic RAID Engine and Optimizer*

Deepak Kenchammana-Hosekote, IBM Almaden Research Center; Dingshan He, Microsoft; James Lee Hafner, IBM Almaden Research Center

Deepak Kenchammana began his talk by discussing the need for a variety of RAID codes. No single RAID code satisfies all aspects of data storage in terms of storage efficiency, reliability, and performance. Also, as data sets grow, using the same RAID code is not practical because disk failures grow with capacity. A greater variety of RAID codes are needed to provide high data reliability using less reliable disks. This is challenging because it is expensive to support several RAID codes. The current mindset is that deploying new RAID codes is expensive and risky. The authors present the RAID Engine Optimizer (REO) as a solution to this problem. REO can handle any XOR-based RAID codes, including N-way mirroring. REO automatically handles all RAID-related errors and leverages dynamic-state-like current cache pages. REO offers competitive performance, yielding ~8% speedup for SPC-1-like workloads.

REO comprises a set of routines invoked by cache on read (on miss), write (flush), rebuild, and migrate. REO routines deal with all necessary RAID translations and executions. REO extends the idea that it is efficient to simultaneously flush dirty pages in cache that belong to the same stripe by defining a W-neighborhood for the victim page. The W-neighborhood is defined as the set of all pages, clean or dirty, in the data cache that are in a 2W + 1 stripe window centered on the victim's stripe. REO is made up of a RAID engine and an execution engine. The RAID engine determines the best strategy for what is to be done and the execution engine figures out how it gets done. Input into the execution engine from the RAID engine is an I/O plan that contains the set of blocks to be read, a set of blocks to be XOR-ed, and a set of blocks to be written. The read strategy for fault-free read is straightforward: Just read it from disk. The challenge with read is the reconstruct case. The write strategy involves identifying all affected parity elements for dirty elements to be written. For every affected parity element there are two update strategies: Parity Compute (PC) and Parity Increment (PI). The execution engine takes the I/O plan and acquires stripe locks, allocates cache pages, submits reads in the read set, executes XORs in the XOR set, and finally submits the writes in the write set. If an error is detected during the execution of an I/O plan, the execution engine aborts the I/O plan and resubmits the I/O plan to the RAID engine. The RAID engine contains an I/O plan optimizer that is responsible for picking the least-cost read or write strategy. Some metrics used for optimization are the number of I/O commands (IOC) submitted to the storage devices and the number of XOR operations. REO was evaluated through trace-driven simulations. The authors built the data cache and REO components for evaluation; Disksim was used to simulate disk I/O. The optimization objective for evaluation was to minimize disk I/O (IOC). Two measures were used in evaluation: total access time and total memory bus usage. REO showed a 10% improvement in disk access time for a fault-free RAID 5 layout and a 7% improvement for a RAID 5 layout with one disk failure.

The first questioner asked what REO would do under page pressure (thrashing). Deepak answered that REO would do nothing worse than any other RAID firmware would do. One way to reduce the additional pages needed to complete in-progress writes is to reduce the window size. If that does not help then you are really stuck with operating at a point where things will complete, albeit slowly. To the second question, whether the authors considered having REO look more broadly into the file cache, Deepak answered no. In response to whether there was any chance of seeing the source code, Deepak stated that he is sure that IBM has plans for it to be made public.

■ *PRO: A Popularity-Based Multi-Threaded Reconstruction Optimization for RAID-Structured Storage Systems*

Lei Tian and Dan Feng, Huazhong University of Science and Technology; Hong Jiang, University of Nebraska—Lincoln; Ke Zhou, Lingfang Zeng, Jianxi Chen, and Zhikun Wang, Huazhong University of Science and Technology and Wuhan National Laboratory for Optoelectronics; Zhenlei Song, Huazhong University of Science and Technology

Hong Jiang began his talk by discussing the importance of data recovery. Disk failures have become more common in RAID-structured storage systems. The improvement in disk capacity has far outpaced improvements in disk bandwidth, lengthening the overall RAID recovery time. Also, disk drive reliability has improved slowly, resulting in a very high overall failure rate in a large-scale RAID storage system. Disk-oriented reconstruction (DOR) is one of the existing I/O parallelism-based recovery mechanisms. DOR follows a sequential order of stripes in reconstruction, regardless of user access patterns. Workload access patterns need to be considered because 80% of the accesses are directed to 20% of the data, according to Pareto's Principle, and 10% of the files accessed on a Web server typically account for 90% of the server requests. The authors present a popularity-based multi-threaded reconstruction optimization (PRO) that takes advantage of data popularity to improve reconstruction performance. PRO divides data units on the spare disks into hot zones. Each hot zone has a reconstruction thread. The priority of each thread is dynamically adjusted according to the current popularity of its hot zone. PRO keeps track of the user accesses and adjusts the popularity of each hot zone accordingly. PRO selects the reconstruction thread with the highest priority and allocates a time slice to it. When a thread's time slice runs out, PRO assigns a time slice to the next highest priority thread. The process repeats until all of the data units have been rebuilt. Priority-based scheduling is used so that the reconstruction regions are always the hottest regions. Time-slicing is used to exploit the I/O bandwidth of hard disks and access locality.

PRO was compared to DOR in the evaluation because DOR is arguably the most effective among the existing reconstruction algorithms. The evaluation of PRO examined reconstruction performance by measuring user response time, reconstruction time, and algorithm complexity. PRO was integrated into the original DOR approach implemented in the RAIDframe software to validate and evaluate PRO. The evaluation was performed by replaying three different Web traces that consisted of read-only Web search activity. It was found that PRO consistently outperformed DOR in reconstruction and user response time by up to 44.7% and 23.9%, respectively. PRO's effectiveness relies on the existence of popularity and locality in the workload as well as the intensity of the workload. PRO also uses extra memory for each thread descriptor. The computation overhead of PRO is $O(n)$, although if a priority queue is used in the PRO algorithm the computation overhead can be reduced to $O(\log n)$. The entire PRO implementation in the RAIDFrame software only added 686 lines of code. Work on PRO is ongoing. Future work includes optimizing the time slice, scheduling strategies, and hot zone length. Currently, PRO is being ported into the Linux software RAID. Finally, the authors plan on further investigating use of access patterns to help predict user accesses and of filesystem semantic knowledge to explore accurate reconstruction.

The first questioner asked about the average rate of recovery for PRO. Hong answered that the average reconstruction time is several hundred seconds in the experimental setup. The second questioner asked how well PRO reconstruction compares to DOR reconstruction under no workload. Hong commented that when there is no workload the reconstruction performance for PRO and DOR is the same. In response to a question about write overhead, Hong stated that his research team is actively looking into this. The last question involved the sensitivity of the results to the number of threads and to the time slice. Hong explained that the impact of the number of threads and time slice is negligible in the current experimental configuration. However, a more elaborate sensitivity study is underway in the project.

LSF '07: 2007 Linux Storage & Filesystem Workshop

San Jose, CA
February 12–13, 2007
Summarized by Brandon Philips (brandon@ifup.org)

Fifty members of the Linux storage and filesystem communities met in San Jose, California, to give status updates, present new ideas, and discuss issues during the two-day Linux Storage & Filesystem Workshop. The workshop was chaired by Ric Wheeler and sponsored by EMC, NetApp, Panasas, Seagate, and Oracle.

JOINT SESSION

Ric Wheeler opened the workshop by explaining the basic contract that storage systems make with the user to guarantee that the complete set of data will be stored, bytes are correct and in order, and raw capacity is utilized as completely as possible. It is so simple that it seems that there should be no open issues, right?

Today, these basic demands are met most of the time, but Ric posed a number of questions. How do we validate that no files have been lost? How do we verify that bytes are correctly stored? How can we utilize disks efficiently for small files? How do errors get communicated between the layers?

Through the course of the next two days some of these questions were discussed, others were raised, and a few ideas were proposed. Continue reading for the details.

Ext4 Status Update

Mingming Cao gave a status update on ext4, the recent fork of the ext3 file system. The primary goal of the fork was the move to 48-bit block numbers; this change allows the file system to support up to 1024 petabytes of storage. This feature was originally designed to be merged into ext3 but was seen as too disruptive [1]. The patch is also built on top of the patch set that replaces the indirect block map and with extents [2] in ext4. Support for greater than 32K directory entries will also be merged into ext4.

On top of these changes a number of ext3 options will be enabled by default in ext4; these include directory indexing to improve file access for large directories, resize inode, which reserves space in the block group descriptor for online growing, and 256-byte inodes. Users of ext3 can use these features today by using mkfs.ext3 -I 256 -O resize_inode dir_index /dev/device.

A number of RFCs are also being considered for inclusion into ext4. This includes a patch that will add nanosecond timestamps [3] and the creation of persistent file allocations [4], which will be similar to posix_fallocate but won't waste time writing zeros to the disk.

Currently, ext4 stores a limited number of extended attributes in-inode and has space for one additional block of extended attribute data, but this may not be enough to satisfy xattr-hungry applications. For example, Samba needs additional space to support Vista's heavy use of ACLs, and eCryptFS can store arbitrarily large keys in extended attributes. This led everyone to the conclusion that someone needs to collect data on how xattrs are being used, to help developers decide how to best implement xattrs. Until larger extended attributes are supported, application developers need to pay attention to the limits that exist on current file systems (e.g., one block on ext3 and 64K on XFS).

Online shrinking and growing was briefly discussed and it was suggested that online defragmentation, which is a planned feature, will be the first step toward online shrinking. A bigger issue, however, is storage management. Ted T'so suggested that the Linux filesystem community can learn from ZFS how to create easy-to-manage storage systems. Christoph Hellwig sees the disk management issue as being a user-space problem that can be solved with kernel hooks and sees ZFS as a layering violation. Either way, it is clear that disk management should be improved.

The fsck Problem

Zach Brown and Valerie Henson were slated to speak on the topic of filesystem repair, but there was a slight delay as Val's laptop was booting. To pass the time she introduced us to the latest fashion: laptop rhinestones. They would make a great discussion piece if you are waiting on a fsck and, if Val's fsck estimates for 2013 come true, having a strategy to pass the time will become very important.

With her system booted Val presented an estimate of 2013 fsck times. She first measured a fsck of her 37-GB home directory with 21 GB in use, which took 7.5 minutes and read 1.3 GB of filesystem data. Next, she used projections of disk technology from Seagate to estimate the time to fsck a 2013 home directory, which will be 16 times larger. Although 2013 disks will have a fivefold bandwidth increase, seek times will only improve by 20%, to 10 ms, leading to a fsck time of 80 minutes! The primary reason for long fscks is seek latency, since fsck spends most of its time seeking over the disk, discovering and fetching dynamic filesystem data such as directory entries, indirect blocks, and extents.

Reducing seeks and avoiding the seek latency punishment are key to reducing fsck times. Val suggested one solution: Keeping a bitmap on disk that tracks the blocks that contain filesystem metadata; this would allow for reading all data in a single arm sweep. This optimization, in the best case, would make a single sequential sweep over the disk and on the 2013 disk reading all filesystem data would only take around 134 seconds, which is a big improvement. A full explanation of the findings and possible solutions can be found in the paper Repair-Driven File System Design [5]. Also, Val announced that she is working full time on a file system called chunkfs [6] that will make speed and ease of repair a primary design goal.

Zach Brown presented a blktrace of e2fsck. The basic outcome of the trace is that the disk can stream data at 26 Mbps and fsck is achieving 12 Mbps. This situation could be improved to some degree without on-disk layout changes if the developers had a vectorized I/O call. Zach explained that in many cases you know the block locations that you need, but with the current API you can only read one at a time.

A vectorized read would take a number of buffers and a list of blocks to read as arguments. Then the application could submit all of the reads at once. Such a system call could save a significant amount of time, since the I/O scheduler can reorder requests to minimize seeks and merge requests that are nearby. Also, reads to blocks that are located on different disks could be parallelized. Although a vectorized read could speed up the fsck, eventually filesystem layout changes will be needed to make fsck really fast.

Libata: Bringing the ATA Community Together

Jeff Garzik gave an update on the progress of libata, the in-kernel library to support ATA hosts and devices. First, he presented the ATAPI/SATA features that libata now supports: PATA+C/H/S, NCQ, FUA, SCSI SAT, and CompactFlash. The growing support for parallel ATA (PATA) drives in libata will eventually deprecate the IDE driver, and Fe-

dora developers are helping to accelerate testing and adoption of the libata PATA code by disabling the IDE driver in Fedora 7 test 1.

Native Command Queuing (NCQ) is a new command protocol introduced in the SATA II extensions and now supported under libata. With NCQ the host can have multiple outstanding requests on the drive at once. The drive can reorder and reschedule these requests to improve disk performance. A useful feature of NCQ drives is the force unit access (FUA) bit, which will ensure that data in write commands with this bit set will be written to disk before returning success. This has the potential of enabling the kernel to have both synchronous and nonsynchronous commands in flight. There was a recent discussion [7] about both NCQ FUA and SATA FUA in libata.

Jeff briefly discussed libata's support for SCSI ATA translation (SAT). SAT lets an ATA device appear to be a SCSI device to the system. The motivation for this translation is the reuse of error handling and support for distro installers, which already know how to handle SCSI devices.

There are also a number of items slated as future work for libata. Many drivers need better suspend/resume support, and the driver API is due for a sane initialization model using an allocate/register/unallocate/free system and use of "Greg blessed" kobjects. Currently, libata is written under the SCSI layer and debate continues on how to restructure libata to minimize or eliminate its SCSI dependence. Error handling has been substantially improved by Tejun Heo and his changes are now in mainline. If you have had issues with SATA or libata error handling, try an updated kernel to see whether those issues have been resolved. Tejun and others continue to add features and to tune the libata stack.

■ *Communication Breakdown: I/O and File Systems*

During the morning a number of conversations sprang up about communication between I/O and file systems. In the case of errors, file systems should be getting information on nonretryable errors and passing that data up to user space. Particularly bad can be situations where retries are happening over and over when the I/O layer knows that an entire range of blocks is missing.

A "pipe" abstraction was discussed to communicate data on byte ranges that are currently in error, under performance strain (because of a RAID5 disk failure), or temporarily unplugged. If a file system was aware of ranges that are currently handling a recoverable error, have unrecoverable errors, or are temporarily slow, it might be able to handle the situations more gracefully.

File systems currently do not receive unplug events, and handling unplug situations can be tricky. For example, if a fibre channel disk is pulled for a moment and plugged back in, it may be down for only 30 seconds, but how should the file system handle the situation? Currently, ext3 remounts the entire file system as read only. XFS has a configurable timeout for fibre channel disks that must be reached before it sends an EIO error. And what should be done with USB drives that are unplugged? Should the file system save state and hope the device gets plugged back in? How long should it wait, and should it still work if it is plugged into a different hub? All of these questions were raised but there are no clear answers.

FS TRACK

■ *Security Attributes*

Michael Halcrow, eCryptFS developer, presented an idea to leverage SELinux and make file encryption/decryption based on application execution. For example, a policy could be defined so that the data would be unencrypted when OpenOffice is using the file but encrypted when the user copies the file to a USB key. After presenting the mechanism and mark-up language for this idea, Michael opened the floor to the audience. The general feeling was that SELinux is often disabled by users and that per-mount-point encryption may be a more useful and easier-to-understand user interface.

■ *Why Linux Sucks for Stacking*

Josef Sipek, Unionfs [8] maintainer, went over some of the issues involved with stacking file systems under Linux. A stacking file system, such as Unionfs, provides an alternative view of a lower file system. For example, Unionfs takes a number of mounted directories, which could be NFS, ext3, etc., as arguments at mount time and merges their name space.

The big unsolved issue with stacking file systems is handling modifications to the lower file systems in the stack. Several people suggested that leaving the lower file system available to the user is just broken and that by default the lower layers should only be mounted internally.

The new fs/stack.c file was discussed, too. This file currently contains a simple inode copy routine that is used by Unionfs and eCryptfs, but in the future more stackable filesystem routines should be pushed to this file.

Future work for Unionfs includes getting it working under lockdep and additional experimentation with an on-disk format. The on-disk format for Unionfs is currently under development and will store white-out files and persistent Unionfs inode data.

■ *B-trees for a Shadowed FS*

Many file systems use b-trees to represent files and directories. These structures keep data sorted, are balanced, and allow for insertion and deletion in logarithmic time. However, there are difficulties in using them with shadowing. Ohad Rodeh presented his approach to using b-trees and shadowing in an object storage device, but the methods are general and useful for any application.

Shadowing may also be called copy-on-write (COW). The basic idea is that when a write is made the block is read into memory, modified, and written to a new location on disk. Then the tree is recursively updated, starting at the child and using COW, until the root node is atomically updated. In this way the data is never in an inconsistent state; if the system crashes before the root node is updated then the write is lost but the previous contents remain intact.

Replicating the details of his presentation would be a wasted effort as his paper, "B-trees, Shadowing and Clone" [9], is well written and easy to read. Enjoy!

■ *eXplode the Code*

Storage systems have a simple and important contract to keep: Given user data, they must save that data to disk without loss or corruption even in the face of system crashes. Can Sar gave an overview of eXplode [10], a systematic approach to finding bugs in storage systems, to help root out the bugs that can break this contract.

eXplode systematically explores all possible choices that can be made at each choice point in the code to make low-probability events, or corner cases, just as probable as the main running path. And it does this exploration on a real running system with minimal modifications.

This system has the advantage of being conceptually simple and very effective. Bugs were found in every major Linux file system, including a fsync bug that can cause data corruption on ext2. This bug can be produced by doing the following: Create a new file, B, which recycles an indirect block from a recently truncated file, A, then call fsync on file B and crash the system before file A's truncate gets to disk. There is now inconsistent data on disk and when e2fsck tries to fix the inconsistency it corrupts file B's data. A discussion of the bug has been started on the linux-fsdevel [11] mailing list.

FS TRACK

■ *NFS*

The second day of the file systems track started with a discussion of an NFS race. The race appears when a client opens up a file between two writes that occur during the same second. The client that just opened the file is unaware of the second write and keeps an out-of-date version of the file in cache. To fix the problem a change attribute was suggested. This number would be consistent across reboots, would be unitless, and would increment on every write.

In general everyone agreed that a change attribute is the right solution; however, Val Henson pointed out that implementing this on legacy file systems will be expensive and will require on-disk format changes.

Discussion then turned to NSFv4 access control lists (ACLs). Trond Myklebust said they are becoming standard and Linux should support them. Andreas Gruenbacher is working on patches to add NFSv4 support to Linux but currently only ext3 is supported; more information can be found on the Native NFSv4 ACLs on the Linux [12] page. A possibly difficult issue will be mapping current POSIX ACLs to NFSv4 ACLs, but a draft document, "Mapping Between NFSv4 and Posix Draft ACLs" [13], lays out a mapping scheme.

■ *GFS Updates*

Steven Whitehouse gave an overview of the recent changes in the Global File System 2 (GFS2), a cluster file system where a number of peers share one large file system. The important changes include a new journal layout that can support mmap, splice, and other system calls on journaled files, page cache level locking, readpages() and partial writepages() support, and ext3 standard ioctls lsattr and chattr.

The readdir() function was discussed at some length, particularly the ways in which it is broken. A directory insert on GFS2 may cause a reorder of the extendible hash structure GFS2 uses for directories. In order to support readdir, every hash chain must be sorted. The audience generally agreed that readdir is difficult to implement and Ted Ts'o suggested that someone should try to go through committee to get telldir/seekdir/readdir fixed or eliminated.

■ *OCFS2*

A brief OCFS2 status report was given by Mark Fasheh. Like GFS2, OCFS2 is a cluster file system, designed to share a file system across nodes in a cluster. The current development focus is on adding features, as the basic filesystem features are working well.

After the status update the audience asked a few questions. The most requested OCFS2 feature is forced unmount and several people suggested that this should be a future virtual filesystem (vfs) feature. Mark also said that users really enjoy the easy setup of OCFS2 and the ability to use it as a local file system. A performance hot button for OCFS2 is the large inodes that occupy an entire block.

In the future Mark would like to mix extent and extended attribute data in-inode to utilize all of the available space. However, as the audience pointed out, this optimization can lead to some complex code. In the future Mark would also like to move to GFS's distribute lock manager.

■ *DualFS: A New Journaling File System for Linux*

DualFS is a file system by Juan Piernas that separates data and metadata into separate file systems. The on-disk format for the data disk is similar to ext2 without metadata blocks. The metadata file system is a log file system, a design that allows for very fast writes, since they are always made at the head of the log, which reduces expensive

seeks. A few performance numbers were presented: under a number of micro- and macro-benchmarks, DualFS performs better than other Linux journaling file systems. In its current form, DualFS uses separate partitions for data and metadata; this forces the user to answer a difficult question: How much metadata do I expect to have?

More information, including performance comparisons, can be found on the DualFS LKML announcement page [14] and the project homepage [15]. The currently available code is a patch on top of 2.4.19 and can be found on SourceForge [16].

■ *pNFS Object Storage Driver*

Benny Halevy gave an overview of pNFS (parallel NFS), which is part of the IETF NFSv4.1 draft [17] and tries to solve the single-server performance bottleneck of NFS storage systems. pNFS is a mechanism for an NFS client to talk directly to a disk device without sending requests through the NFS server, fanning the storage system out to the number of SAN devices. There are many proprietary systems that do a similar thing, including EMC's High Road, IBM's TotalStorage SAN, SGI's CXFS, and Sun's QFS. Having an open protocol would be a good thing.

However, Jeff Garzik was skeptical of including pNFS in the NFSv4.1 draft particularly because to support pNFS the kernel will need to provide implementations of all three access protocols: file storage, object storage, and block storage. This will add significant complexity to the Linux NFSv4 implementation.

Benny explained that the pNFS implementation in Linux is modular to support multiple layout-type specific drivers, which are optional. Each layout driver dynamically registers itself using its layout type and the NFS client calls it across a well-defined API. Support for specific layout types is optional. In the absence of a layout driver for some specific layout type, the NFS client falls back to doing I/O through the server.

After this overview Benny turned to the topic of OSDs: object-based storage devices. These devices provide a more abstract view of the disk than the classic "array of blocks" abstraction seen in today's disks. Instead of blocks, objects are the basic unit of an OSD, and each object contains both metadata and data. The disk manages the allocation of the bytes on disk and presents the object data as a contiguous array to the system. Having this abstraction in hardware would make filesystem implementation much simpler. To support OSDs in Linux Benny and others are working to get bi-directional SCSI command support into the kernel and support for variable-length command descriptor blocks (CDBs).

■ *Hybrid Disks*

Hybrid disks with an NVCache (flash memory) will be in consumers' hands soon. Timothy Bisson gave an overview of this new technology. The NVCache will have 128–256 MB of nonvolatile flash memory that the disk can manage as a cache (unpinned) or the operating system can manage by pinning specified blocks to the nonvolatile memory. This technology can reduce power consumption or increase disk performance.

To reduce power consumption, the block layer can enable the NVCache Power Mode, which tells the disk to redirect writes to the NVCache, thereby reducing disk spin-up operations. In this mode the 10-minute write-back threshold of Linux laptop mode can be removed. Another strategy is to pin all filesystem metadata in the NVCache, but spin-ups will still occur on nonmetadata reads. An open question is how this pinning should be managed when two or more file systems are using the same disk.

Performance can be increased by using the NVCache as a cache for writes, resulting in a long seek. In this mode the block layer would pin the target blocks, ensuring a write to the cache instead of incurring the expensive seek. Also, a file system can use the NVCache to store its journal and boot files for additional performance and reduced system start-up time.

If Linux developers decide to manage the NVCache there are many open questions. Which layer should manage the NVCache, the file system or block layer? And what type of API should be created to leverage the cache? Another big question is how much punishment these caches can take. According to Timothy it takes about a year (using a desktop workload) to fry the cache if you are using it as a write cache.

■ *Scaling Linux to Petabytes*

Sage Weil presented Ceph, a network file system that is designed to scale to petabytes of storage. Ceph is based on a network of object-based storage devices, and complete copies of each object are distributed across multiple nodes, using an algorithm called CRUSH. This distribution makes it possible for nodes to be added and removed from the system dynamically. More information on the design and implementation can be found on the Ceph homepage [18].

CONCLUSION

The workshop concluded with the general consensus that bringing together SATA, SCSI, and filesystem people was a good idea and that the status updates and conversations were useful. However, the workshop was a bit too large for code discussion. More targeted workshops will need to be held to work out the details of some of the issues discussed at LSF '07. Topics for future workshops include virtual memory and filesystem issues and extensions that are needed to the VFS.

REFERENCES

[1] http://lwn.net/Articles/187336/.

[2] http://lwn.net/Articles/187321/.

[3] http://article.gmane.org/gmane.comp.file-systems.ext4/986.

[4] http://article.gmane.org/gmane.comp.file-systems.ext4/899.

[5] http://infohost.nmt.edu/~val/review/repair.pdf.

[6] http://www.usenix.org/events/hotdep06/tech/prelim_papers/henson/henson.pdf.

[7] http://article.gmane.org/gmane.linux.ide/15942/.

[8] http://unionfs.filesystems.org.

[9] http://www.cs.huji.ac.il/~orodeh/papers/ibm-techreport/H-0245.pdf.

[10] www.stanford.edu/~engler/explode-osdi06.pdf.

[11] http://marc.theaimsgroup.com/?l=linux-fsdevel&m=117148291716485&w=2.

[12] http://www.suse.de/~agruen/nfs4acl/.

[13] http://www.citi.umich.edu/projects/nfsv4/rfc/draft-ietf-nfsv4-acl-mapping-03.txt.

[14] http://lwn.net/Articles/221841/.

[15] http://ditec.um.es/~piernas/dualfs/.

[16] http://sourceforge.net/project/showfiles.php?group_id=187143&package_id=218377.

[17] http://www.nfsv4-editor.org/drafts/drafts.html.

[18] http://ceph.sourceforge.net/.

Announcement and Call for Papers **USENIX**

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

USENIX, the Advanced Computing Systems Association, in cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee (MSSTC), and IEEE TCOS

http://www.usenix.org/fast08

February 26–29, 2008 **San Jose, CA, USA**

Important Dates

Paper submissions due: *September 12, 2007, 9:00 p.m. EDT (firm deadline)*
Notification of acceptance: *November 1, 2007*
Final papers due: *January 8, 2008*
Work-in-Progress Reports/Poster Session proposals due: *January 16, 2008*

Conference Organizers

Program Chairs

Mary Baker, *Hewlett-Packard Labs*
Erik Riedel, *Seagate Research*

Program Committee

Andrea C. Arpaci-Dusseau, *University of Wisconsin, Madison*
Randal Burns, *Johns Hopkins University*
Howard Gobioff, *Google*
Christos Karamanolis, *VMware*
Kim Keeton, *Hewlett-Packard Labs*
Geoff Kuenning, *Harvey Mudd College*
Darrell Long, *University of California, Santa Cruz*
Petros Maniatis, *Intel Research Berkeley*
Robert Morris, *Massachusetts Institute of Technology*
Brian Noble, *University of Michigan*
Alma Riska, *Seagate Research*
Antony Rowstron, *Microsoft Research, UK*
Jiri Schindler, *Network Appliance*
Margo Seltzer, *Harvard University*
Doug Terry, *Microsoft*
Theodore Ts'o, *IBM*
Andrew Warfield, *University of British Columbia*
Ric Wheeler, *EMC*
Theodore Wong, *IBM Research*
Erez Zadok, *Stony Brook University*

Steering Committee

Andrea C. Arpaci-Dusseau, *University of Wisconsin, Madison*
Remzi H. Arpaci-Dusseau, *University of Wisconsin, Madison*
Jeff Chase, *Duke University*
Greg Ganger, *Carnegie Mellon University*
Garth Gibson, *Carnegie Mellon University and Panasas*
Peter Honeyman, *CITI, University of Michigan, Ann Arbor*
Merritt Jones, *MITRE Corporation*
Darrell Long, *University of California, Santa Cruz*
Jai Menon, *IBM Research*
Margo Seltzer, *Harvard University*
Chandu Thekkah, *Microsoft Research*
John Wilkes, *Hewlett-Packard Labs*
Ellie Young, *USENIX Association*

Overview

The 6th USENIX Conference on File and Storage Technologies (FAST '08) brings together storage system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems. The conference will consist of two and a half days of technical presentations, including refereed papers, Work-in-Progress reports, and a poster session.

Topics

Topics of interest include but are not limited to:

- Archival storage systems
- Caching, replication, and consistency
- Database storage issues
- Distributed I/O (wide-area, grid, peer-to-peer)
- Empirical evaluation of storage systems
- Experience with deployed systems
- Manageability
- Mobile and personal storage

- Parallel I/O
- Performance
- Reliability, availability, disaster tolerance
- Scalability
- Security
- Storage networking
- Virtualization

Deadline and Submission Instructions

Submissions will be made electronically via a Web form, which will be available on the FAST '08 Call for Papers Web site, http://www.usenix.org/fast08/cfp. The Web form asks for contact information for the paper and allows for the submission of your full paper file in PDF format.

Submissions must be full papers (no extended abstracts) and must be no longer than thirteen (13) pages plus as many additional pages as are needed for references (e.g., your paper can be 16 total pages, as long as the last three or more are the bibliography). Your paper should be typeset in two-column format in 10 point type on 12 point (single-spaced) leading, with the text block being no more than 6.5" wide by 9" deep. Submissions longer than this will not be reviewed.

Authors must not be identified in the submissions, either explicitly or by implication (e.g., through the references or acknowledgments). Blind reviewing of full papers will be done by the program committee, assisted by outside referees. Accepted papers will be shepherded through an editorial review process by a member of the program committee.

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. 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 chairs, fast08chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Accepted material may not be subsequently published in other conferences or journals for one year from the date of acceptance by USENIX. Papers accompanied by nondisclosure agreement forms will not be read or reviewed. All submissions will be held in confidence prior to publication of the technical program, both as a matter of policy and in accordance with the U.S. Copyright Act of 1976. Submissions violating these rules or the formatting guidelines will not be considered for publication.

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(s) at the conference.

Work-in-Progress Reports and Poster Session

The FAST technical sessions will include slots for Work-in-Progress reports, preliminary results, "outrageous" opinion statements, and a poster session. We are particularly interested in presentations of student work. Please send WiP submissions to fast08wips@usenix.org.

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 Conference Department at bofs@usenix.org. BoFs may also be scheduled at the conference.

Registration Materials

Complete program and registration information will be available in November 2007 on the conference Web site. The information will be in both HTML and a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list: http://www.usenix.org/about/mailing.html.

Announcement and Call for Papers **USENIX**

5th USENIX Symposium on Networked Systems Design & Implementation (NSDI '08)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

http://www.usenix.org/nsdi08

April 16–18, 2008 **San Francisco, CA, USA**

Important Dates

Paper titles and abstracts due: *October 2, 2007*
Complete paper submissions due: *October 9, 2007*
Notification of acceptance: *December 21, 2007*
Papers due for shepherding: *January 25, 2008*
Final papers due: *February 19, 2008*

Conference Organizers

Program Chairs

Jon Crowcroft, *University of Cambridge*
Mike Dahlin, *University of Texas at Austin*

Program Committee

Paul Barham, *Microsoft Research*
Ken Birman, *Cornell University*
Miguel Castro, *Microsoft Research*
Jeff Chase, *Duke University*
Steve Gribble, *University of Washington*
Matthias Grossglauser, *EPFL*
Krishna Gummadi, *Max Planck Institute for Software Systems*
Steven Hand, *University of Cambridge*
Brad Karp, *University College, London*
Dina Katabi, *Massachusetts Institute of Technology*
Eddie Kohler, *University of California, Los Angeles*
Sue Moon, *KAIST*
Robert Morris, *Massachusetts Institute of Technology*
Sylvia Ratnasamy, *Intel Research*
Luigi Rizzo, *ICIR*
Timothy Roscoe, *ETH Zürich*
Srinivasan Seshan, *Carnegie Mellon University*
Emin Gün Sirer, *Cornell University*
Amin Vahdat, *University of California, San Diego*
Arun Venkataramani, *University of Massachusetts Amherst*

Steering Committee

Thomas Anderson, *University of Washington*
Mike Jones, *Microsoft Research*
Greg Minshall
Robert Morris, *Massachusetts Institute of Technology*
Mike Schroeder, *Microsoft Research*
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:

- Novel architectures for communications systems
- Mobility and wireless system architecture challenges
- Sensor networking and other energy-constrained systems
- Novel operating system support for networked systems

- Virtualization and resource management for networked systems
- Highly available and reliable networked systems
- Security and resilience of networked systems
- Overlays and peer-to-peer systems
- Distributed storage, caching, and query processing
- Federated, autonomous, and self-configuring networked systems
- Large-scale networked systems testbeds, design, and evaluation
- Network measurements, workload, and topology characterization
- Managing, debugging, and diagnosing problems in networked systems
- Practical protocols and algorithms for networked systems
- Application experiences based on 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. Papers that do not meet the requirements on size and format 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 2, 2007, and the corresponding full paper is due by October 9, 2007. All papers must be submitted via the Web form, which will be available on the Call for Papers Web site, http://www.usenix.org/nsdi08/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 January 25, 2008. 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 19, 2008, 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 Chairs, nsdi08chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

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. BoFs may also be scheduled at the conference.

Registration Materials

Complete program and registration information will be available in January 2008 on the conference Web site. The information will be in both HTML and a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list at http://www.usenix.org/about/mailing.html.

;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