;login:

THE MAGAZINE OF USENIX & SAGE
June 2001 • Volume 26 • Number 3

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

USENIX Upcoming Events

10th USENIX Security Symposium

AUGUST 13–16, 2001
Washington, D.C., USA

http://www.usenix.org/events/sec01/

Registration materials now available

5th Annual Linux Showcase and Conference

Co-sponsored by USENIX & Atlanta Linux Showcase, in cooperation with Linux International

NOVEMBER 6–10, 2001
Oakland, California, USA

http://www.linuxshowcase.org

Notification of Acceptance: July 23, 2001

Registration materials available: August, 2001

Final Papers due: September 14, 2001

XFree86 Technical Conference

(Co-located with the 5th Annual Linux Showcase & Conference)

NOVEMBER 7-8, 2001
Oakland, California, USA

http://www.usenix.org/events/xfree86/

Refereed talk abstracts due: July 3, 2001

Notification to authors: July 23, 2001

Optional camera-ready final papers due: September 14, 2001

15th Systems Administration Conference (LISA 2001)

Sponsored by USENIX & SAGE

DECEMBER 2–7, 2001
San Diego, California, USA

http://www.usenix.org/events/lisa2001

Notification of acceptance: July 2001

Registration materials available: September, 2001

Camera-ready final papers due: October 1, 2001

First Conference on File and Storage Technologies (FAST)

JANUARY 28–29, 2002
Monterey, California, USA

http://www.usenix.org/events/fast/

Submissions due: July 13, 2001

Notification to authors: September 14, 2001

Registration materials available: October, 2001

Camera-ready final papers due: November 15, 2001

BSDCon 2002

FEBRUARY 11-14, 2002
San Francisco, California, USA

Submissions due: early September, 2001

The Systems and Network Administration Conference (SNAC)

Sponsored by USENIX & SAGE

MARCH 17-21, 2002
Dallas, Texas, USA

2002 USENIX Annual Technical Conference

JUNE 9–14, 2002
Monterey, California, USA

16th Systems Administration Conference (LISA 2002)

Sponsored by USENIX & SAGE

NOVEMBER 3–8, 2002
Philadelphia, Pennsylvania, USA

For a complete list of future USENIX events, see **http://www.usenix.org/events**

contents

;login: Vol. 26 #3, June 2001

;login: is the official magazine of the USENIX Association and SAGE.

;login: (ISSN 1044-6397) is published bimonthly, plus July and November, by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$50 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $60 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.

Cover photo: Hard at work at OSDI 2000 (l-r: Dave Presotto, Frans Kaashoek, Ellie Young)

motd

by Rob Kolstad

Dr. Rob Kolstad has long served as editor of *;login:*. He is also head coach of the USENIX-sponsored USA Computing Olympiad.

<kolstad@usenix.org>

Excellence

On Debbie Scherrer's recommendation from several years ago, I took my parents to view the "World Famous" Lippazzaner Stallions last night at their Pueblo, Colorado tour stop. I had no expectations, having ridden horses a few times and even enjoyed seeing well-made photographs of them. Most people's favorite part is the "airs": the jumping and rearing up (as in "Hi ho Silver, away!" from the Lone Ranger intro). Those horses were definitely coordinated, ever moreso considering they weigh in four digits. As horses go, they were excellent.

We hear a lot about excellence in our industry. You can hardly turn through a trade magazine without someone extolling the excellence of their product, their service, their record, or their management style. Entire books discuss excellence. I think it's a desirable quality.

Of course, excellence is decidedly *not* perfection. Perfection is yet another step and is seen only rarely in our world. There are those who felt Michael Jordan's basketball skills bordered on perfection; others admire Tiger Woods's golfing performances. Let's stick to excellence for this discussion – perfection is just really challenging in general.

Personally, I like excellence. I thought Dan Klein's talk from several years ago about the parallels between Blazon (a language for describing coats-of-arms) and Postscript was excellent. I think some of Mark Burgess's thoughts on system administration are excellent, even though I think I disagree with a few of the others. We occasionally see an excellent article in *;login:* or an excellent talk at a conference. It's really great.

Making excellent software or creating a site that is administered excellently is really challenging. The more skilled one becomes at creating either programs or environments, the more flaws one sees on the way to excellence. This is the worst sort of reward for improving skills! "I'm better now and I can see that I've got to spend much more time on the tasks I'm solving." How awful.

On the other hand, it also raises the bar in our world. Many of the technology-oriented parts of our lives truly have improved. Satellite-delivered home entertainment has a technical quality (i.e., lines per inch and lack of interference) that was simply never reached with old-style broadcasting through the ether. You can find technological success stories throughout the world from hybrid rice that feeds India to better ceramics for creating lighter engine parts. The *Science News* weekly newsmagazine has examples of these sorts of things every issue.

How does one achieve excellence in any one thing? I don't know exactly. I do know, however, that really excellent results seem to take a lot of time, maybe even an incredible amount of time.

Have you ever deleted a program and had to re-create it from memory? Remember how it took only a fraction as long to create and the new program ran better, cleaner, faster, and was easier to maintain? This seems to offer a clue: excellence rarely seems to appear quickly or "the first time." Excellence appears to require not only inspiration but sharpening, tuning, and all sorts of refinements often not visible at the onset of creation.

I've tried to do excellent things for several years (e.g., excellent programming contests for the USACO students). It's hard – but I think I'm getting better at it. I wish I could say the same for several of my other endeavors.

I think it's fun to think about excellence. Maybe there's something you do that is or could be excellent. Sometimes, I think creating or doing something excellent (a program, great peanut brittle, playing duo-piano, a super clear man page) has a bit of its own reward just in its birth. I hope you'll consider doing anything excellent – I think the world will be a better place.

apropos

Radio Buttons and Resumes

by Tina Darmohray

Tina Darmohray, co-editor of *;login:*, is a computer security and networking consultant. She was a founding member of SAGE.

<tmd@usenix.org>

A while back I had Dave Clark, owner of a system administration recruiting and placement company, write a series of articles about preparing for a job search. One of his articles focused on how to prepare a good resume. While there are variations on the theme, for the most part, resume format has been a fairly static entity. At least that's what I thought until I came across an interesting twist during some Web surfing the other day.

Stanford has a Web site full of information about how to apply for positions with the University. There's a searchable database of open positions and information on how to apply. As expected, there's specific information regarding resumes. The Web site covers all the essentials: where to send your resume, how to send it, and even provides a tool to assist you in constructing an online resume if you don't already have one. The part I didn't anticipate was the detailed information on how to prepare your resume in order to maximize a computer's ability to scan it.

It's not surprising that a large organization is using a computer to scan resumes. What I found amazing, however, were the "tips" for writing a resume for just such a scan. First off, there is consideration for the mechanical end of scannability. These are things that will allow the scanner to read the information on the page accurately, like color of paper, acceptable fonts, spacing, expected headings, placement of key information, lack of "fancy treatments," and overall layout apparently all make a difference. So far so good. Here's the twist: you don't just maximize for optical scannability, you can also create the resume to "maximize hits." For me, it's these optimizations that make a good machine-readable resume different from a good human-readable resume. Apparently, in order to increase your "hits" you should "use key words" to describe your skills, be concise, and use "jargon and acronyms." In fact, it looks like "increasing your list of key words" is a feature. Suddenly it was clear to me why a litany of operating systems (you know, every version listed separately), programming and scripting languages, hardware platforms, application packages, protocol names, and anything else just short of the kitchen sink now appears under the "Skills" heading on so many resumes!

Stanford, apparently, is pretty focused on this scannability and optimization. I got the feeling that any resume intended for a human really didn't have much of a chance in their screening process. In fact, the last little tidbit on their resume Web page actually came out and said what I suspected. It suggests that you may want to have two versions of your resume: "One for the computer to read" and "One for people to read. Carry this one to the interview with you." I had to wonder if the ultimate machine-scannable resume might just have a name, address, and telephone number, followed by a Skills heading, with as many buzzwords and acronyms as possible, and just skip the prose that attempts to explain what it is that one has actually *done* for a living. Makes me wanna try it, just to test my theory.

When I described what I'd learned about the "modern" resume to a friend later that day, he suggested that going this far overboard stopped just shy of radio buttons and checkoff boxes on a Web page. Now that he mentions it, it might save us all a lot of trouble . . . .

letters to the editor

BEWARE OF WHAT YOU WISH FOR

from Frederick M Avolio
<fred@avolio.com>

Tina:

I am not sure if you were writing with "tongue-in-cheek," but assuming not, I'm happy to suggest why no one will share security policies and acceptable use guides. No one believes theirs is any good.

There is no wizardry involved, yet there is still what borders on shame. When really, most IT professionals can put together a good and useful set of documents, far better than nothing at all.

LIABILITY RISK OF BEING USED AS A JUMPING OFF POINT FOR AN ATTACK

From Toby Kohlenberg
<toby@seaport.net>

To John Nicholson:

I just read your article in Vol. 26, No. 2 of *;login:* and wanted to compliment you on it and ask a follow-up question or two. You describe quite well the liability of an organization when they have failed to sufficiently protect data they have about a customer/user/employee/whomever, but what about the liability of an organization if their systems are cracked and then used as a jumping off point for further attacks – either destructive attacks such as DDoS or compromising attacks where data or resources may be stolen? I seem to recall this coming up a couple of times during the Yahoo/eBay/others debacle last year, but I don't remember what the outcomes were. Can you provide any further information?

John Nicholson replies:

Thanks for the positive feedback. I really appreciate it. If you ever read one of my articles and you think I've gotten something wrong, please also let me know. Also, if you ever have any ideas for a topic for an article, I'd love to hear them.

As far as your question is concerned, the prospect of using someone's computer as a platform for launching attacks is an interesting one (and one that I probably should have addressed).

If you fail to use "reasonable efforts" to keep your server secure, then it's very possible that you could be found liable for damage done to other computers. It goes back to the issue of proximate cause. The logic is similar to bars/bartenders being held liable for the damage caused by a drunk driver. The bartender failed to use "reasonable" caution in serving drinks to someone who was "reasonably obviously" intoxicated. Therefore, the logic goes, the bar/bartender should be at least partly responsible for the damage done by the drunk driver. Another example might be if a gun shop owner fails to "reasonably" properly secure his shop and someone breaks in and takes a gun and ammunition. Since society wants to encourage the gun shop owner to realize that there could be consequences to failing to secure such a potentially dangerous product, a court could find the gun shop owner liable for damage done or crimes committed by the criminal.

As far as I know, no one has ever actually taken legal action against a company because a cracked box on that company's network was used as an attack platform.

So, there's the warning about potential consequences. The other half of the article was intended to propose developing policies that would protect a company if the issue ever went to court.

From a policy point of view, as far as preventing a cracker from using a cracked server as a platform for attacking others, you might include in the definition of what is a "reasonable" security policy having some kind of restrictions and sniffing on outgoing traffic. If your firewall restricts outgoing traffic (assuming that the cracker hasn't gotten into your firewall, too), then that still might prevent a cracked box from being used as an attack platform. Alternatively, if you have some kind of auditing function that sniffs outgoing traffic and fires off an alert if outgoing traffic fits a certain profile, then that might also be sufficient.

Hope this answers your questions. If you have any other questions, feel free to drop me a note any time.

conference reports

This issue's reports are on the Linux 2.5 Kernel Developers Summit and on the 4th Symposium on Operating Systems Design and Implementation (OSDI 2000)

OUR THANKS TO THE SUMMARIZERS:

FOR THE LINUX 2.5 KERNEL DEVELOPERS SUMMIT:

Rik Farrow, with thanks to La Monte Yarroll and Chris Mason for sharing their notes.

FOR OSDI 2000:

M. Rasit Eskicioglu for organizing and editing the reports.

Darrell Anderson
Tamara Balac
Vijay Gupta
David Oppenheimer
Tep Narula
Mac Newbold

For additional information on the Linux 2.5 Kernel Developers Summit, see the following sites:

<http://lwn.net/2001/features/KernelSummit/>

<http://cgi.zdnet.com/slink?91362:12284618>

<http://www.osdn.com/conferences/kernel/>

Linux 2.5 Kernel Developers Summit

San Jose, California March 30-31, 2001

Summarized by Rik Farrow

The purpose of this workshop was to provide a forum for discussion of changes to be made in the 2.5 release of Linux (a trademark of Linus Torvalds). I assume that many people reading this will be familiar with Linux, and I will attempt to explain things that might be unfamiliar to others. That said, the odd-numbered releases, like 2.3 and now 2.5, are development releases where the intent is to try out new features or make large changes to the kernel. The even-numbered releases are considered the stable releases.

I got my first impression of the people attending the conference at the Thursday night reception. There was only one woman out of the 65 registered attendees, but other than that, this appeared very similar to any other USENIX event. The real difference is that few of these people were system administrators, and most were kernel hackers. I walked around asking people what their focus area in the kernel was. That question turned out to be hard to answer, even though I could make some guesses by looking at the Kernel Developer's mailing list traffic summary: *<http://kt.zork.net/kernel-traffic/latest.html>*. For example, Rik van Riel, originally from the Netherlands but now working for Conectiva in Brazil, focuses on virtual memory and memory management (VM and MM). I also met one or two "kernel janitors," programmers who clean up kernel code, remove defunct code, etc.

I was also pleased to discover that this meeting, put on by USENIX and OSDN (*<http://www.osdn.com>*) and sponsored by IBM, EMC, and AMD, was the first opportunity for many of these people to meet in person. Perhaps this is not so amazing given the distributed nature of Linux development, but I certainly thought that, in all of this time, someone would have brought this group together before.

Another difference appeared when the first session started on Friday morning. The conference room was set up with circular tables, each with power strips for laptops, and only a few attendees were not using a laptop. USENIX had provided Aeronet wireless setup via the hotel's T1 link, and people were busy typing and compiling. Chris Mason of OSDN noticed that Dave Miller had written a utility to modulate the speed of the CPU fans based upon the temperature reading from his motherboard. Another person whipped up a quick program to test an assertion made by the first presenter about degraded performance in the 2.4 release.

Requirements for a High Performance Database

Lance Larsh, Oracle Corporation

If you thought that having big business make suggestions about improving the Linux kernel would be poorly received, you would be wrong. In fact, I could tell that attendees were mostly receptive, as they want Linux to become a better commercial OS platform.

Larsh began by explaining a little about how Oracle works. He also told us that databases like to do raw I/O, and that this is not much of a benefit in the current Linux implementation. For example, even if a continuous batch of sectors is to be written, the kernel breaks it up into smaller batches and adds a buffer header to each sector. Another problem had to do with the elevator algorithm, which sorts and merges requests based on their physical location on a hard drive (spindle). Another problem involved io_request_lock, a global lock that Larsh suggested should be per device unless global synchronization was really required.

Larsh also suggested that Linux do away with the elevator algorithm and let the hardware do the work. Linus Torvalds asked if Larsh had tried setting some elvtune parameter to one, and Larsh said he hadn't. One thing that became clear to me was that most of the Linux kernel developers were software guys (something that Andre Hedrick really made a point of later). Modern hard drives reorder the physical location of tracks on the fly based on the current location of the heads, so using any elevator algorithm makes little sense.

Oracle also has problems with the memory model used by Linux. Some IA32 (Intel x86) -based systems can have in excess of 4GB of RAM, but Linux device drivers handle this by using a bounce buffer to copy data to a region below 1GB, losing performance to the copy. Asynchronous I/O was also a problem, as is the use of the O_SYNC and O_DSYNC flags. Quite a lively debate started at this point, with one participant saying that O_DSYNC was the default in 2.4.

Then Larsh dropped a bombshell. He reported that an SMP system with SCSI drives was 10 to 15 times slower, measured with iozone, in 2.4 than in 2.2. This effect does not show up with IDE drives.

Oracle would also like support for large page sizes. Richard Henderson said that this would also benefit scientific applications. This topic came up again on Saturday during van Riel's presentation about MM. In general, Oracle wants things standardized across as many OS platforms as possible, in the same way, for example, that shared memory is.

This session went 17 minutes over schedule and ended with an exchange between Linus, Larsh, and Alan Cox. Linus suggested fast semaphores for scheduling rather than spin locks, and Cox suggested that the user space spin locks work like Mozilla. "What Mozilla is doing is counting how many times you spin on locks before giving up, assuming that some other process has been locked but is not scheduled."

Ted Ts'o, who moderated the event, called a break at that point. Breaks were always 30 minutes, giving ample time for discussion.

SCTP

La Monte H.P. Yarroll, Motorola

La Monte Yarroll described a new protocol that will be peer to UDP and TCP (layer four for OSI fans). SCTP stands for Stream Control Transmission Protocol (RFC2960) and has several design goals:

- Sequenced delivery of user messages within multiple streams
- Network-level fault tolerance through support of multi-homing at either or both ends of an association
- MTU set at layer four to prevent fragmentation at the IP layer
- Optional bundling of multiple messages within the same packet

SCTP (*<http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2960.html>*) is very long (134 pages), but Yarroll stated that there are already 24 implementations that interoperate. Essentially, SCTP combines the reliability of TCP with the ability to send messages, even multiple message streams, over the same connection. It has some built-in fail-over because it supports the concept of multi-homing: that is, a single server can listen at multiple interfaces, and if one path quits responding, it can resume the same connection using a different interface and presumably another path.

Someone asked if SCTP can do load balancing, and Yarroll responded that this is an open research issue with no known solution. "First, slag one network, move over to another one, slag that one, move back, and so on," quipped Yarroll. Someone else asked whether SCTP was any better than TCP in connection setup and breakdown. Yarroll reported that SCTP is somewhat better, as only the third packet used in setup can carry data, and that multiple streams can use the same connection. Also, there are no bitwise flags, and all options are word-aligned.

Someone else asked if there is any talk of moving part of the protocol into hardware. Yarroll answered, "It is a dream. There are a lot of properties that should make SCTP hardware implementable." Ted Ts'o pointed out that fiber channels are very expensive, and SCSI over SCTP would be a viable option.

During the break, Stephen Tweedie, the next presenter, moved toward the front and Linus intercepted him at the table where I was sitting. Soon, Ben LaHaise joined in a spirited discussion about zero copy writes. Zero copy writes avoid the performance hit of a memory to memory copy, and Linus shared his skepticism about how it is being implemented. My impression was of a professor with not a lot of seniority arguing with his grad students and other professors. At one point, Linus said something that I thought was very revealing: "We don't want to wind up like Windows NT with lots of subtle bugs. We want stability over performance."

Block Device Layer

Stephen Tweedie, RedHat

Ts'o introduced this session by joking that it was completely uncontroversial and not relevant to the kernel. This was a fitting beginning.

Tweedie began by discussing scalability issues. These include large numbers of devices, large devices (current 2TB limit), 512-byte block size being a problem for large disks, and related SCSI issues, like large numbers of logical units. He jumped next to robustness. Currently, information from the SCSI layer does not pass to higher layers, so a one-bit error could result in a RAID disk being taken offline.

Broaching the issue of under-performance, Tweedie stated that the kernel cannot pass in single I/Os that are con-

tiguous on disk but discontinuous in memory. "Each I/O merge is a scan of up to 8,192 requests!" Tweedie said. Scheduling could be made more fair by scheduling per spindle rather than for host bus adapter.

Device naming has been and will continue to be an issue. There are worldwide namings in fiber channel, SCSI naming by probe order, and the same with IDE devices. Ts'o mentioned that in some cases you will not be able to enumerate all devices at boot time, and sixteen bits for dev_t (that holds the minor device number) will not be enough.

Jens Axbone mentioned that he had done some tests with Andre Hedrick where they moved the io_request_locks within the device layer and got better performance. In the future, elevator scans should only occur once, when either inserting or scanning, not twice as happens now. Because of writeback caching on ATA, you have to do a write-back flush even if you disable the cache on the drive. Don Duggans quipped, "The placebo bit."

Tweedie agreed that each driver should maintain its own queue. Someone said that we build devices that look like 36 logical devices but are really hundreds of spindles. We would prefer that you can disable part of the elevator. Tweedie responded, "That can be done, but you will still want us to merge requests, just not order them."

Advanced File System Integration

Stephen Lord, SGI

Lord discussed some advanced features of XFS, suggesting that some of them could be moved into the kernel as part of the VFS interface. The key ideas surrounded the notion of delayed writes. Instead of scheduling a write of data immediately to disk, in delayed writes only the space for the data is reserved, and the actual write is done later. As most programs will continue to write data, delayed writes cause disk writes to become batched, and temporary files may never need to be written to disk at all.

After some comments in the Q&A about the need for Linux to scale larger, Dave Miller asked if it makes sense for anyone to use 128 CPU systems. Lord answered that SGI's customers were buying them. Miller suggested that the latency issues could be reduced by using clusters of eight CPU machines. Lord replied that clusters can fail because of variable bandwidth needed by different parts of the application. Others seemed to think that smaller scale clusters might be the future. Miller brought up NUMA (non-uniform memory access) architectures and wondered if it is necessary to scale up to so many CPUs.

The discussion shifted back to talk about delayed I/O and what happens if insufficient space is reserved for file metadata. Lord replied that SGI has been successfully doing this for years.

Illuminating the Netdriver API . . . One More Time

Jamal Hadi Salim, Znyx Networks (also Robert Olsson and Alexy Kutsnetsov)

Hadi Salim exposed a serious problem with existing Linux kernels: under extreme network loads, the system collapses. A comparison between a FreeBSD system and a Linux system showed that the BSD kernel could handle up to 70kpps (kilopackets per second) but that Linux folds at only 24kpps. Keeping in mind that a T1 filled to capacity with 64 byte packets is about 25kpps, this is not good. Furthermore, on SMP systems, performance is worse. This behavior also starves other network interfaces that are not overloaded.

When Andy Grover asked why BSD did so much better, someone replied, "They lied!"

Hadi Salim, working with Olsson and Kutsnetsov, had experimented with a solution that appears to provide a tenfold increase in performance. Their goals were to reduce interrupts on overload, drop packets early on overload, remove or reduce unfairness, and balance latency and throughput, without requiring specific hardware solutions (e.g., Tulip chip).

An obvious solution occurred to me (too bad no one asked me about this earlier). Each packet arriving at the network interface generates an interrupt, and it is this interrupt processing that soon overwhelms the kernel. Serial card designers of the early nineties had already figured this out, and the fastest drivers used polling instead of interrupts. Hadi Salim's solution is similar in that interrupts are disabled when the load increases and only re-enabled when the DMA ring assigned to the device is empty. If more packets arrive when the DMA ring is full, they are dropped without any further processing. Using 2.4.0 beta 10 they could get 200kpps peak using commodity hardware that supports DMA and polling.

Larry McVoy, who seems to have worked for all the UNIX workstation vendors at some point, remarked, "Rob Warnick at SGI did an enormous amount of good work in this area, forced because they [SGI] had to work with slow processors. If you do this stuff right, you just pass the packets right up, but if you get a blast, then you start to do interrupt collapsing." When Miller remarked that this would clean up the drivers a lot, there was a round of applause.

There was more discussion about the effect of this solution on servers (better performance), latency (less latency), and older drivers. Hadi Salim said that the system reverts to old behavior when the driver does not support polling. The session ended with everyone feeling good about this solution, which is likely to find its way into the 2.5 kernel.

Linux Hot Plug

Johannes Erdfelt, VA Linux; Greg Kroah-Hartman, WireX

Erdfelt started off by saying that they don't have a solution to this problem. His own area of expertise is USB, particularly hubs and hot plug needs to deal with SCSI, Firewire, PCMCIA/Cardbus, and hotswap PCI in addition to USB devices. The problems include name and file permissions: that is, the device name presented to the user should not change and neither should the permissions associated with that device. Since not all of these devices have anything resembling a serial number or UUID, identifying them can be a problem.

Other issues include what to do when there are multiple drivers (e.g., using a parallel port) and how to notify applications (hey, a joystick just appeared!).

Existing solutions include cardmgr for PCMCIA (which works because it is not very complicated [laughter] compared to USB, scsidev, or devfs/devfsd). Using a script, /sbin/hotplug, works but does not help with naming. At this point, Erdfelt asked for suggestions.

Someone immediately asked why devfs (think procfs) doesn't work, and Erdfelt answered that it doesn't solve the naming problem. Ts'o pointed out that in the PCMCIA world, shell scripts could handle things pretty well, but the SCSI devices would be more difficult. Linus said that /sbin/hotplug worked well for him. He thought that vendors could be relied on to provide scripts for all the hot-pluggable devices and that would solve 99% of the problem. "A script is a valid answer for a geek's machine; it can be edited, and RedHat can do it for every possible device."

Peter Anvin said that he is responsible for assigning device numbers. And right now, we are running out of character numbers. He just assigned number 228 out of 225, and suggested that the discussion (or flame war) be done offline in the 10 p.m. BoF. This did not stop the discussion, which continued until the end of the session. Anvin made an interesting point when he mentioned that the Japanese wanted Japanese device names rather than English ones.

Reception

The reception drew most of the attendees, even some people who had said earlier that they couldn't stay. I wound up talking to Peter Loscocco of the NSA about his part in a project to add real security to the Linux kernel and missed the BoFs. Well, I could have attended since they went until midnight, but I packed it in. There were also BoFs after the conference finished. You can read about the BoFs, and get another perspective on the conference, at LWN: *<http://lwn.net/2001/features/KernelSummit/>*.

Kernel kbuild

Keith Owens, OC Software; and Eric Raymond

Nine o'clock Saturday morning started off pretty quietly, no surprise there. The topic concerned a proposed replacement for the kernel configuration tools in the present distribution. I was happy to hear this, but not everyone else was. The present system presents a text-based menu that does not let you go backward, and a GUI-based one that also fails to satisfy me.

Raymond is not too fond of it either. "I've been examining the existing kernel configuration system, and I have about concluded that the best favor we could do everybody involved with it is to take it out behind the barn and shoot it through the head." Owens and Raymond set out to build a much improved system, and they did.

The new system (CML2) makes it impossible to generate an invalid configuration. The new system uses a Python engine to enforce a set of rules to do this. This prompted concern from Jes Sorenson, who is apparently porting Linux to IA64. "Can I make it work without Python?" he asked. Many people pointed out that the work could be done on the cross-compiling system and that Python was not required on the target system.

There was actually a fair amount of resistance to this effort. Raymond pointed out that the new version uses 40% less code, runs faster, and has features the old tool does not. At one point, the discussion veered into arguing about the kernel symbol table. But it ended on a good note, with Raymond saying, "Down the road, I want to make configuration so easy that your Aunt Tilly could do it." This was followed by enthusiastic applause.

Future VM Work

Rik van Riel, Conectiva S.A.

Van Riel had a detailed presentation covering a lot of difficult topics in VM, his own specialty. Occasionally he wound up listening to the discussion that raged about contentious issues, such as page size.

Van Riel began by discussing memory balancing, deciding which pages to steal and when. The current implementation scans the process page tables and steals whatever it can. Better methods would involve keeping track of working sets, providing processes with a defined amount of physical memory. Another approach would use reverse mapping, keeping track of which processes have a mapping from a virtual page to a physical page. Doing this would add eight bytes to the page table entry, doubling its size.

Once again, moving to a larger page size was suggested. Linus pointed out that early versions of his kernel did just that, but that it would make no sense to do this "just for Oracle." Van Riel explained that there would be benefits for other users, such as smaller page tables and a reduction in page faults. Large pages are

not suitable for all uses, so the ideal would be to mix large and small pages.

Shared page tables would also help in the case where processes share memory. Again, this is also something that Oracle asked for, but it would involve constraints (like having to be aligned on a 4MB boundary on some architectures). Another layer of locking would also be needed, leading to the possibility of deadlocks.

The Linux kernel still has some problems with deadlocks: for example, it needs to allocate memory to free some pages. Thrashing can also occur; Van Riel suggested suspending processes when the system begins to thrash, in order to stabilize the load, and gradually releasing each process as the system recovers.

Mandatory Access Controls / SE Linux

Peter Loscocco, NSA

I thought that this proposal was well received, although Loscocco told me later that he was disappointed that he didn't get more support. You can get more details, or the code if you like, at <*http://www.nsa.gov/selinux/*>. Loscocco also told me that in the eighties, he was responsible for the ARPANET IMP that passed through the NSA site, and that stories about how the NSA was copying all the data on the ARPANET were really funny. Eventually, the NSF moved the IMP somewhere else because of difficulties in keeping it up and running. But that's another story.

SE Linux uses a large kernel patch to add Mandatory Access Controls (MAC) to a standard Linux kernel. MAC means that permissions are no longer discretionary, and that only a security administrator can change them. The changes to the kernel are pervasive and fine-grained, meaning that it is possible to tie down everything.

One important point is that SE Linux is not an Orange Book system. The system has not been evaluated, although it has evolved from earlier work such as DTOS within MACH, and Flask with Fluke (see a paper presented at a USENIX Security Symposium). SE Linux uses Type Enforcement, which goes way beyond Orange Book designs. Type Enforcement includes the usual subject (user) and object (resource) found in earlier models but adds the program to the equation. With Type Enforcement, you can specify that a user can write to /etc/shadow but only when running the passwd program.

The design goals of SE Linux include development of:

- Separation policies to keep data separate from some other part of the system
- Containment policies to restrict Web server access to authorized data
- Integrity policies to protect applications from modification
- Invocation policies to control the chain of processing

The current implementation focuses on enforcement, not policy. Loscocco said that the sample implementation does come with some policy models that could be used to nail down a Web server, for example.

This session ended with Linus saying that he liked the code he had seen, but he was aware that there were many security projects in progress right now, and that he did not want to have to choose between them. If a function call could replace each section of code the NSA design had added, then it might be acceptable. People who did not want to use the security could replace this with a function that simply returns.

Asynchronous I/O for Linux

Ben LaHaise, RedHat Canada Ltd.

Asynchronous I/O would allow programs to write data to some device and then continue without waiting for the operation to complete. LaHaise pointed out that this would be useful in event-based applications when you want to avoid using thread-based I/O (8KB overhead per thread per each request, and there could be tens or thousands of requests). It also makes efficient use of raw I/O, fits well with zero copy, and has lower overhead for daemons than select/poll system calls.

Asynchronous I/O, as proposed by LaHaise, would add four new system calls:

_submit_ios() allows a process to fire off an asynchronous operation.
_io_cancel() is for canceling outstanding operations.
_io_getevents() gets information on completed operations.
_io_wait() allows a process to wait for the completion of a specific operation, essentially turning it synchronous.

Linus, who obviously was already familiar with the proposal, interrupted LaHaise at this point: "You have already added four system calls, why have a new device?" The device in question would be named /dev/aio and be used to arrange for a section of mapped memory that the calling process could use to detect I/O completion. Linus was not at all happy with this use of mapped memory and continued to ask if LaHaise really thought that mmap was necessary.

Toward the end of his defense of mmap, Ulrich Drepper, defender of the system call API, asked, "Why do you want to add all this support to character devices? It is completely wrong to do this." LaHaise pointed out that POSIX AIO doesn't do a very good job of the semantics, and that his approach will really make a difference with sockets, particularly UDP. Drepper asked if you can have AIO for fsync(), and LaHaise responded that he had code that could currently do that.

This project is not finished yet, and is currently about a ~3000 line patch. Work to be done included adding network sockets, limiting memory pinning, and writing some documentation.

Linux and Power Management, ACPI

Andy Grover, Intel

Grover started out by explaining the reasons why PC power management is important. He suggested some obvious things, like green PCs and mobile PCs. When he mentioned that servers might also be able to do this to reduce power when sitting idly in large clusters, Linus reacted. It was obvious this is something he wanted.

In the older scheme, currently supported by the Linux kernel, APM handles power management. The trouble with that is that APM is an obsolete interface, and that the BIOS keeps track of APM so that it is not managed by the OS. The replacement for APM is the Advanced Configuration and Power Interface (ACPI), conceived by Intel, Microsoft, and Toshiba, which allows OS-directed power management.

Grover went on to describe five soft power levels and four hard (device) levels, ranging from fully on to power off. Grover felt that implementing soft power level three would be the best target for 2.5. S3 permits idle devices to be disabled and has better power saving than levels one or two and quicker wakeup than level four (sleep). Enabling level three is a prerequisite for doing level four.

Grover explained that implementing level three means that it must be possible to shut off devices, and then to turn them all on again while restoring sufficient state for them to return to working condition. This implies having a device manager that knows enough about all the devices to shut them down or reinitialize them.

A lively discussion began at this point, with different people wondering how to make this work. Linus pointed out that the PCI device has extra, unused fields in its structure that would allow it to be extended to handle any device, and build an internal device tree. He had actually considered adding the changes in earlier, but the patches are very large, and adding large patches disturbs people. Ts'o pointed out that 2.5 might be the ideal time to do this.

The following Web sites have more info: *<http://developer.intel.com/technology/iapc/acpi>*; *<http://phobos.fachschaften.tu-muenchen.de/acpi>* (for ACPI 4 Linux)

BitKeeper

Larry McVoy, BitMover Inc.

Before he got started, McVoy suggested that we thank Ted Ts'o for making the developers summit possible. Then, he had a short rant about scaling up the Linux kernel, suggesting that SMP clusters, with one kernel monitoring every four kernels, was the way to go.

McVoy was really there to talk about BitKeeper, a source code control system. Subversion, another CVS system, had been the topic of a Friday night BoF, which McVoy could not attend. According to McVoy, as well as Victor Yodaiken, BitKeeper had the only GUI interface that would really work for kernel hackers. He went on to describe various features, as well as demonstrating them for the kernel hackers.

"BitKeeper is a peer-to-peer system. You get revision control files; we merge revision histories. We can sync sideways, rather than go up and then down, which is what we did when I worked at Sun (and I wrote their systems). You have to keep track of revisions, but we compress them."

Listening to McVoy and the reaction to his demonstration made it appear that BitKeeper really would work well in the world of Linux. There are some problems: for example, getting more checked out than you want if several modules have been modified since the last time you peered. Raymond complained when he found that the only editor bindings

Summit group photo. See the Linux Weekly News *for a color version with ids of the participants <http://lwn.net/2001/features/KernelSummit/>* [photo: J. Corbet/LWN.net]

4th Symposium on Operating Systems Design and Implementation (OSDI 2000)

OCTOBER 23–25, 2000

SAN DIEGO, CALIFORNIA, USA

KEYNOTE ADDRESS

SYSTEMS ISSUES IN GLOBAL INTERNET CONTENT DELIVERY

Daniel Lewin, Akamai Technologies, Inc.

Summarized by Darrell Anderson

Daniel Lewin started his talk with a brief introduction to the development of the Internet, emphasizing how it is really built, and how that influences what his company does. In the beginning, the Web was very simple: content providers on one side of the "Internet," users on the other. In between, network providers made sure there was a path from every user to every content provider, and vice versa. This scheme has four bottlenecks: the first mile, peering points, the network backbone, and the last mile.

The first mile includes the content provider's databases, application servers, Web servers, load balancers, switches, routers, and bandwidth. This centralized traffic creates an inherent bottleneck.

Around 7,000 networks account for up to 95% of the access traffic, and the number of those networks is growing. The edge of the Internet is becoming more diverse, with many small ISPs.

Peering is critical to how the Internet works. Peering is a bottleneck because there's no incentive to peer well — networks compete. Also, the technology for peering is inadequate: pipe size is limited, and it is difficult to peer two networks at more than five to 10 places simultaneously.

The third bottleneck is the backbone. Backbones are difficult to build and are very expensive. The business model of a backbone requires high utilization to drive down costs. Though capacity is increasing, it cannot keep up with demand. Peak utilization of the backbone is around 200Gbps (usage data from forward proxy logs), close to the "useful" capacity of the network. Demand can increase faster than capacity.

The last mile presents the final bottleneck. Once the last mile gets fast, one would expect that the whole Internet will get faster. However, if you gave everybody a cable modem, the current infrastructure of the Internet would collapse. The bottom line is, centralized delivery is slow and unreliable.

At a very high level, Akamai does "edge distribution," deploying servers inside many networks. The goal is to infuse all 7,000 networks that matter, distributing data from the content provider to these servers, serving content from as close to the end users as possible. Edge distribution bypasses most bottlenecks (first mile, peering, and backbone), improving performance, reliability, and capacity. Akamai provides network monitoring tools, and its servers are free.

Akamai estimates speeds on average are between two and 46 times faster, with an 86% reduction in download times. In addition, edge distribution improves consistency and availability.

Lewin then posed the following questions: How should a content distribution network organize servers? Storage? What data do we need to predict performance? How can we gather this data reliably and in real time?

This data enables resource management/server selection. Data gathering and server selection must be distributed and fault tolerant, and must work with imperfect information and in an unreliable setting. Also, system monitoring and management need to represent data visually to alert and allow humans to interact with the system. Object distribution and invalidation introduce problems in maintaining consistency in a massively distributed system. A content distribution system must provide access information the same way a centralized server would.

Later, Lewin described the "common point" metric, estimating the latency between end users and servers. Rather than measure actual latency between users and servers, measure latency of segments of routes, ending in a common point and enabling correlation. As a set coverage problem, common points reduce the set size from 200K to 6K, enabling feasible measurement. Pulling it together, the system computes the common point sets, gathers network data, distributes data, and performs server selection.

Akamai uses the DNS hierarchy to break up resource management and server selection into usable chunks.

Lewin concluded that the Internet is an evil place, subject to the Heisenberg uncertainty principle. It is difficult to predict popularity, distributions, and capacity. Server selection wants information quickly and accurately, and even in the face of inaccuracies, should not overload available resources. This is a hard problem.

Akamai's solution makes a few simple (and sometimes wrong) assumptions: that utilizations converge, popularities are poison, and the number of active users behind any particular DNS server is never too large. When these assumptions fail, servers or links may be overcommitted. The problem is easily parallelized because groups of users may be split.

For more information, see <*http://www.akamai.com*>.

SESSION: APPLYING LANGUAGE TECHNOLOGY TO SYSTEMS

Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions

Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem, Computer Systems Laboratory, Stanford University

Summarized by David Oppenheimer

Dawson Engler described a compiler extension system that allows compile-time checking of application-specific rules. The extensions are written in a state-machine language called *metal*, which somewhat resembles the yacc specification language, and are dynamically linked into a modified version of g++ called xg++. The metal specification describes patterns in the source language that, at compile time, cause state transitions in the state machine described by the metal specification. At the time xg++ translates a function into the compiler's internal representation, it applies the metal extensions down every code path in the function. States corresponding to rule violations signal a potential rule violation in the source program. This system is one instantiation of a general concept Engler calls *Meta-Level Compilation (MC)*, which raises compilation from the level of the programming language to the "meta level" of the code itself (e.g., interfaces, components, and system-level properties), allowing the checking, optimization, and transformation of systems at a high level much as is done for low-level code by compilers for code written in traditional programming languages.

Engler argued that MC offers benefits over traditional techniques for detecting violations of system rules. Specifically, he suggested that MC rules are significantly easier to write than formal specifications (verification), scale much better with code size (testing), and don't require difficult reasoning about the code (manual inspection). By informing the compiler of system-specific rules about the code, MC allows automated checking of many system-level properties at compile time.

Engler applied an MC system consisting of metal specifications that extend the xg++ compiler, to Linux, OpenBSD, the Stanford FLASH machine's protocol code, and the Xok exokernel. By checking rules, Engler's system found over 600 bugs in these systems. Most extensions were written in fewer than 100 lines of code and by individuals unfamiliar with the MC system itself.

During Q&A, several conference participants asked about opportunities to improve the system in areas such as reducing the number of false-positive errors reported, handling function pointers, and automatically deriving rules from code by looking at what the code "usually" does (from a static code path perspective). In response to a question about the feasibility of using the tool throughout the software development process, Engler indicated that this sounded like a potentially good idea, and he pointed out that the system's usefulness is improved when programmers structure their code to be as simple as possible, so that it is more amenable to compiler analysis.

For more information, see *<http://www.stanford.edu/~engler/>*.

Devil: An IDL for Hardware Programming

Fabrice Mérillon, Laurent Réveillère, Charles Consel, Renaud Marlet and Gilles Muller, Compose Group, IRISA/INRIA

Summarized by David Oppenheimer

Gilles Muller described Devil, an Interface Definition Language (IDL) for hardware programming. Devil attempts to ease the driver development process by defining an IDL in which a driver writer composes a strongly typed, high-level, easy-to-write description of a hardware device's software interface.

The Devil IDL is based on a few key abstractions: *ports*, which serve as a communication point and correspond to an address range; *registers*, which serve as a repository of data and are the grain of data exchange between a device and the CPU; and *variables*, which serve as the programmer interface and correspond to collections of register fragments that are given semantic values, such as bounded integers or enumerated types.

The IDL compiler checks the consistency of a Devil specification with respect to properties such as conformance to type rules, use of all declared entities, absence of multiple definition of entities, and absence of overlap of port and register descriptions. It then generates the necessary low-level driver code, which includes code to check for the proper use of the generated interface.

Muller described the Devil description of a device driver for the Logitech Busmouse and several other devices, including an IDE disk controller. Muller has found that device drivers generated using Devil are five times less prone to errors than is C code, and that Devil offers negligible performance overhead while improving programmer productivity. Muller's vision is that hardware vendors will supply device specifications as a Devil description, which can then be used to generate documentation and the device driver itself.

During Q&A, one conference attendee pointed out that embedding the hardware specification inside the compiler limits the possibility of simultaneously generating drivers for multiple platforms.

For more information, see *<http://www.irisa.fr/compose/devil/>*.

TAMING THE MEMORY HOGS: USING COMPILER-INSERTED RELEASES TO MANAGE PHYSICAL MEMORY INTELLIGENTLY

Angela Demke Brown and Todd C. Mowry, Carnegie Mellon University

Summarized by Tep Narula

Angela Demke Brown began her talk by pointing out the rapid memory consumption behavior of computational problems with large data sets, termed "out-of-core" applications. Such applications often suffer from high page-fault rates caused by the operating system's default virtual memory management policy. An earlier work by the same team had shown that, by using the compiler to analyze and insert page prefetches into the code, out-of-core applications could achieve good performance improvement on a dedicated machine. However, out-of-core tasks with aggressive prefetching tend to have a severe negative impact on other applications in a multiprogrammed environment. Brown then described the extensions they made in order to turn an out-of-core task into a "good neighbor" without placing any additional burden on the programmer.

The system consists of three parts: OS support, compiler analysis, and runtime layer. The OS support includes primitives for page prefetch and releases, as well as information on page location, usage, and availability. This was implemented as a memory management policy module and a kernel daemon called *releaser* on SGI IRIX 6.5. The compiler algorithm performs reuse analysis, locality analysis, loop splitting, and software pipelining in order to decide where to insert prefetch and release hints. The implementation is a pass in the Stanford University Intermediate Format (SUIF) compiler. The runtime layer dynamically analyzes both the static, compiler-supplied hints and the dynamic, OS-supplied system status and decides when to issue a request for either prefetch or release of a page to the OS. The runtime layer is implemented as a library that spawns a pool of pthreads to handle the prefetch and release requests within the application space. There are two release policies implemented in the library: aggressive release and priority-based buffered release. Experiments were performed using the out-of-core version of five applications taken from the NAS Parallel benchmark suite plus the MATVEC kernel. An SGI Origin 200 with four processors was used for the experiments. Overall, the results showed significant performance improvements both in dedicated and multiprogrammed scenarios.

SESSION: SCHEDULING

SURPLUS FAIR SCHEDULING: A PROPORTIONAL-SHARE CPU SCHEDULING ALGORITHM FOR SYMMETRIC MULTIPROCESSORS

Abhishek Chandra, Prashant Shenoy, and Micah Adler, University of Massachusetts, Amherst; Pawan Goyal, Ensim Corporation

Summarized by Darrell Anderson

Scheduling is important for diverse Web and multimedia applications, such as HTTP, streaming, e-commerce, and games. Applications are often hosted on large, multiprocessor servers. A key challenge is to design OS mechanisms to provide fair and proportional resource allocation. Other requirements include isolating misbehaving or overloaded applications and achieving low overheads for efficient implementation in real systems.

Proportional-share scheduling is one class of scheduling algorithms that addresses these requirements. It associates a weight with each application and allocates CPU bandwidth proportional to weight. There are a number of algorithms in use, but do they work well on multiprocessor systems?

Abhishek Chandra illustrated how one such algorithm, *Start-Time Fair Queuing* (*SFQ*), can lead to starvation when scheduling three threads on two CPUs. This starvation is a result of "infeasible weight assignment," where the accounting is different from actual allocation. A simple correction, or "weight readjustment," can be made limiting any one thread's weight to a single CPU's bandwidth, preserving overall weight ratios. Weight readjustment is efficient, running in time proportional to the number of CPUs in the system, and can be incorporated easily into existing scheduling algorithms.

A second problem arises under frequent arrivals and departures of short jobs, which Chandra calls the "short jobs problem." Again, SFQ performs correctly on uniprocessor systems but breaks down when scheduling across multiple processors.

Surplus Fair Scheduling (*SFS*) addresses this problem. With this algorithm, the scheduler measures observed processor bandwidth share, which it compares with the ideal share, computing the scheduling surplus for each thread. By scheduling threads in order of increasing surplus, the short jobs problem sees fair scheduling on single and multiprocessor systems.

Chandra presented proportional allocation tests where SFS yields near optimal allocation. Second, SFS was compared with time sharing for isolation and scheduling overhead. SFS provides superior isolation at modest additional scheduling overhead.

Q: Don't many scheduling algorithms, such as lottery scheduling, have nearly trivial extensions for multiprocessor systems?

A: Lottery scheduling has problems with proportional share when applied directly on a multiprocessor. On a multiprocessor, tickets do not translate to scheduling weight.

Q: You claim a proportional-share scheduling algorithm will not work well on multiprocessors. Does your algorithm

scale? What happens to the overhead with a very large number of processors?

A: We have developed some algorithms independent of the number of processors.

Q: How important is the virtual time in your algorithm?

A: The idea is basically heuristic.

Q: It doesn't seem that the problem is inherent in SFQ. It seems you need to have a notion of a global virtual time, rather than weight readjustment.

A: We looked at a few algorithms we could apply, but did not come up with a simple answer.

For more information, see <*http://lass.cs.umass.edu/software/gms*>.

Performance-Driven Processor Allocation

Julita Corbalán, Xavier Martorell, and Jesús Labarta, Universitat Politècnica de Catalunya

Summarized by Darrell Anderson

Performance-driven processor allocation uses runtime information to make sure that processors are always in use. The scheduling problem is how to allocate processors to applications, both for space sharing and time sharing. Things work best when the number of processes is equal to the number of processors.

Processor allocation should be proportional to application performance. A drawback of this metric is that application performance is not known before execution. One solution involves a priori calculation by measuring multiple executions, using previous results to predict later performance. Julita Corbalán proposed an alternative approach where processors are allocated to those applications that can take advantage of them. This runtime dynamic performance analysis approach, called *Performance-Driven Processor Allocation (PDPA)* requires coordination between medium- and long-term schedulers.

Corbalán used the NANOS execution environment on a shared memory multiprocessor. NANOS uses a queuing system and CPU Manager to schedule OpenMP parallel applications. The dynamic performance analysis is done by the SelfAnalyzer, a tool that estimates execution time for processes.

The SelfAnalyzer remembers baseline performance results for two and four processors, which are then used to predict performance. Performance-driven processor allocation is space sharing, allocating for acceptable efficiency. Processes run to completion with minimum allocation of one processor. Dynamic partitioning and reallocation is driven by the runtime system.

Corbalán compared PDPA against three other multiprocessor schedulers, using the OpenMP application suite on an SGI Origin 2000 with 64 processors running IRIX 6.5.8 and multiprogramming level set to 4, PDPA performs as well as, and frequently better than, the competing schedulers. Corbalán showed different applications that perform well for one scheduler, with matching PDPA performance. In some cases, PDPA performs significantly better than all three alternatives. Corbalán observed that it is very important to provide accurate performance information to the scheduler.

For more information, see <*http://www.ac.upc.es/NANOS*>.

Policies for Dynamic Clock Scheduling

Dirk Grunwald and Philip Levis, University of Colorado; Keith Farkas, Compaq Western Research Laboratory; Charles Morrey III and Michael Neufeld, University of Colorado

Summarized by Darrell Anderson

"Saving energy or power is important, both for battery life and at the microarchitecture level for clock speeds," said Michael Neufeld, as he indicated that this work is a study of proposed algorithms and is a negative result.

Instantaneous power consumption of CMOS components is proportional to the square of voltage, times frequency. Batteries will perform better if drained at a lower rate. Secondly, frequency depends on voltage. Greater voltage permits higher frequency, to a point. Lower frequencies provide more than linear reduction in power needs. It is better to run a system slowly and steadily than to run it as fast as possible and then shut the processor off. Clock scaling algorithms require load prediction and speed adjustment.

Adding to prior work, Neufeld's contribution includes a real implementation instead of simulation, focusing on practical aspects. The authors use an exponential weighting algorithm proposed in earlier work, predicting utilization from earlier intervals. Their implementation uses a 10 millisecond interval time, looking back three intervals with 50% and 70% busy as thresholds.

They modified the Compaq Itsy to measure current draw and power consumption with 5,000 samples per second. They ran the Compaq-modified Linux kernel v2.0.30, adjusted to perform clock/voltage scaling. They also used the Kaffe Java VM, instrumenting it to record and replay applications. They ran four applications, measuring energy and smoothness. Smoothness implies infrequent changes in clock speed and voltage. They ran an MPEG player, hoping that its characteristics would translate well to clock scaling. In practice, the algorithm performed only marginally better.

The negative result: weighted averaging methods do not appear to work well. What went wrong? Averaging attenuates oscillations, but does not remove them. Larger interval history might help, but would reduce responsiveness. Even if tuning were possible, it would be fragile. Also, a linear change in frequency doesn't

always mean a linear change in idle time. The authors don't have a conclusive explanation.

Neufeld points out that existing hardware has only limited voltage-scaling capabilities. A wider range of hardware would be very useful for experimentation.

SESSION: STORAGE MANAGEMENT

Towards Higher Disk-Head Utilization: Extracting "Free" Bandwidth from Busy Disk Drives

Christopher R. Lumb, Jiri Schindler, Gregory R. Ganger, David F. Nagle, Carnegie Mellon University; Erik Riedel, Hewlett-Packard Labs

Summarized by Mac Newbold

"Disk drives increasingly limit performance in many computer systems," Greg Ganger pointed out as he explained the need for better utilization of disk bandwidth. His presentation outlined a method of scheduling disk accesses that can extract an additional 20–50% of a disk's potential bandwidth without affecting the service times of the original requests. Their method describes two pools of requests: foreground requests, which include the normal and high-priority workload of the disk, and background requests, low-priority tasks that must get done but whose time of completion is less important. This "free" bandwidth is extracted by scheduling the low-priority requests between high-priority ones so that time normally spent on rotational latencies gets used for background reads and writes.

Typical disk use generally requires the disk heads to spend a relatively large amount of time seeking the desired track and waiting for the desired sectors to rotate to the disk head. These are the seek time and the rotational latency, respectively. The seek time is inevitable. However, the rotational latency can be used for other reads without affecting the time the original request would take. The authors call this process *freeblock scheduling*.

The effectiveness of freeblock scheduling relies on the ability to find background requests that fit well between the foreground requests. Tasks that are most appropriate for this are processes that are low priority, have large sets of desired blocks, require no particular order of access, and have small working memory footprints. Applications that fit these requirements include those that perform scanning, internal storage optimization, or prefetching and prewriting.

The actual results published in the paper are very promising. They demonstrate that for a process that wants to read the entire disk in the background, the full potential free bandwidth (35–40% of total potential) can be utilized until over 90% of the disk has already been scanned, and even until the entire disk has been scanned, over half of the potential free bandwidth can actually be utilized. They also show increases of disk bandwidth utilization on the order of 10 times that of the original utilization.

It remains to be seen how much of this scheduling can be done outside of modern disk drives, given the complexity of their internal algorithms and the lack of low-level interfaces. If freeblock scheduling indeed proves to be compatible with modern disk-drive technology, it could be very beneficial and could significantly increase disk bandwidth.

Latency Management in Storage Systems

Rodney Van Meter, Quantum Corporation; Minxi Gao, University of California, Berkeley

Summarized by Vijay Gupta

Rodney Van Meter indicated that the primary motivation for this work was the 11-orders-of-magnitude difference between latency of access to a tape and that of access to RAM.

To address the above problem, the authors proposed the concept of *Storage Latency Estimation Descriptors (SLEDs)*. An API, SLEDs allow applications to understand and take advantage of the dynamic state of the storage system. SLEDs are complementary to the notion of hints (which are used in transparent informed prefetching). Whereas hints are given by applications to the OS, SLEDs are given by the OS to the applications.

Van Meter motivated the use of SLEDs by giving the example of an application which makes two sequential passes over a file. Suppose there are three pages in the buffer, and the file size is five pages. If the page replacement strategy is LRU, then the second pass will have five hard faults. On the other hand, SLED reorders reads in the second pass so that there are only two hard faults. Thus reordering I/Os yields large gains.

The SLEDs were added to v2.2.12 of the Linux kernel. The authors added new ioctl options which return SLEDs data. In addition, they modified several UNIX utilities such as `find`, `wc`, `grep` and `GMC` (which is a GUI file manager) to assess the benefit of SLEDs. `wc` reorders I/O; `grep` reorders and prunes directory search trees; `find` uses their `-latency` predicate. The feature of the `-latency` predicate is that if the latency to access some portion of the file system is going to be beyond the specified latency, then that portion of the file system would be skipped.

Then Van Meter presented a large, complex example of an astronomy application (LHEASOFT) which is made up of 840,000 lines of C and Fortran. It has a 100,000-line I/O library. The application was modified to reorder the I/O operations. This reordering achieved 11–25% reduction in execution time in spite of the fact that the program was already optimized with the I/O library.

Overall, the paper raised some very important issues for heterogeneous systems, which are becoming increasingly common.

A Low-Overhead, High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References

Jong Min Kim, Jongmoo Choi, Jesung Kim, Sang Lyul Min, Yookun Cho, and Chong Sang Kim, Seoul National University; Sam H. Noh, Hong-Ik University;

Summarized by Tamara Balac

This talk focused on a new solution to an old problem. The problem is determining which pages the OS should cache in the main memory. The motivation for their solution approach was that LRU has problems for sequential and looping references. If the access is purely sequential, then caching recently used pages is wasteful. If the access is looping, then LRU cannot figure it out. To overcome this difficulty, they proposed a new concept called *Inter-Reference Gap (IRG)*. IRG for a block is the difference between the points when the block is successively accessed.

The types of references which the authors considered are: *sequential*, *looping*, and *other*. They developed a new scheme, called *unified buffer management (UBM)*, which includes *automatic detection*, *replacement*, and *allocation*, to exploit these references.

In automatic detection, they take into account "fileID, start, end, period" for references. Initially, the period is infinity. As more references are encountered, the period is adjusted. For sequential access, the period always stays at infinity. But in the case of looping references, there is a definite period. For replacement, they adopt different schemes depending on the type of reference. For sequential access, they use MRU; for looping references, they use a period-based replacement scheme; and for other references, they use LRU. The allocation scheme is a bit too mathematical to explain here, although the intuition involves using marginal gains and IRG.

To wrap up, Noh showed the results for trace-driven simulations and showed graphs for how UBM takes into account the looping references.

SESSION: SECURITY

How to Build a Trusted Database on Untrusted Storage

Umesh Maheshwari, Radek Vingralek, and William Shapiro, STAR Lab, InterTrust Technologies Corporation

Summarized by Tamara Balac

Digital rights management protects rights of the provider (i.e., database balances, contracts). Existing systems are missing trusted storage in bulk. Umesh Maheshwari presented a Trusted DataBase system (TDB) that resists accidental and malicious corruption by using Crypto Basis, which leverages persistent storage in a trusted environment.

The TDB architecture consists of three layers: a Collection Store, an Object Store, and a Chunk Store. The Chunk Store provides trusted storage for variable-sized sequences of bytes which are the unit of encryption and validations (100B–10KB). The Object Store manages a set of named objects. The Collection Store manages a set of named collections of objects.

Performance evaluation demonstrated that Crypto overhead is small compared to I/O and that TDB performs well compared to commercial packages (e.g., XDB).

End-to-End Authorization

Jon Howell, Consystant Design Technologies; David Kotz, Dartmouth College

Summarized by Mac Newbold

In a very entertaining presentation, Jon Howell explained the need for an end-to-end authorization scheme. Currently, when a local user needs to grant access to a resource to a non-local user, it creates an authentication problem, because the server doesn't know about the non-local user. Typically this is solved by creating an account locally for the non-local user by sharing passwords, or by installing a gateway that is implicitly trusted by the server, all of which we know have many weaknesses. The solution proposed by Howell is a system for delegating authority in a way that the server has a complete proof of correctness and an audit trail for the access the remote user was given.

The system is based on delegations. For instance, local user Alice wants to give remote user Bob access to some of her files. So she delegates her authority over those files to Bob. Then when Bob asks for file X in that set of files, the server is presented with a collection of statements: first the request, "*Bob wants to access file X*," then the delegation, "*Bob speaks for Alice concerning file X*," and finally the basis for delegation, "*Alice owns file X*." The server then can make a decision about allowing Bob to access X, instead of relying on one or more gateways to decide. In this scheme, gateways are required only for translation and relay of requests and proofs. Of course, at this point, the client trusts the gateway not to abuse its authority. To deal with this, a gateway authentication scheme can be used.

One advantage of a system like this is that the gateway is very simple. It only needs to carefully quote each request and only use Alice's authority for Alice's requests. It need not make access decisions. This also allows for multiple gateways to bridge the gap between client and server, and ultimately the server is the one who grants or denies access.

The implementation of the authorization scheme uses Simple Public Key Infrastructure (SPKI) and is part of the Snowflake project. The paper includes performance evaluations that reflect some additional overhead for the end-to-end authorization but points out that a large part of the added overhead is directly attributed to slow and inefficient

SPKI libraries. It is estimated that optimized SPKI libraries would perform almost as well as a Java and Java SSL implementation of HTTP authorization. Because Snowflake performs steps that very closely correspond to those performed by SSL, most of which are computationally expensive cryptographic operations, it is expected that an optimized Snowflake would perform as well as SSL plus a small overhead for the proving steps that are not performed by SSL. These results show that their technique is potentially a very valuable resource for end-to-end authentication.

Self-Securing Storage: Protecting Data in Compromised Systems

John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, Gregory R. Ganger, Carnegie Mellon University

Summarized by Tamara Balac

Intrusions are a fact of modern computing. What are system administrators to do? Diagnosis and recovery. Nevertheless, the major problem with diagnosis and recovery is that intruders can manipulate ALL stored information.

John Strunk presented an implementation of a *Self-Securing Storage System,* named S4, that prevents undetectable modifications by providing a complete history of all modifications. S4 is a separate piece of hardware that runs a separate operating system. The next step is to add internal reasoning and auditing, by creating a new version (called collections) on every write, and to store these for a guaranteed amount of time (called detection window).

The benefits of S4 include providing an opportunity to analyze security compromises, enabling speedy recovery, and allowing recovery from accidents (like accidental file modifications).

Diagnostic comparison of S4 and conventional systems showed that conventional systems use guessing, while S4's audit log shows the sequence of storage events. Additionally, an S4 administrator can recreate the state of the storage at any point of time.

Feasibility evaluation showed that large detection windows, even multi-week detection windows, are possible. More importantly, the performance overhead was less than 15%.

Fast and Secure Distributed Read-Only File System

Kevin Fu, M. Frans Kaashoek, and David Mazières, MIT Laboratory for Computer Science

Summarized by Vijay Gupta

The motivation for this talk was that Internet users increasingly rely on publicly available data for everything from software installation to investment decisions. Unfortunately, the vast majority of public content on the Internet comes with no integrity or authenticity guarantees. This work uses a secure file system (SFS) that was built by the authors and presented at SOSP 1999.

David Mazières started off by providing an example of installing an OS over the network. He also gave reasons why people avoid security: performance, scalability, reliability, and convenience. Another issue in a distributed system is that the more replicas you have, the greater the chance of a break-in. To mitigate this problem, people resort to ad hoc solutions. As an example, lots of software packages contain PGP signatures. The problem with PGP is that it is not general purpose; most users ignore signatures, and it requires the continued attention of the user.

So, they propose a *self-certifying read-only file system* , which has been designed to be widely replicated on untrusted servers. This acts as a content distribution system providing secure, scalable access to public, read-only data. With this, one can publish any data. It is convenient because you can access it from any application. It is scalable because publishing has been separated from the distribution infrastructure.

In their approach, an administrator creates a database of a file system's contents and digitally signs it offline using the file system's private key. The administrator then widely replicates the database on untrusted machines. Client machines pick their data from these machines and before data is returned to the applications, SFS checks the authenticity of data. The good thing about their approach is that no cryptographic operations are performed on servers, and the overhead of cryptography on the clients is low. This is accomplished using collision-resistant cryptographic hashes. For details about the scheme, the interested reader is referred to the paper.

The read-only file system is implemented as two daemons (sfsrocd and sfsrosd in the SFS system). A performance evaluation shows that sfsrosd can support 1,012 short-lived connections per second on a PC, which is 92 times better than a secure Web server. Finally, Mazières also mentioned the necessity of periodically re-signing data and putting them on to the server to prevent break-ins.

For more information, see <*http://www.fs.net*>.

SESSION: NETWORKING

Overcast: Reliable Multicasting with an Overlay Network

John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole Jr., Cisco Systems

Summarized by Mac Newbold

Overcast addresses many of the problems with current multicasting solutions in the Internet. One option is IP multicast, but it has several weaknesses. It can only be used for live transmission, has no delivery guarantees, requires support in routers, servers, and clients, and makes accurate billing and good security nearly impossible. Another alternative is a con-

tent distribution system, which provides on-demand content. But it doesn't perform so well with live content, often is not scalable, and doesn't allow for nodes to be added or deleted on the fly.

John Jannotti indicated that Overcast provides a unified reliable multicast solution. It is self-organized, handles live, time-delayed, and on-demand content, and any HTTP client can join without modification. As an overlay network, it is also incrementally deployable and requires no special support from the underlying network or its routers. It uses HTTP over TCP port 80 and has other features that make it compatible with NATs, firewalls, and HTTP proxies.

One key to the Overcast design is the process of building an efficient multicast distribution tree. The source of the multicast is designated as the root of the tree, and any nodes that want to join contact that root node and attach to the tree as its child. Periodically, each node considers its "sibling" nodes and "grandparent" nodes as possible new parents. If it finds that its connection to a sibling node is better than its connection to its current parent (for instance, in terms of latency, bandwidth, or hop counts), it makes that sibling node its new parent. It could also find that a connection directly to its grandparent would be a better connection than through its current parent, and it would, in effect, become a sibling to its parent node. This protocol makes the topology flexible when faced with changing network conditions.

Overcast uses only "upstream" connections when contacting other nodes; that is, the child always must establish a connection with the parent. This ensures that HTTP proxies and NATs do not interfere with Overcast functionality. For this reason, an up/down protocol was created for maintaining information about node status. It requires each node to check in with its parent periodically, and if it misses a check-in, it is considered dead. Through the use of "death certificates" and "birth certificates," it notifies the hierarchy of changes in topology. A key to scalability here is a system for quenching messages that aren't necessary for nodes further up the hierarchy. Overcast network usage for these messages scales sublinearly, and space usage on the root node is linear, but even in a group of millions of nodes, total RAM cost for the root would be under $1,000.

Jannotti referred to the paper which also outlines a system for replicating the root node to increase reliability of the root server itself. The system uses techniques used by normal redundant server setups, such as DNS redirection or round-robin load balancing and, in the case of a failed server, IP address takeover. The Overcast-specific solution is to replicate root state by setting up the servers linearly, with each replicated root being the only child of the root node above it, so that the rest of the hierarchy is descended from each of the root nodes. This way when one fails, another can immediately take over without any interruption of service or loss of state.

Jannotti concluded that the reliable multicast solution proposed in Overcast is a very feasible solution in terms of deployability, scalability, efficiency, flexibility, and usability in real-world situations.

System Support for Bandwidth Management and Content Adaptation in Internet Applications

David Andersen, Deepak Bansal, Dorothy Curtis, and Hari Balakrishnan, MIT Laboratory for Computer Science; Srinivasan Seshan, Carnegie Mellon University

Summarized by Vijay Gupta

David Andersen opened his talk by saying that the primary motivation of the work was to facilitate end-to-end congestion control. TCP uses an additive increase, multiplicative decrease (AIMD) scheme for congestion control. That's wonderful for FTP and email, which use just one TCP connection. But HTTP uses four connections in parallel between the same two endpoints in the Internet. Furthermore, not all applications necessarily need the reliability of TCP, so such applications use UDP. To make them TCP-friendly, they end up using some home-grown congestion control. The goal of this work is to have some kernel-level mechanisms to facilitate TCP friendliness.

Andersen showed where the congestion controller occurs in the Linux kernel, where they implemented their scheme. They use callbacks for orchestrating transmissions and application notification. The clients for these callbacks are in-kernel TCP clients.

The implementation handles flow control and congestion control. It also has a user-level rate callback API, called libcm, which tells if bandwidth goes up by some factor (e.g., a factor of 2).

Andersen addressed evaluation issues such as the impact on the network and on the connections. Their approach has a positive effect on the TCP friendliness of host-to-host flows, although the throughput of the connections may be slightly worse.

For testing the flow integration, they used a series of Web-like requests on the Utah testbed (see <*http://www.cs.utah.edu/flux/testbed/*>). Andersen also presented graphs to show that congestion manager (CM) is efficient, and he then showed some results for layered MPEG-4 (which is not in the paper). He also said that implementing an adaptive visual audio toolkit (vat) was very trivial using the CM toolkit.

For more information, see <*http://nms.lcs.mit.edu/projects/cm/*>.

SESSION: STORAGE DEVICES

OPERATING SYSTEM MANAGEMENT OF MEMS-BASED STORAGE DEVICES

John Linwood Griffin, Steven W. Schlosser, Gregory R. Ganger, and David F. Nagle, Carnegie Mellon University

Summarized by Vijay Gupta

John Griffin began the talk with an overview of *microelectromechanicals* (*MEMS*) and MEMS-based storage devices. MEMS is a new storage technology currently under development in industry and academia. Real-life systems such as car airbags already use them. The results in this paper are based on collaboration with the MEMS lab at CMU.

The real advantage of MEMS-based storage devices is that the seek time is an order of magnitude less than with disks. The purpose of the talk was to show that work on disk-scheduling algorithms is also applicable to MEMS-based storage devices. This was shown through a variety of graphs which had the same shape for disk-scheduling and MEMS-based storage.

An interesting aspect of MEMS-based storage is that the access is faster for data in the center, and slower for data at the borders. So the authors suggest that the center be used for metadata and small objects, and that the border be used for large-streaming media objects.

During Q&A, one person asked about price/GB, and Griffin said that it might be cheaper than hard disk. Someone else asked when we could see them in real systems, and he replied that it could be expected in a few years. For power requirements, he referred to their upcoming ASPLOS paper, while for MTTF, Griffin responded that he did not have an answer.

For more information, see <*http://lcs.Web.cmu.edu/research/MEMS*>.

TRADING CAPACITY FOR PERFORMANCE IN A DISK ARRAY

Xiang Yu, Benjamin Gum, Yuqun Chen, Randolph Y. Wang, Kai Li, Princeton University; Arvind Krishnamurthy, Yale University, Thomas E. Anderson, University of Washington

Summarized by Vijay Gupta

Large disks are now available for a fraction of what they used to cost, and hence, one can be more creative about how to use them. In this paper, Randolph Wang proposes a way to reduce the usage level of disks.

Since the access times between memory and disks keeps on increasing, RAID systems were made to improve the read/write throughput of the systems and to improve reliability. However, this work goes beyond RAID by contributing an SR-array, which flexibly combines striping with rotational replication to reduce both seek and rotational delay. The average seek distance becomes less than one-third of the ratio of the maximum to average rotational delay, and the rotational delay for reads is reduced to half.

Wang went on to show some of the equations which they derived as part of their theoretical model. They have a prototype MimdRAID implementation (which is a simulator) that puts the theory to test. The bottom line was that the MimdRAID prototype can deliver latency and throughput results unmatched by conventional approaches.

For more information, see

<*http://www.cs.princeton.edu/~rywang/mimdraid*>.

INTERPOSED REQUEST ROUTING FOR SCALABLE NETWORK STORAGE

Darrell C. Anderson, Jeffrey S. Chase, Amin M. Vahdat, Duke University

Summarized by Mac Newbold

Jeff Chase described a new storage system architecture called Slice. It takes advantage of high-speed networks to interpose a request switching filter, a microproxy or μproxy, and presents an NFS interface to clients that has a back end which scales well in both bandwidth and capacity.

Because of recent advances in LAN performance, a specialized Storage Area Network (SAN) with faster network connections (like Fibre Channel) is no longer required, and similar approaches can be used in a LAN environment to provide scalable network storage. The Slice file service is a group of servers that cooperate to provide an arbitrarily large "virtual volume" to a client, who sees it as a single file server. Client requests are separated into three classes: high-volume I/O to large files, I/O on small files, and operations on namespace or file attributes. This diverts high-volume data flow around manager nodes and allows specialization of the servers for each type of data.

The μproxy handles all bulk I/O requests and was designed to be small, simple, and fast. It has been implemented as a loadable packet filter module for FreeBSD. It may rewrite source or destination addresses or other fields in request and response packets. It maintains a bounded amount of soft state that is not shared across clients, so it can easily be placed on the client itself, in a network interface, or in a network element close to the storage servers. All of its functions can be replicated freely to provide scalability, with the constraint that requests for a given client all pass through the same μproxy. The μproxy routes requests directly to the storage array without any further intervention by management nodes.

The storage nodes themselves use an object-based method as opposed to sector-based. This feature lets the μproxy be located outside of the server's trust boundary and use encryption to protect object identifiers, limiting damage from a compromised μproxy to those files and directories that its clients have permission to access. Slice is also compatible

with sector-based storage if every μproxy is trusted. Redundancy can also be provided at two levels, either internally to each storage node or across nodes through mirroring and striping. It can be configured on a per-file basis, and a Slice configuration could even use redundancy at both levels for stronger protection.

The two types of management nodes, the directory servers and the small file servers, take load off of the storage nodes and allow for further specialization for these types of operations. These managers are data-less, and all their state comes from the storage arrays, so they provide only memory and CPU resources to cache and manipulate the structures. The directory server handles all lookups, creation, renaming, and deletion of directories and files and their attributes. The small file servers provide more efficient space allocation and file growth, as well as batching of multiple small requests into larger ones for efficient disk writes. The managers also help provide atomicity and recovery features through a write-ahead log and two-phase commits.

Performance data indicates that Slice does indeed scale very well. Name-intensive benchmarks showed directory service can be improved simply by adding more directory server sites, and for any given configuration, the performance scales linearly with the number of clients. Performance was evaluated with the industry-standard SPECsfs97 benchmark, and Slice kept up perfectly with the offered load up to its saturation point, which can be easily raised by adding more storage nodes.

Slice definitely provides a network storage solution that is scalable, reliable, practical, easy to upgrade, and comparable to similar commercial solutions in terms of performance. With benefits like these, we are sure to hear more about Slice in the near future.

For more information, see <*http://www.cs.duke.edu/ari/slice/*>.

SESSION: RELIABILITY

Proactive Recovery in a Byzantine-Fault-Tolerant System

Miguel Castro and Barbara Liskov, MIT Laboratory for Computer Science

Summarized by Tamara Balac

Today's computer systems provide crucial information and services which make them more vulnerable to malicious attacks and make the consequences of these attacks, as well as software bugs, more serious. As an alternative to the usual technique of rebooting the system, this paper proposes a new means of system recovery that does not use public key cryptography.

Miguel Castro presented an asynchronous state-machine replication system that offers both integrity and high availability and is able to tolerate Byzantine faults which can be caused by malicious attacks or software errors. The paper presents a number of new techniques, like proactive recovery of replicas, fresh messages, and efficient state transfer, needed to provide good recovery service.

The task of recovery from Byzantine faults is made harder by the fact that the recovery protocol itself needs to tolerate other Byzantine-faulty replicas. Attackers must be prevented from impersonating recovered replicas. The advantage of this system over previous state-machine replication algorithms is the use of symmetric cryptography for authentication of all protocol messages, which bypasses the major public key cryptography bottleneck.

This algorithm has been implemented as a simple interface, generic program library that can be used to provide Byzantine-fault-tolerant versions of different services.

For more information, see <*http://www.pmg.lcs.mit.edu*>.

EXPLORING FAILURE TRANSPARENCY AND THE LIMITS OF GENERIC RECOVERY

David E. Lowell, Western Research Laboratory, Compaq Computer Corporation; Subhachandra Chandra and Peter M. Chen, University of Michigan

Summarized by David Oppenheimer

David Lowell described the abstraction of "failure transparency," in which an operating system provides the illusion of failure-free operation by automatically recovering applications after hardware, operating system, or application failures without explicit programmer assistance. His work proposes two invariants that must be upheld to achieve failure transparency. The "save-work invariant" specifies what application state must be preserved to mask the failures. The "lose-work invariant" specifies what application state must be discarded to allow recovery from failures that affect the application state.

Lowell's theory defines a space of recovery protocols. Each point in the space represents a different technique for upholding save-work. One axis of this space is the effort made to commit only visible events, while the other is the effort made to identify and convert non-deterministic events. Lowell mapped a number of protocols onto this space and discussed how performance, simplicity, reliability, recovery time, and other design variables vary over the space.

Lowell presented performance results, using "save-work invariant," for four applications: nvi, magic, xpilot, and TreadMarks. He ran each application on a number of protocols from the protocol space. For these applications he found an overhead of 0–12% when his recovery system used reliable memory as stable storage. He found overhead of 13–40% for the interactive workloads when using disk as stable storage.

In addition to nvi, Lowell used postgres for measuring performance with "lose-work invariant." By injecting faults into nvi and postgres, Lowell also measured the fraction of application faults that violate lose-work by committing after the fault is activated. He found that upholding save-work for these applications caused them to violate lose-work in at least 35% of application crashes from non-deterministic faults. Merging this result with published fault distributions, he estimated that perhaps greater than 90% of application crashes in the field violate lose-work, making application generic recovery impossible in those cases.

Because operating system faults often do not manifest as propagation failures, OS faults cause violation of lose-work less frequently than application faults. Lowell presented fault-injection study results in which nvi and postgres violated lose-work in 15% and 3% of OS crashes, respectively.

Lowell concluded that for stop failures, which do not require upholding lose-work since by definition they crash the application before corrupting any application state, low-overhead failure transparency is possible for many real applications. Recovering from propagation failures is much more difficult because upholding save-work often forces violation of lose-work. From this study, Lowell concluded that providing failure transparency for stop failures is feasible, but that recovery from propagation failures cannot be accomplished transparently and must involve help from the application.

During Q&A, conference attendees asked about the feasibility of rebooting applications as a mechanism for stopping a program before an error has propagated, and the difference in the chance of violating lose-work for hardware failures as opposed to software failures.

For more information, see *<http://www.eecs.umich.edu/Rio>*.

DESIGN AND EVALUATION OF A CONTINUOUS CONSISTENCY MODEL FOR REPLICATED SERVICES

Haifeng Yu and Amin Vahdat, Duke University

Summarized by David Oppenheimer

Amin Vahdat described the design and evaluation of TACT, a continuous consistency model for replicated services. This model proposes a continuous range of consistency options for distributed applications, rather than simply the traditional strong and optimistic concurrency policies. By proposing a set of metrics to quantify the consistency spectrum — numerical error, order error, and staleness — TACT allows the investigation of tradeoffs among consistency, availability, and performance as consistency policies are varied in the space between strong and optimistic concurrency.

Applications that use TACT specify their desired consistency semantics using *conits*. A conit is a three-dimensional vector associated with each application-specific physical or logical unit of consistency (e.g., a block of seats on a flight in a distributed airline reservation system). The three elements of a conit are numerical error, which bounds the discrepancy between the local value of a piece of data and the value in the "final image" of the data; order error, which bounds the difference in the order in which updates are applied to a local replica and the ordering of those updates in the "final image"; and staleness, which specifies a maximum amount of time before a non-local write is accepted to be applied locally. Bayou-style anti-entropy is used as the mechanism for maintaining consistency among replicas.

TACT is implemented as a middleware layer that enforces the consistency bounds specified by the application's conits. It allows applications to dynamically trade consistency for performance based on service, network, and request characteristics. Three systems have been

built using the TACT platform: a bulletin board, an airline reservation system, and a system for enforcing quality of service (QoS) guarantees among distributed Web servers. For these systems Vahdat evaluated such issues as latency for posting a bulletin board message as a function of the numerical error bound; reservation conflict rate, throughput, and reservation latency as a function of inconsistency in the airline reservation system; and number of consistency messages as a function of relative error for the distributed Web server QoS system.

During Q&A, Vahdat indicated that his group is currently investigating issues such as how responsive the system is to rapid variation in desired consistency levels and how much effort is required by a programmer to incorporate conits into an application.

For more information, see <*http://www.cs.duke.edu/ari/issg/TACT/*>.

SESSION: SYSTEM ARCHITECTURE

Scalable Distributed Data Structures for Internet Service Construction

Steven D. Gribble, Eric A. Brewer, Joseph M. Hellerstein, and David Culler, University of California, Berkeley

Summarized by Mac Newbold

Building and running a cluster-based Internet service is hard. Steven Gribble explained the implementation of a Distributed Data Structure (DDS) that is designed to be a reusable storage layer for Internet services. The goals of the DDS are to be scalable, highly available and reliable in the face of failures, to maintain strict consistency of distributed data, and to provide operational manageability. In particular, they have designed and implemented a distributed hash table that meets these goals.

The basic design starts with a cluster. This provides low latency, redundancy, and makes a two-phase or multiple round-trip system feasible. All instances of the service see the same data structure, so any client can work with any server for any transaction, which simplifies load balancing and request routing.

This design provides incremental scalability as nodes are added to the cluster and was tested up to terabytes of storage space. Because individual nodes and disks might fail, each partition of data is replicated on multiple nodes, forming a replica group, which are all kept strictly coherent. Any replica can service a data lookup, but any state changes must happen in all replicas.

A simple algorithm is also in place to provide fault tolerance. If a replica crashes, it is simply removed from the replica group and operation continues. When a node joins or rejoins a replica group, the partitions that it will duplicate are copied from existing replicas, and the node is added to the replica group. Individual partitions are kept small (approximately 100MB), so that an entire data partition can be copied in 1 to 10 seconds (given a 100Mbps to 1Gbps network). The partition to be copied is locked by the joining node, copied, and the replica group maps are updated, and the locks are released. Any write operations on that partition will have failed during that time, but after the lock is released, retries will succeed.

Performance data shows that the maximum throughput of the DDS scales linearly with the number of bricks. Read operations also scale linearly up to the saturation point, where read throughput plateaus, but again, by adding bricks, the throughput is increased. Write operations run into problems, however, because garbage collection ends up causing an imbalance between the nodes in a replica group, and because the writes have to happen on all replicas, the slow one becomes the bottleneck. Even performance in recovery was very promising; an N-brick DDS during a single failure and recovery appeared to yield performance near to that of an (N-1)-brick DDS, except that the partitions on the failed brick were available only for reading.

An example of the usefulness of DDS for rapid construction of Internet Services is Sanctio, an instant messaging gateway. It translates between ICQ, AOL's AIM protocol, email, and voice messaging over cellular phones. It also uses AltaVista's BabelFish to do language translation. During his presentation, Gribble told of using Sanctio to translate his English ICQ connection to an Italian AIM connection to communicate with a friend's grandmother in Italy. Using DDS for its storage, Sanctio was completed in less than one person month, and the code that interacts with the DDS took less than a day to develop.

This work shows that a Distributed Data Structure can provide a scalable, highly available, reliable, consistent, and easy-to-use interface to network storage, and shows its usefulness for constructing Internet services.

Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java

Godmar Back, Wilson H. Hsieh, and Jay Lepreau, University of Utah

Summarized by Tamara Balac

Godmar Back described the design and implementation of KaffeOS, a Java virtual machine that supports the operating system abstraction of a process and provides the ability to isolate applications from each other or to limit their resource consumption and still share objects directly. Processes enable several important features. First, the resource demands for Java processes can be accounted for separately, including memory consumption and GC time. Second, Java processes can be terminated, if their resource demands are too high, without damaging the system. Third, termination reclaims the resources of the terminated Java process.

Each process executes as if it were to run as its own virtual machine, including separate garbage collection of it's own heap. These features make the KaffeOS capable of running untrusted code safely, since it can prevent simple DoS attacks and handle misbehaving code. But this feature comes with the price of KaffeOS being 11% slower than the freely available JVM, on which it is based. Back claims that this is a reasonable cost for the safety that it provides. Even though KaffeOS was substantially slower, it showed much better performance than commercially available JVMs scaling in the presence of uncooperative code.

For more information, see <*http://www.kaffe.org*> and <*http://www.cs.utah.edu/flux/*>.

Knit: Component Composition for Systems Software

Alastair Reid, Matthew Flatt, Leigh Stoller, Jay Lepreau, and Eric Eide, University of Utah

Summarized by David Oppenheimer

Alastair Reid described *Knit*, a component definition and linking language for systems code. The goal of the Knit project is to make components practical for systems programming. Knit defines a static configuration language for components that makes it easier to deal with complex component interdependencies. The Knit component model is based on the concepts of *atomic units* and *compound units*. Atomic units are composed of *imports* (names of functions and variables that will be supplied to the unit by another unit); *exports* (names of functions and variables defined by the unit for use by other units); and C declarations of each exported name (which are usually specified by a list of files). Atomic units are linked together to form compound units. Compound units specify how imports and exports of atomic units are linked to one another. Each component may also describe its initialization requirements.

Knit provides several benefits to developers of component-based systems: (1) it can be used with existing C and assembly language code with little modification to that code; (2) it automatically schedules component initialization and finalization, even when components have mutual dependencies; (3) it checks programmer-declared domain-specific architectural invariants; and (4) it inlines functions across component boundaries.

Reid described how his group has modified the Utah OSKit and a subset of the Click router software to use Knit. Knit-based OSKit programs had the same performance (2% slower to 3% faster) as equivalent OSKit programs built using traditional tools. Using Knit's cross-module inlining, an optimization that allows the use of very small components without incurring a significant performance penalty, the group was able to speed up a subset of Click by 35%.

During Q&A, conference attendees asked about the memory overhead of cross-module inlining, about providing protection between components, and about describing constraints among components obtained from sources that do not know about one another or about the final composition of the components.

For more information, see <*http://www.cs.utah.edu/flux/alchemy/*>.

needles in the craystack: when machines get sick

Part 4: Entropy: The Good, the Bad, and the Aged

by Mark Burgess

Mark is an associate professor at Oslo College and is the program chair for LISA 2001.

<Mark.Burgess@iu.hio.no>

DEDICATED TO THE MEMORY OF CLAUDE SHANNON (1916-2001)

You and I, and everyone on the planet, are doomed to die because of a memory leak in the human genome. For better or for worse, whether bug or a feature, DNA contains a sequence of repeated molecular material called telomeres which is used in the unzipping and replication of the DNA strand. Each time DNA is replicated, one of these telomeres is used up and does not get transferred to the copy. Finally, after 50 or so forks, all the telomeres have been used up, and the cell replication program crashes. It is a classic case of unlimited use of limited resources.

Enzyme telomerase is a garbage collection agent which returns this resource to the resource pool. In the growing fetus, it is hard at work, rejuvenating the stem cells which provide the clay for the developing body. But in time it ceases to be produced and the telometer starts clocking up our fare.

Runaway resource usage is nothing to write home about. It is happening all around us. Until the recent trend toward recycling made a small dent in a huge problem, most of the Earth's resources were harvested and disposed of in such a way that they were unrecoverable. We think little about waste. We consume and we abandon. What we abandon is even toxic to the system: fuel emissions, for example, produce poisonous gases, upset the greenhouse balance and even the protective ozone layer. Klondike Pete with his trusty mule searched the hills for years to dig up a few grams of gold, only for future generations to spread it thinly over electronic devices, which are now being buried under piles of dirt, when we are bored with them so that the gold can never be recovered. Burying nuclear waste might frighten some, but burying precious resources is a more certain threat to our future.

With computers we see the same phenomenon not only in the disposal of hardware, the circuitry, and the cases, but also with the resources of the software: disk space is used wastefully (lack of tidying, growing log files), memory leaks in buggy software (never frees RAM or disk), creeping featurism in software (placing ever greater demands on resources). DOS attacks and spam take advantage of the limitation of finite resources and show us the folly of presumption. The idea that we should reduce our consumption of precious resources is not a popular paradigm in contemporary Western society, but it will be a necessary one.

Environmentally conscious observers have long pointed out the negative effects of resource abuse on the environment, but it is less common to point out the steady decline of our resource pool. It is not just fossil fuels, but metals, forests, and biodiversity itself which are at risk. This familiar problem has always existed and will always exist, because resources are inevitably finite.

Availability of resources has been discussed in many contexts, but all examples of resource depletion are essentially symptomatic of a fundamental phenomenon: the build-up of useless information, of waste. In the safe abstract world of physics, this phenomenon acquired the name of entropy. The concept was originally used in the study of

ideal gases, but it was later extended to many other phenomena, as general principles were understood. It was many years before the full meaning of entropy was revealed. Many of us have acquired a mythological understanding of entropy, through accounts of popular physics, as being the expression of what we all know in our guts: that everything eventually goes to hell. Disorder increases. Things break down. We grow old and fall apart.

Entropy: Information's Lost+Found

Although there is nothing wrong with the essence of this mythological entropy, it is imprecise and doesn't help us to understand why resource consumption has inevitable consequences, nor what it might have to do with computers. Entropy is a useful measure, and its increase is an important principle, so understanding it is key to all science. What makes entropy a poorly understood concept is its subtlety. The exuberant mathematician John Von Neumann is reputed to have told Claude Shannon, the founder of information theory, that he should call his quantity H *informational entropy*, because it would give him a great advantage at conferences where no one really understood what entropy was anyway.

Entropy is a useful measure, and its increase is an important principle, so understanding it is key to all science.

Before statistical methods were introduced by Boltzmann and others, entropy was defined to be *the amount of energy in a system which is unavailable for conversion into useful work*, i.e., the amount of resources which are already reduced to waste. This had a clear meaning to the designers of steam engines and cooling towers but did not seem to have much meaning to mathematicians. Physicists like Boltzmann and Brillouin, and later Shannon, made the idea of entropy gradually more precise so that, today, entropy has a precise definition, based on the idea of *digitization* – discussed in the last part of this series. The definition turns out to encompass the old physical idea of entropy as unusable resources while providing a microscopic picture of its meaning.

Think of a digitized signal over time. At each time interval, the signal is sampled, and the value measured is one of a number of classes or digits C, so that a changing signal is classified into a sequence of digits. Over time, we can count the occurrences of each digit. If the number of each type i is n_i, and the total number is N, we can speak of the probability of measuring each type of digit since measurement started. It is just the fraction of all the digits in each class $p_i=n_i/N$. Shannon then showed that the entropy could be defined by

$$H = - p_1 \log_2 p_1 - p_2 \log_2 p_2 \ldots - p_C \log_2 p_C$$

where the base 2 logarithm is used so that the measurement turns out to be measured in "bits." This quantity has many deep and useful properties that we don't have to go into here. Shannon showed that the quantity H was a measure of the amount of information in the original signal, in the sense that it measured the amount of its variation. He also showed that it is a lower limit on the length of an equivalent message (in bits) to which a signal can be compressed.

The scale of entropy tells us about the distribution of numbers of digits. It has a minimum value of zero if all of the p_i are zero except for one, i.e., if all of the signal lies in the same class, or is composed of the same single digit for all time, e.g., "AAAAAAAAA...." This corresponds to a minimum amount of information or a maximum amount of order in the signal. Entropy has a maximum value if all of the p_i are the same. This means that the digitized signal wanders over all possible classes evenly and thus contains the maximum amount of variation, or information, i.e., it is a very disordered signal such as "QWERTYUIOPASD...."

Noise is indeed information: in fact, it is very rich information.

The entropy is just a number: it does not "remember" the sequence of events which led to the value describing state of the system, because it involves an implicit averaging over time (we cannot recover a sequence of changes from a single number). But it does record how much average information was present in the sequence since measurements began.

As a metaphor, entropy is discussed with three distinct interpretations: gradual degradation (the ugly), total information content (the good), and loss of certainty (the bad). Let's consider these in turn.

Ugly: this interpretation is based on the assumption that there are many random changes in a system (due to unexpected external factors, like users, for instance), which cause the measured signal (or state of the system) to gradually wander randomly over all possible states. Entropy will tend to increase due to random influence from the environment (noise). This corresponds to a gradual degradation from its state of order at the initial time. This interpretation comes from physics and is perhaps more appropriate in physics than in computer science, because nature has more hidden complexity than do computers; still, it carries some truth because users introduce a lot of randomness into computer systems. Thus, entropy is decay or disorder.

Good: information can only be coded into a signal by variation, so the greater the variation, the greater the amount of information which it could contain. Information is a good thing, some would say, but this view does not have any prejudice about what kind of information is being coded. Thus entropy is information.

Bad: if there is a lot of information, distributed evenly over every possibility, then it is hard to find any meaning in the data. Thus entropy is uncertainty, because uncertainty is conflicting information.

It is not hard to see that these viewpoints all amount to the same thing. It is simply a matter of interpretation: whether we consider information to be good or bad, wanted or unwanted; change is information, and information can only be represented by a pattern of change. Our prejudicial values might tend to favor an interpretation where a noisy radio transmission contains less information than a clear signal, but that is not objectively true.

Noise is indeed information: in fact, it is very rich information. It contains information about all the unrelated things which are happening in the environment. It is not desired information, but it is information. Much of the confusion about entropy comes from confusing information with *meaning*. We cannot derive meaning from noise, because it contains too much information without a context to decipher it. Meaning is found by restricting and isolating information strings and attaching significance to them, with context. It is about looking for order in chaos, i.e., a subset of the total information.

In fact, at the deepest level, the meaning of entropy or information is simple: when you put the same labels on a whole bunch of objects, you can't tell the difference between them anymore. That means you can shuffle them however you like, and you won't be able to reverse the process.

Entropy grows when distinctions lose their meaning and the system spreads into every possible configuration. Entropy is reduced when only one of many possibilities is prevalent. What does this have to do with forgetfulness and wastefulness? There are two principles at work.

The first, *grouping by digitization* (ignoring detail), involves reducing the number of classes or distinctions into fewer, larger groups called digits. By assimilating individual classes into collective groups, the number of types of digits is fewer, but the numbers of each type increase and thus the entropy is smaller. This, of course, is the reason for our current obsession with the digital: digital signals are more stable than continuous signals, because the difference between 1 and 0 is coarse and robust, whereas continuous (analog) signals are sensitive to every little variation. The second principle is about *repeated occurrences* of the same type of digit. One digit in the same class is as good as the next, and this means that repeated occurrences yield no more information than can be gleaned from counting. Similarity and distinction can be judged by many criteria and this is where the subtlety arises. When we measure basic resources in terms of abstract definitions (car, computer, sector, variable, etc.) there are often a number of overlapping alternative interpretations, which means the entropy of one abstract classification differs from that of a different abstract classification.

Memory fragmentation of resources can be discussed in terms of entropy.

Consider an example: fragmentation of memory resources can be discussed in terms of entropy. In memory management, one region of memory is as good as the next and thus it can be used and reused freely. The random way in which allocation and de-allocation of memory occurs leads to a fragmented array of usage. As memory is freed, holes appear amidst blocks of used memory; these are available for reuse, provided there are enough of them for the task at hand. What tends to happen, however, is that memory is allocated and de-allocated in random amounts, which leaves patches of random sizes, some of which are too small to be reused. Eventually, there is so much randomization of gap size and usage that the memory is of little use. This is an increase of entropy.

Several small patches are not the same as one large patch. There is *fragmentation* of memory, or wasted resources. One solution is to defragment, or shunt all of the allocated memory together, to close the gaps, leaving one large contiguous pool of resources. This is expensive to do all the time. Another solution is quantization of the resources into fixed-size containers. By making memory blocks of a fixed size (e.g., pages or sectors), recycling old resources is made easier. If every unit of information is allocated in fixed amounts of the same size, then any new unit will slot nicely into an old hole, and nothing will go to waste.

This is essentially the principle of car parks (aka, parking lots in the US). Imagine a makeshift car park in a yard. As new cars come and park, they park at random, leaving random gaps with a distribution of sizes (non-zero entropy). New cars may or may not fit into these gaps. There is disorder and this ends up in wastefulness. The lack of discipline soon means that the random gaps are all mixed up in a form which means that they cannot be put to useful work. The solution to this problem is to buy a can of paint and mark out digital parking spaces. This means that the use of space is now standardized: all the spaces are placed into one size/space category (zero entropy). If one car leaves, another will fit into its space.

The reason for dividing memory up into pages and disks into sectors is precisely to lower the entropy; by placing all the spaces into the same class, one has zero entropy and less wastage. C's union construction seems like an oddball data type, until one understands fragmentation; then, it is clear that it was intended for the purpose of making standard containers.

Standardization of resource transactions is a feature which allows for an efficient use of memory, but the downside of this is that it results in increased difficulty of location. Since the containers all look the same, we have to go and open every one to see what is

The accumulated entropy of a change is a measure of the amount of work which would be needed to remember how the change was made.

inside. In order to keep track of the data stored, different labels have to be coded into them which distinguish them from one another. If these labels are ordered in a simple structure, this is easy. But if they are spread at random, the search time required to recover those resources begins to increase. This is also entropy at work, but now entropy of the physical distribution of data, rather than the size distribution. These problems haunt everyone who designs computer storage.

The accumulated entropy of a change is a measure of the amount of work which would be needed to remember how the change was made. It is therefore also that amount of information which is required to *undo* a change. In the car parking example, it was a measure of the amount of resources which were lost because of disorder. In sector fragmentation it is related to the average seek time. Entropy of the universe is a measure of the amount of energy which is evenly spread and therefore cannot be used to do work. We begin to see a pattern of principle: inefficient resource usage due to the variability of change with respect to our own classification scheme. Entropy reflects these qualities and is often used as a compact way of describing them. It is not essential, but it is precise and lends a unifying idea to the notion of order and disorder.

Rendezvous with Ram

In classifying data we coarse grain, or reduce resolution. This means actively forgetting, or discarding the details. Is this forgetfulness deadly or life-giving?

If we recycle the old, we use less resources but are prevented from undoing changes and going back. The earlier state of order is lost forever to change, by erasure. We could choose to remember every change, accumulate every bit of data, keep the packaging from everything we buy, keep all of the garbage, in which case we drown in our own waste. This is not a sustainable option, but it is the price of posterity.

Forgetting the past is an important principle. In a state of equilibrium, the past is unimportant. As long as things are chugging along the same with no prospect of change, it doesn't matter how that condition arose. Even in periods of change, the distant past is less important than the recent past. Imagine how insane we would be if we were unable to forget. One theory of dreaming is based on the idea that dreams are used for short-term memory erasure, collating and integrating with long-term experience.

In *Star Trek: The Next Generation* it was suggested that the android Data never forgets. That being the case, one might ask how come he hasn't exploded already? Where do all those memories go? If elephants never forget, no wonder they are so big! Taxation has long been a way of opposing the accumulation of material wealth or potential resources (money). Throughout the centuries, all manners of scheme have been introduced in order to measure that wealth: hearth (fireplace) tax, window tax, poll tax, income tax, road toll, and entrance fees. Since income tax was introduced, records in the UK have been kept on all citizens' activities for years at a time, although there is an uncanny feeling that tax inspectors might pursue them to the grave for lost revenue, replacing the accumulation of wealth with an accumulation of records. In fact, after 12 years, tax records are destroyed in the UK. A sliding-window sampling model, rather than a cumulative model is the essence of recycling.

Queuing and Entropy

Memory is about recording patterns in space, but entropy of spatial classification is not the only way that resources get used up. Another way is through entropy of time resources. Everyone is familiar with the problem of scheduling time to different tasks. Interruption is the system administrator's lot. As one reader commented to me, his

company often insists: drop what you are doing and do *this* instead! It results in fragmentation of process: only a small piece of each task gets done. In computer systems it is algorithmic complexity which is responsible for sharing time amongst different tasks. Context switching is the algorithm multi-tasking computers use for sharing time strictly between different tasks. This sharing of time implies some kind of queuing with all its attendant problems: starvation and priorities. Context switching places tasks in a round-robin queue, in which the system goes through each task and spends a little time on it, by assigning it to a CPU. This is an efficient way of getting jobs done, because the number of objects or classes is approximately constant, and thus the parking lot principle applies to the time fragments. It does not work if the number of jobs grows out of control. But if one simply piles new jobs on top of one another, then none of the jobs will get finished. Anyone who has played strategy games like Risk knows that it does not pay to spread one's army of resources too thinly.

This is much the same problem that is considered in traffic analysis (cars and network packets). At a junction, cars or packets are arriving at a certain rate. The junction allows a certain number to flow through from each adjoining route, but if the junction capacity is too slow, then the entropy of the total resources grows to infinity because the number of different digits (cars or packets) is growing. No algorithm can solve this problem, because it has to focus on smaller and smaller parts of the whole. This is the essence of the expression *a watched kettle never boils* taken to extremes. Spamming or denial of service attacks succeed because resources are used up without replacement. This leads to "starvation" of time resources and/or memory resources.

It was once suggested to me that cars should not have to drive more slowly on the motorway when lanes are closed for repair: according to hydrodynamics, everyone should drive much faster when they pass through the constricted channel, to keep up the flow. Unfortunately, unlike ideal fluids, cars do not have purely elastic collisions. Queues build up because there is a limit to how fast transactions can be expedited by a communications channel.

Queues build up because there is a limit to how fast transactions can be expedited by a communications channel.

Reversible Health and Its Escape Velocity

The message I have been underlining above is that there is a fundamental problem where limited resources are involved. The problem is that reversibility (the ability to undo) depends on information stored, but that information stored is information lost in terms of resources. There is an exception to this idea, however. Some corrections occur in spite of no log being made.

What goes up must come down. Common colds and other mild diseases are not fatal to otherwise healthy individuals. The pinball will end up in its firing slot. A compass always points to magnetic North. Moths fly toward a flame. Adults prefer to look younger. Follow the light at the end of the tunnel. Ideals.

If you drop a ball to the ground, it does not usually land at your feet and remain there: the ground is seldom flat, so it begins to roll downhill until it finds a suitable local minimum. It tries to minimize its potential energy under the incentive of gravitation. Now energy is just a fictitious book-keeping parameter which keeps track of how much it cost to lift the ball up earlier and how much will be repaid by letting it roll down again. Like entropy, energy is a summary of information about how resources are distributed in the face of an unevenness. In thermodynamics, energy U and entropy S appear in relationships with the opposite sign: dF = dU - TdS

A potential is an anti-entropy device. It makes a difference by weighting the possibilities.

F is the free energy, or the amount of energy available for conversion into useful work, while U is the total energy and S is the entropy (the amount of energy which is spread about in a useless form). T is the temperature, which acts as an essentially irrelevant integrating factor for the entropy in this formulation.

A potential is an anti-entropy device. It makes a difference by weighting the possibilities. It tips the balance from pure randomness, to favor one or few possibilities or ideals. Think of it as a subsidy, according to some accounting principle, which makes certain configurations cheaper and therefore more likely. A potential can guide us into right or wrong, healthy or unhealthy states if it is chosen appropriately. While entropy is simply a result of statistical likelihood (deviation from ideal due to random change), a potential actually makes a choice.

Potentials are all around us. Indeed, they are the only thing that make the world an interesting place. Without these constraints on behavior, the Earth would not go around the sun; in fact it would never have formed. The universe would just be a bad tasting soup. Emotions are potentials which guide animal behavior and help us to survive. I have often wondered why the writers of *Star Trek* made such an issue about why the android Data supposedly has no emotions. For an android without emotions, he exhibits constantly emotional behavior. He is motivated, makes decisions, shows complex "state" behavior, worries about friends and has "ethical subroutines." The fact that he does not have much of a sense of humor could just as well mean that he originated from somewhere in Scandinavia (this would perhaps explain the pale skin, too).

Probably, complex animals could not develop adaptive behavior (intelligence) without emotions. Whatever we call complex-state information, it amounts to an emotional condition. Emotions may have a regulatory function or a motivational function. They provide a potential landscape which drives us in particular directions at different times. They alter the path of least resistance by enticing us to roll into their local minima. It's pinball with our heads. There is probably no other way of building complex adaptive behavior than through this kind of internal condition. We think of our emotions as being fairly coarse: happy, sad, depressed, aroused, but in fact, they have complex shades. We just don't have names for them, because as was noted in the last issue, we digitize.

Reversal of state can be accomplished without memory of the details, if there is an independent notion of what ideal state is: a potential well, like an emotional context, driving the system towards some tempting goal. That is because a potential curves the pathways and effectively makes them distinguishable from one another, labeling them with the value of the potential function at every point. Health is such a state: a complex multifaceted state whose potential is implemented in terms of a dynamical immune system rather than a static force. Ecologies and societies also have emergent goals and preferences. These represent a highly compressed form of information, which perhaps summarizes a great deal of complexity, just as a curve through a set of points approximates possibly more complex behavior.

If one could make a computer's ideal condition, its path of least resistance, how simple it would be to maintain its stability. Usually, it's not that simple, however. The playing field is rather flat, and armies battle for their position. It is as though hordes of barbaric changes are trying to escape into the system, while administrators representing the forces of law and order try to annex them. Neither one has any particular advantage other than surprise, unless the enemy can be placed in a pit.

Computer systems remain healthy and alive when they recycle their resources and do not drift from their ideal state. This was the idea behind *cfengine* when I started writing it eight years ago. The ideal computer state is decided by system policy, and this weights the probabilities p_n so that randomness favors a part of the good, rather than the bad or the ugly. Although a potential can only guide the behavior on average (there will always be some escape velocity which will allow a system to break its bounds), the likelihood of its long-term survival, in a world of limited resources, can only be increased by compressing complex information into simple potentials. This was the message of *cfengine* and the message of my papers at LISA 1998 and 2000. Now all we need is to design the pinball table.

So p_n – you feelin' lucky?

Declarations in c9x

by Glen McCluskey

Glen McCluskey is a consultant with 20 years of experience and has focused on programming languages since 1988. He specializes in Java and C++ performance, testing, and technical documentation areas.

<glenm@glenmccl.com>

Last time, we started looking at some of the new language features in C9X, the recent update to C. In this column we'll look at C9X declarations and how they have changed from what you are familiar with.

Function Declarations

If you've used C for a long time, you might remember that the language was once much looser about function declarations. You didn't have to declare functions before use; for example, you could say:

```
void f()
{
    g(37);
}
```

without complaint. If you didn't specify a return type for a function:

```
f()
{
}
```

it would be assumed to be int, like this:

```
int f()
{
}
```

And you didn't have to use a return statement, or maybe the return statement didn't match the return type:

```
int f()
{
    return;
}
```

These examples are invalid in C9X; you must declare functions before use, you cannot default the return type of a function, the return type must match return statements, and a return statement is required for a function with a non-void return type.

You are still allowed to declare functions without a prototype:

```
void f();

void g()
{
    f(37, 47);
}
```

but in the C9X standard this usage is marked as obsolescent. Note also that:

```
void f();
```

is not the same as:

```
void f(void);
```

or:

```
void f() {}
```

The first of these is a non-prototype function declaration, with parameter information unspecified, while the latter two declarations specify that the function has no parameters.

These tighter rules are not arbitrary; they lead to better code. Consider, for example, this program:

```
#include <stdio.h>

void f();

int main()
{
    f(37);

    return 0;
}

void f(int a, int b)
{
    printf("%d %d\n", a, b);
}
```

We use an old-style declaration of f(), then call the function with a single argument. The function actually requires two arguments, so when the printf() is encountered, one of the parameters has a garbage value.

The same consideration applies to return statements, if, for example, you fail to return a value from a function declared to return an int.

Inline Functions

With C9X you can define a function using the inline keyword, like this:

```
inline void f() {...}
```

inline is a hint to the compiler that it should optimize calls to the function, perhaps by expanding them in the context of the caller. There is no requirement that a compiler actually observe this hint.

Inline functions are quite subtle in a couple of areas, illustrated by the following example, composed of two translation units:

```
// file #1

void f()
{
}

void g(void);

int main()
{
    f();
    g();

    return 0;
}

// file #2

inline void f()
{
}

void g()
{
```

```
    f();
}
```

The inline definition of f() has external linkage. To give the function internal linkage, we would use static inline void f().

This definition is not an external definition, and because f() is actually called, an external definition is required somewhere within the program. This definition is found in the first translation unit.

A program can contain both an inline and an external definition of a function. The program cannot rely on the inline definition being used, even if it is clearly visible, as in the case above where g() calls f().

The inline and external definitions are considered distinct, and a program's behavior should not depend on which is actually called. As another example of this idea, consider:

```
// file #1

const int* f()
{
    static const int x = 0;
    return &x;
}

// file #2

inline const int* f()
{
    static const int x = 0;
    return &x;
}

int main()
{
    return f() == f();
}
```

Given these definitions of f(), it is not necessarily the case that main() always returns 1. Different f()s may be called, containing different static objects. An example of where this might happen would be with a compiler that applies some heuristic about how big inline expansions are allowed to get, with the external definition of the function used in cases where the inline expansion would be too large.

The C inline model is similar to that of C++, but differs in that (1) if a function is declared inline in one translation unit, it need not be declared so in all translation units, and (2) all definitions of an inline function need not be the same, but the program's behavior should not depend on whether an external definition or an inline version of a function is called.

Mixed Declarations and Code

In C9X you can intersperse declarations and code, a feature also found in C++. It's no longer necessary to place all declarations at the beginning of a block. Here's an example that counts the number of "A" characters in a file, using this coding style:

```
#include <stdio.h>

int main(int argc, char* argv[])
```

```
{
    if (argv[1] == NULL) {
        fprintf(stderr, "Missing input file\n");
        return 1;
    }

    FILE* fp = fopen(argv[1], "r");
    if (fp == NULL) {
        fprintf(stderr, "Cannot open input file\n");
        return 1;
    }

    int cnt = 0;
    int c;

    while ((c = getc(fp)) != EOF) {
        if (c == 'A')
            cnt++;
    }

    fclose(fp);

    printf("count = %d\n", cnt);

    return 0;
}
```

This style is in many ways more natural than the older approach of grouping all declarations at the top of a block. Each declaration is introduced as needed, in a meaningful context, and there's less temptation to recycle declarations, by using a variable in multiple ways within a function.

The scope of each identifier declared in this way is from its point of declaration to the end of the block.

Declarations in for Statements

You can also use declarations within for statements, like this:

```
#include <stdio.h>

int main()
{
    for (int i = 1, j = 100; i <= 10; i++, j—)
        printf("%d %d\n", i, j);

    return 0;
}
```

This is obviously a convenient way to specify loop indices. If you use this approach, a common coding style is no longer valid:

```
int main()
{
    int max = 10;

    for (int i = 1; i <= max; i++) {
        if (i == 5)
            break;
    }
    if (i > max)
        ; // loop completed normally
```

```
    return 0;
}
```

The scope of the loop index goes to the end of the for statement, so the index name is unknown outside. If you want to code this way, you need to continue declaring the loop variable before the loop.

Here's another (legal) example of for statements, one that demonstrates the subtleties of scoping:

```
void f()
{
    int i = 37;

    for (int i = 47; i <= 100; i++) {
        int i = 57;
    }
}
```

The for statement introduces a new block, and the statement body is also a distinct block. To make the blocks more visible, we could write the code like this:

```
void f()
{
    int i = 37;

    {
        for (int i = 47; i <= 100; i++) {
            int i = 57;
        }
    }
}
```

The three declarations of i are all valid, because each occurs in a new scope. Each declaration hides variables of the same name in outer scopes.

The new declaration features promote better programming and more natural documentation. They are natural to use and reduce programming errors.

using CORBA with java

A Mini Napster, Part II

by Prithvi Rao

Prithvi Rao is the co-founder of KiwiLabs, which specializes in software engineering methodology and Java/CORBA training. He has also worked on the development of the MACH OS and a real-time version of MACH. He is an adjunct faculty at Carnegie Mellon and teaches in the Heinz School of Public Policy and Management.

<prithvi+@ux4.sp.cs.cmu.edu>

Introduction

In Part I of this two-part series, I presented the development of a Java client/server application using CORBA. Specifically, I embarked on the development of a "mini Napster" example.

The development effort began with the writing of a Napster IDL (Interface Definition Language) and ended with the description of the functionality of the files that were generated as a result of compiling Napster.idl using the idltojava compiler.

In this article I wish to present the client and server code to complete this example. I hope to demonstrate that writing a CORBA application is neither overwhelming nor limited to those practitioners with advanced knowledge of software engineering and a general software development background.

Indeed, I successfully teach this topic to many students at Carnegie Mellon University who have not had a significant amount of experience in writing software or distributed client-server applications.

The Napster Server

In this section I present the implementation of the Napster Server (NapsterServer.java).

Import the CORBA packages for naming and locating CORBA objects:

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.util.*;
```

Import the package that contains the server skeletons:

```
import Napster.*;
```

Implement the interface generated by the idltojava compiler:

```
public class NapsterServer extends _NapsterServerIImplBase {
    private Vector records_database;
```

Build the constructor for the NapsterServer class:

```
    public NapsterServer()
    {
    records_database = new Vector(10, 5);
    }
```

Write the findItemInServer method:

```
    public Record findItemInServer(String albumName)
    {
        // Get the record with the highest version number
    Enumeration record_iter         = records_database.elements();
    int highest_version_number      = -1;
        int index                   = -1;

        while(record_iter.hasMoreElements())
        {
        Record record = (Record) record_iter.nextElement();
```

```
            if(record.album_name.equals(albumName))
            {
                if(record.version > highest_version_number)
                {
                    highest_version_number = record.version;
                    index = records_database.indexOf(record);
                }
            }
        }

        // Check if a record with album_name = albumName was found
    if(highest_version_number == -1)
    {
        // The record does not exist in the database so return a dummy record
        // with an empty album name. Also you need to set the other fields to
        // some dummy values even though you don't need them logically,
        // because otherwise CORBA will throw an Exception
            Record dummy_record          = new Record();
            dummy_record.album_name      = " ";
            dummy_record.artist_name     = " ";
            dummy_record.owner_name      = " ";
            dummy_record.version         = -1;
            return dummy_record;
        }

        // If we are here then we must have found the record in the database,
        // so return the found record
        return (Record) records_database.elementAt(index);
    }
```

Write the addRecordInServer method:

```
  public String addRecordInServer(Record desiredRecord)
    {
        // Get the record with the highest version number
    Enumeration record_iter = records_database.elements();
        while(record_iter.hasMoreElements())
        {
        Record record = (Record)record_iter.nextElement();
            if((record.album_name.equals(desiredRecord.album_name)) &&
            (record.artist_name.equals(desiredRecord.artist_name)))
            if(record.version >= desiredRecord.version)
            return "Duplicate Record, Record Not Added to the Server";
        } // while

        // We can now safely add the record to the database
    records_database.addElement(desiredRecord);
    return "Record Successfully Added";
  }
```

Write the deleteItemInServer method:

```
  public boolean deleteItemInServer(Record desiredRecord)
    {
        int num_of_records       = records_database.size();
        boolean record_deleted   = false;
        for(int i = 0; i < num_of_records; i++)
        {
```

```
            Record record = (Record) records_database.elementAt(i);
            if((record.album_name.equals(desiredRecord.album_name)) &&
                    (record.artist_name.equals(desiredRecord.artist_name)))
            {
                records_database.removeElementAt(i);
                record_deleted = true;
            }
        }

        return record_deleted;
    }
```

Write the updateRecordInServer method

```
public boolean updateRecordInServer(Record desiredRecord, String
                                    newOwner)
{
        // Get the record with the highest version number
    Enumeration record_iter         = records_database.elements();
        int highest_version_number = 0;
        int index                  = 0;

        while(record_iter.hasMoreElements())
        {
            Record record = (Record)record_iter.nextElement();
            if((record.album_name.equals(desiredRecord.album_name)) &&
                    (record.artist_name.equals(desiredRecord.artist_name)))
            {
                if(record.version > highest_version_number)
                {
                    highest_version_number = record.version;
                    index = records_database.indexOf(record);
                }
            }
        }

        // If the record was not found then return false
        if(index == 0)
            return false;

        // Otherwise update the record
        Record record          = (Record) records_database.elementAt(index);
        record.owner_name      = newOwner;
        record.version +       = 1;

        records_database.addElement(record);

        return true;
    }
}
```

The Napster Main Class

The following is the Napster main class (Napster.java) in which all the CORBA work happens. First import all the CORBA packages:

```
import Napster.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
```

```
import org.omg.CORBA.*;

public class Napster
{
    public static void main(String args[])
    {
        try
        {
```

Create the ORB and initialize it. This is where the ORB knows how to find the location of the NameServer with which to register the name of the server objects.

```
            ORB orb = ORB.init(args, null);
```

Create a Server Object here:

```
            NapsterServer napster_server = new NapsterServer();
```

Connect the Server Object to the ORB:

```
            orb.connect(napster_server);
```

Get a reference to the nameserver:

```
            org.omg.CORBA.Object objRef =
                                    orb.resolve_initial_references("NameService");
```

Get an object reference that is a "Java" object and not a CORBA object:

```
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

Make sure that the server object can be located using "NapsterServer":

```
            NameComponent nc = new NameComponent("NapsterServer", "");
```

Save this in a component array:

```
        NameComponent path[]    = {nc};
```

Bind this component name with the ORB so that the client can get a reference to the server object:

```
        ncRef.rebind(path, napster_server);
```

Make sure that you can have multiple clients connect to the server:

```
            java.lang.Object sync = new java.lang.Object();
            synchronized(sync)
            {
                sync.wait();
            } // try

            catch(Exception e) {
                System.err.println("ERROR: " + e);
                e.printStackTrace(System.out);
            }
        }

    }
```

The Napster Client

In this section I present the Napster Client (NapsterClient.java). First import the CORBA packages and other Java packages:

```
import Napster.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class NapsterClient
{
private NapsterServerI napster_server;
private BufferedReader stdin;
```

Write the constructor for the NapsterClient:

```
public NapsterClient(NapsterServerI p_napster_server, BufferedReader p_stdin)
{
napster_server = p_napster_server;
stdin   = p_stdin;
}
```

Write the method to get a record:

```
public void getRecord()
{
try {
     System.out.print("Please Enter the Album Name: ");
String album_name      = stdin.readLine();
     System.out.print("Please Enter Your Name: ");
String user_name       = stdin.readLine() ;

Record found_record    = new Record();

found_record           = napster_server.findItemInServer(album_name) ;

// If we don't find the record then give an appropriate message to the user
if(!found_record.album_name.equals(album_name))
     System.out.print("The Album Name That You Entered Does Not Exist");
else
{
// We found the record. So create a new record with the current user
// as owner and an increased version number to make it the latest record
     System.out.println("Information Regarding the Album:");
     System.out.println("Album Name : " + found_record.album_name);
     System.out.println("Artist Name   : " + found_record.artist_name);
     System.out.println("Owner         : " + found_record.owner_name);
     System.out.println("Version       : " + found_record.version);

Record new_record = new Record();
     new_record.album_name    = album_name;
     new_record.artist_name   = found_record.artist_name;
     new_record.owner_name    = user_name;
     new_record.version       = ++found_record.version;

String server_response = napster_server.addRecordInServer(new_record);

     System.out.println("New Record with the Following Info Added to the
                         Server:");
     System.out.println("Album Name  : " + new_record.album_name);
     System.out.println("Artist Name   : " + new_record.artist_name);
     System.out.println("Owne          : " + new_record.owner_name);
```

```
        System.out.println("Version: " + new_record.version);

        System.out.print(server_response);
        } // else
        } // try

            catch(Exception e)
            {
            System.out.println("Error: " + e);
            e.printStackTrace(System.out);
        } // catch
    }
```

Write the method to add a record:

```
    public void addRecord()
    {
        try
            {
            // Get the fields of the new record to create

            System.out.print("Please Enter Your Name: ");
            String user_name       =  stdin.readLine();

            System.out.print("Please Enter the Album Name: ");
            String album_name     =  stdin.readLine();

            System.out.print("Please Enter the Artist Name: ");
            String artist_name      =  stdin.readLine() ;

            // Create a new record and initialize its fields

            Record new_record         =     new Record();
            new_record.album_name =     album_name;
            new_record.owner_name =     user_name;
            new_record.artist_name  =     artist_name;
            new_record.version        =     1;

        String server_response = napster_server.addRecordInServer(new_record);
            system.out.print(server_response);

            } //try
        catch(Exception e)
        {
        System.out.println("Error: " + e);
        e.printStackTrace(System.out);
        } // catch
    }
```

Write a method to delete a method:

```
    public void deleteRecord()
    {
        try
            {
            System.out.print("Please Enter the Album Name: ");
            String album_name     = stdin.readLine()j;

            System.out.print("Please Enter the Artist Name: ");
            String artist_name      = stdin.readLine();
```

```
        // Create a record that you would like to be deleted from the server
        // database

        Record new_record          =  new Record();
        new_record.album_name      =  album_name;
        new_record.owner_name      =  new String("Aravind");
        new_record.artist_name     =  artist_name;
        new_record.version         =  1;

    System.out.println("Before Calling Server Delete");
    boolean record_deleted = napster_server.deleteItemInServer(new_record);

    System.out.println("After Calling Server Delete");

    if(record_deleted)
        System.out.print("The Record Was Successfully Deleted on the
                    Server");
    else
        System.out.print("The Record Could Not Be Deleted on the Server");
        } // try
    catch(Exception e)
    {
    System.out.println("Error: " + e);
    e.printStackTrace(System.out);
    } // catch
}
```

Write a method to update a record:

```
public void updateRecord()
{
    try
        {
        System.out.print("Please Enter the Album Name: ");
        String album_name = stdin.readLine();

        System.out.print("Please Enter the Artist Name: ");
        String artist_name = stdin.readLine() ;

        // Create a record that you would like to be updated from the server
        // database
        Record update_record          =  new Record();
        update_record.album_name      =  album_name;
        update_record.artist_name     =  artist_name;

        // Got to give dummy values for the other values in the record otherwise
        // CORBA will complain
        update_record.owner_name    = new String(" ");
        update_record.version       = 1;

        System.out.print("Please Enter the New Owner Name: ");
        String new_owner = stdin.readLine() ;

    // Call the update function on the server
    boolean record_updated =
          napster_server.updateRecordInServer(update_record, new_owner);

        if(record_updated)
        {
            System.out.print("The Record Was Successfully Updated on the
                    Server");
```

```
            System.out.println("Album Name  : " + album_name);
            System.out.println("Artist Name   : " + artist_name);
            System.out.println("Owne          : " + new_owner);
            }
            else
            System.out.print("The Record Could Not Be Updated on the Server");
          }
          // try
          catch(Exception e)
          {
          System.out.println("Error: " + e);
          e.printStackTrace(System.out);
        } // catch
}
```

Write a method to display menu options:

```
public void displayMenu()
      {
      // Display the Menu
      System.out.println("\n ");
      System.out.println("Enter One of the Following Options");
      System.out.println("1. To Get an Album on the Server");
      System.out.println("2. To Add an Album to the Server");
      System.out.println("3. To Delete an Album on the Server");
      System.out.println("4. To Update an Album on the Server");
      System.out.println("5. To Exit");
      }
```

Write the client main method:

```
public static void main(String args[])
{
try
{
```

Initialize the ORB. The args array tells the ORB on the client machine where the name-server is running (machine name and port to use to connect to the nameserver):

```
ORB orb = ORB.init(args, null);
```

Get a reference to the NameServer:

```
org.omg.CORBA.Object objRef = orb.resolve_initial_references
                                        ("NameService");
```

Create a Java object from a CORBA object:

```
NamingContext ncRef   = NamingContextHelper.narrow(objRef);
```

Find the path on the NameServer where the server object is located:

```
NameComponent nc     = new NameComponent("NapsterServer", " ");
```

Set the path:

```
NameComponent path[]= {nc};
```

Get a server object and narrow the reference from a CORBA object to a java object (all in one call):

```
NapsterServerI napster_server =
NapsterServerIHelper.narrow(ncRef.resolve(path));
BufferedReader stdin =
new BufferedReader(new InputStreamReader(System.in));
```

Start the client:

```
NapsterClient napster_client = new NapsterClient(napster_server, stdin);

String choice_str;
int choice_int = 1;

// Keep looping till the user tells us to quit while(choice_int != 5)
{
      napster_client.displayMenu() ;
      choice_str = stdin.readLine() ;
      choice_int = Integer.parseInt(choice_str) ;

      switch(choice_int)
      {
         case 1:
         napster_client.getRecord();
            break;
         case 2:
         napster_client.addRecord() ;
            break;
         case 3:
         napster_client.deleteRecord();
            break;
         case 4:
         napster_client.updateRecord() ;
            break;
         case 5:
         choice_int = 5;
            break;
         default:
         System.out.println("Invalid Option. Please Try Again");
            break;
         } // switch
            } // while
         }
         catch(Exception e)
      {
         System.out.println("Error: " + e);
         e.printStackTrace(System.out);
         } // catch
      } // main
} // class
```

Running Napster

1. Install the idltojava compiler.

2. Install JDK1.2.1.

3. Compile the Napster.idl file: idltojava Napster.idl.

4. Compile the remaining files: Napster.java, NapsterServer.java, NapsterClient.java.

```
javac *.java Napster/*.java
```

5. Run the nameserver.

```
tnameserv -ORBInitialHost <IP address of machine running nameserver>
        -ORBInitialPort <port used: default is 900>
```

6. Run the NapsterServer.

```
java Napster -ORBInitialHost <IP address of machine running nameserver>
        -ORBInitialPort <port used: default is 900>
```

7. Run the NapsterClient.

```
java NapsterClient -ORBInitialHost <IP address of machine running
                        nameserver>
        -ORBInitialPort <port used: deafault is 900>
```

Discussion

It is evident that the server and clients both perform similar steps for initialization. Specifically, the server has to register its server object implementations in the nameserver so that the client knows where to find them in the nameserver. It is also evident that conversions from CORBA to Java objects must be performed because the Java interpreter cannot manipulate CORBA objects.

Conclusion

In this two-part series I have presented the development of a mini Napster example. This example is not particularly sophisticated, but it demonstrates how to write a Java client-server application using CORBA.

We can make the observation that once the CORBA initialization is complete then the rest of the development effort is a matter of pure Java. Indeed, this is mostly true in larger Java/CORBA applications. The main considerations when using advanced features of CORBA relate to the various CORBA services such as security, event, transaction, timing, and many others. Even so, the programming task is to gain access to a server object implementation providing the service.

The Napster example hopefully accomplishes at least the following:

- Demonstrates that CORBA is no more difficult to understand than Java
- Demonstrates that the CORBA and Java object model are very similar (modules match to packages and interfaces in CORBA and to interfaces in Java)

The true power of CORBA can best be understood by doing, and hopefully this article has provided readers with the incentive to perform further investigation into this powerful technology.

intrusion detection

by Clif Flynt

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

<clif@cflynt.com>

The philosophy that drove the design of most of the UNIX system data files is that it's best to sacrifice a little speed in favor of ease of maintenance. Thus, most data files (/etc/password, /etc/hosts, etc.) contain printable ASCII data. If you need to examine them you can do it with more, and if you need to repair them, you can do it with vi.

There are some files that contain binary data: the utmp file, for example. The utmp file contains data about all the users on a system (assuming everything is working correctly), including their login id, the time they logged in, the IP address they logged in from, etc.

It might be nice to watch this file and report when it changes. For example, if there is suddenly someone running as root on our firewall, it might be something we are interested in knowing about.

Early versions of Tcl supported only ASCII strings, and could not handle binary data. With version 8.0, Tcl moved to a more versatile internal data representation and added the binary command. The binary command allows easy conversion from binary representations to printable ASCII representations of data. Tcl is still oriented around printable ASCII strings, but the binary command makes it possible to handle binary data as well.

The binary command has two subcommands, binary scan and binary format.

The syntax for these is:

Syntax: binary format formatString arg1 ?arg2? ... ?argn?

binary format	Returns a binary string created by converting one or more printable ASCII strings to a binary format.
formatString	A string that describes the format of the ASCII data.
arg*	The printable ASCII to convert.

Syntax: binary scan binaryString formatString arg1 ?varName1? ... ?varNamen?

binary scan	Converts a binary string to one or more printable ASCII strings
binaryString	The binary data.
formatString	A string that describes the format of the ASCII data.
*varName**	Names of variables to accept the printable representation of the binary data.

The formatString for the binary commands allows binary data to be collected from or distributed to a number of variables in a variety of formats. It resembles a regular expression string, but has a slightly different flavor.

Like the regular expression string, a binary command format string is composed of sets of two fields – a descriptor for the type of data, followed by an optional count modifier.

The binary command supports several identifiers for converting strings, decimal data, hex data, or floating point values by 8-bit, 16-bit, or 32-bit data widths. Here are a few of the commonly used descriptors:

h Converts from binary to/from hexadecimal digits in little-endian order.
binary format h2 34 – returns "C" (0x43).
binary scan "4" h2 x – stores 0x43 in the variable x

H Converts from binary to/from Hexadecimal digits in big-endian order.
binary format H2 34 – returns "4" (0x34).
binary scan "4" H2 x – stores 0x34 in the variable x

c Converts an 8-bit value to/from ASCII.
binary format c 0x34 – returns "4" (0x34).
binary scan "4" c x – stores 0x34 in the variable x

s Converts a 16-bit value to/from ASCII in little-endian order.
binary format s 0x3435 – returns "54" (0x350x34).
binary scan "45" s x – stores 13620 (0x3534) in the variable x

S Converts a 16-bit value to/from ASCII in big-endian order.
binary format S 0x3435 – returns "45" (0x350x34).
binary scan "45" S x – stores 13365 (0x3435) in the variable x

i Converts a 32-bit value to/from ASCII in little-endian order.
binary format i 0x34353637 – returns "7654" (0x350x34).
binary scan "45" s x – stores 13620 (0x3534) in the variable x

I Converts a 32-bit value to/from ASCII in big-endian order.
binary format I 0x34353637 – returns "4567" (0x350x34).
binary scan "45" S x – stores 13365 (0x3435) in the variable x

f Converts 32-bit floating point values to/from ASCII.
binary format f 1.0 – returns the binary string "0x0000803f".
binary scan "\x00\x00\s80\83f" fx – stores 1.0 in the variable x

The optional count can be an integer, to list the exact number of conversions to perform, or a *, to use all remaining data.

The format string can be arbitrarily complex, with multiple descriptor/count pairs separated by spaces.

Here's an example of some C code to write a structure to a disk file, and the Tcl code to read and translate the data:

C Code to Generate a Structure

```
#include <stdio.h>
#include <fcntl.h>
main () {
   struct a {
      int i;
      float f[2];
      char s[20];
   } aa;
   FILE *of;

   aa.i = 100;
   aa.f[0] = 2.5;
   aa.f[1] = 3.8;
   strcpy(aa.s, "This is a test");

   of = fopen("tstStruct", "w");
   fwrite(&aa, sizeof(aa), 1, of);
   fclose(of);
}
```

Tcl Code to Read the Structure

```
# Open the input file, and read data
set if [open tstStruct r]
set d [read $if]
close $if

# scan the binary data into variables.

binary scan $d "i f2 a*" i f s
# The string data includes any
# binary garbage after the NULL byte.
# Strip off that junk.

set 0pos [string first [binary format c 0x00] $s]
incr 0pos -1
set s [string range $s 0 $0pos]

# Display the results
puts $i
puts $f
puts $s
```

The output from the Tcl code is:

```
100
2.5 3.79999995232
This is a test
```

The flip side to this is to write a structure in Tcl, and read it with a C program. This pair of programs will perform that operation.

Tcl Code to Generate a Structure

```
set str [binary format "i f2 a20" 100
                {23.4 56.78} "the other test"]

set if [open tstStruct2 w]
puts -nonewline $if $st
close $if
```

C Code to Read the Structure

```
#include <stdio.h>
#include <fcntl.h>
main () {
    struct a {
        int i;
        float f[2];

        char s[20];
    } aa;

    FILE *of;

    of = fopen("tstStruct2", "r");
    fread(&aa, sizeof(aa), 1, of);
    fclose(of);

    printf("I: %d\n", aa.i);
    printf("f[0]: %f ", aa.f[0]);
    printf("f[1]: %f\n", aa.f[1]);
    printf("nstr: %s\n", aa.s);
}
```

The C program generates this output:

```
I: 100
f[0]: 23.400000 f[1]: 56.779999
str: the other test
```

The binary command makes it (relatively) easy to parse the utmp file. All we need to do is look up the utmp.h include file on our system, examine the structure definition, and create a format string that the binary command can use to parse each structure in the utmp file.

On a Linux system, the utmp structure looks like this:

```
/* The structure describing an entry in the user accounting database.  */
struct utmp
{
    short int ut_type;              /* Type of login.  */
    pid_t ut_pid;                   /* Process ID of login process.  */
    char ut_line[UT_LINESIZE];      /* Devicename.  */
    char ut_id[4];                  /* Inittab ID.  */
    char ut_user[UT_NAMESIZE];      /* Username.  */
    char ut_host[UT_HOSTSIZE];      /* Hostname for remote login.  */
    struct exit_status ut_exit;     /* Exit status of a process marked
                                       as DEAD_PROCESS.  */
    long int ut_session;            /* Session ID, used for windowing.  */
    struct timeval ut_tv;           /* Time entry was made.  */
    int32_t ut_addr_v6[4];          /* Internet address of remote host.  */
    char __unused[20];              /* Reserved for future use.  */
```

```
};
```

This would lead you to believe that a format string like this should do the trick for extracting the members of the structure:

```
#       type  pid   line  id  user  host   exit  session   time  addr
set f   "s    i     a32   a4  a32   a256   s2    i         i2    i4"
```

In a perfect world, this would work fine.

In this world, certain processors require that an integer start on an integer boundary, and if a structure declares a single short followed by a long integer, then there will be two bytes of padding added so that the integer can start on a long-word boundary.

The binary command includes a type descriptor that is not a data type: the @ character. Where the other type descriptors accept a count modifier, the @ descriptor will accept an absolute location in the data as a modifier.

This can be used to set the imaginary cursor in the binary data to a long-word boundary, skipping the padding. The @ can also be used to set the cursor location to the start of each structure in the dataset.

The next trick is that while the utmp structure is allowing for an IPV6 address, only the first 4 bytes are actually being used today. So, rather than using i4 for the IP address, we can use c4 (to read 4 bytes of address).

This format string works for Linux:

```
# set st to the start of the structure
#     start   type  padding        pid  line  id  user  host   exit  session time addr
set f "@$st   s     @[expr $st+4] i    a32   a4  a32   a256   s2    i        i2   c4"
```

This would let us generate a report with all the numbers converted to a printable format. This is better than nothing (but not a lot better).

The first field is the type of process described in this record. The utmp.h file describes the meanings of these types. It's a relatively easy task to cut and paste from that file, edit a little, and convert the #define lines to a Tcl associative array that we can use as a lookup table to convert the numeric types to a more human-friendly value:

```
foreach {name num} {
        EMPTY            0     RUN_LVL              1
        BOOT_TIME        2     NEW_TIME             3
        OLD_TIME         4     INIT_PROCESS         5
        LOGIN_PROCESS    6     USER_PROCESS         7
        DEAD_PROCESS     8     ACCOUNTING           9} {
    set types($num) $name
}
```

The time stamp can be converted to a more human-friendly form with the Tcl clock command.

The Tcl clock command has several subcommands that will obtain the current time (in seconds since the epoch), convert a time in seconds to human-readable format, or convert a human-style time/date into seconds.

This Tcl command will convert the system format time in the first field of the timeval structure into a date and time in the format MM/DD/YY HH:MM:SS:

```
clock format [lindex $time 0] -format "%D %r"
```

Finally, we don't want to report on the contents of the utmp file every 10 seconds, or even every minute. We just want to know what's happened if it changes.

The Tcl file command also has many subcommands. A useful one for this application is the file mtime that reports the last modification time for a file.

We can't set a trigger to go off when a file is modified, but we can loop on the file mtime value and only report the contents of the utmp file when the modification time changes.

```
set mtime 0
...
    while {[file mtime /var/run/utmp] == $mtime} {
        after 10000
    }
    set mtime [file mtime /var/run/utmp]
```

The code shown below will generate output resembling this when someone logs in:

Type	Pid	Line	ID	User	Host	Exit	Session		Time	Addr
DEAD_PROCESS	7		si			0 0	0	04/03/01	07:23:27 PM	0.0.0.0
BOOT_TIME	0	~	~~	reboot		0 0	0	04/03/01	07:23:27 PM	0.0.0.0
RUN_LVL	20019	~	~~	runlevel		0 0	0	04/03/01	07:23:27 PM	0.0.0.0
DEAD_PROCESS	157		l3			0 0	0	04/03/01	07:24:05 PM	0.0.0.0
DEAD_PROCESS	01		ud			0 0	0	04/03/01	07:24:05 PM	0.0.0.0
USER_PROCESS	702	tty1	1	lclif		0 0	0	04/03/01	07:24:34 PM	0.0.0.0
LOGIN_PROCESS	703	tty2	2	LOGIN		0 0	0	04/03/01	07:24:05 PM	0.0.0.0
LOGIN_PROCESS	704	tty3	3	LOGIN		0 0	0	04/03/01	07:24:05 PM	0.0.0.0
LOGIN_PROCESS	705	tty4	4	LOGIN		0 0	0	04/03/01	07:24:05 PM	0.0.0.0
LOGIN_PROCESS	706	tty5	5	LOGIN		0 0	0	04/03/01	07:24:05 PM	0.0.0.0
LOGIN_PROCESS	707	tty6	6	LOGIN		0 0	0	04/03/01	07:24:05 PM	0.0.0.0
USER_PROCESS	746	pts/2	/2	lclif		0 0	0	04/03/01	07:24:44 PM	0.0.0.0
USER_PROCESS	743	pts/1	/1	lclif		0 0	0	04/03/01	07:24:44 PM	0.0.0.0
USER_PROCESS	744	pts/0	/0	lclif		0 0	0	04/03/01	07:24:44 PM	0.0.0.0
USER_PROCESS	745	pts/3	/3	lclif		0 0	0	04/03/01	07:24:44 PM	0.0.0.0
DEAD_PROCESS	999	pts/5	/5			0 0	0	04/03/01	08:47:04 PM	127. 0.0.1
USER_PROCESS	1163	pts/4	/4	clif	vlad	0 0	0	04/03/01	09:17:44 PM	127. 0.0.1

In the code below, I've assigned the output channel to be stdout for my testing. Since the first thing a hacker is likely to do after they've broken into your system is rewrite utmp to hide their presence, this monitor might be a good one to combine with the client/server-based monitors discussed in the previous articles to push the information off the possibly compromised system and onto a (hopefully) more secure (or at least less obvious) machine inside your network.

As usual, this code is available online at <*http://noucorp.com*>.

```
# Grab the type definitions from /usr/include/bits/utmp.h

foreach {name num} {
        EMPTY           0    RUN_LVL         1
        BOOT_TIME       2    NEW_TIME        3
        OLD_TIME        4    INIT_PROCESS    5
        LOGIN_PROCESS   6    USER_PROCESS    7
        DEAD_PROCESS    8    ACCOUNTING      9 } {
    set types($num) $name
}
```

```
# We'll use $zero to trim trailing zeros from the data.
set zero [binary format c 0x00]

# $reportFmt defines the widths of the columns in the report
set reportFmt {%-16s %6s %12s %20s %-12s %6s %5s %7s %21s %16s}

set output stdout

set mtime 0
while {1} {
    while {[file mtime /var/run/utmp] == $mtime} {
        after 10000
    }
    set mtime [file mtime /var/run/utmp]

    # Display a header
    puts [format $reportFmt Type Pid Line ID User Host Exit Session Time Addr]

    # Open read and close the utmp file
    set if [open /var/run/utmp r]
    set d [read $if]
    close $if

    # Save the length of the data buffer for future use.
  set utmpLen [string length $d]

  # This is the start of the utmp structure being examined
    set start 0

    # As long as there is data to parse, parse it and step to the
    # next structure.

    while {$start < $utmpLen} {
        # start of struct  type padding      pid line id user host exit\
        session time addr
        set fmt "@$start    s  @[expr $start +4] i   a32  a4 a32  a256 s2  \
            i       i2   c4"
        binary scan $d $fmt type pid line id user host exit session time addr

        # Trim trailing zeros
        foreach v {line id user host} {
            set $v [string trim [set $v] $zero]
        }

        puts $output [format $reportFmt \
            $types($type) $pid $line $id $user $host $exit $session \
            [clock format [lindex $time 0] -format "%D %r"] [join $addr "."]]

        incr start 384
    }
}
```

musings

It is truly amazing how busy one can get.

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

<rik@spirit.com>

By the time you get this issue of *;login:*, it will be the beginning of summer. Summer often means that the pace of work slows down, as students go home or coworkers take vacations. And the slowdown in the economy is already helping some people. Although most of the layoffs (firings, in a less gilded age) are in factories, some are in the tech sector. And others who have survived the dot-com craze are now working at a much more reasonable pace.

I was fortunate enough to meet some of the movers and shakers for a two-day meeting that finished two days before I wrote this. You might not consider the Linux Developers conference a relaxing event, and you would be right, but it was certainly stimulating. I had often wondered about the people who write and guide the creation of the Linux kernel. You can find out more about this event, and something about the personalities involved, by reading my summaries in this issue.

I also discovered just the thing to while away the idle hours I don't have, and perhaps you have already heard of it. The Honeynet Project (<*http://project.honeynet.org/challenge/*>) is a collection of security guys who have decided to focus on attack signatures and forensics. They had posted a forensics challenge, and one of their number, Dave Dittrich of the University of Washington, let me know about it in January. I was too buried to even think about it then, but by the middle of February things had calmed down enough for me to take a peek at the challenge, and when I did, I started to get really excited. Well, you really have to be interested in security like I am to get turned on by something like this.

As one of their projects, members of the group monitor networks looking for attack signatures, using the free ID software snort. On the night of November 7, snort detected a scan and then an attack against a Linux system. The victim, a RedHat 6.2 server install, had only been up 2 1/2 days before the successful attack. One of the project members monitored the system, and once things had settled down, took it offline and made a full-image copy (using dd) of the entire hard disk. Good thing it was only a three gigabyte drive.

Conversion

The challenge was to determine how the system had been attacked, which tool was used, who did it, as well as what the attacker did after the attack. As a hint, snort logs from the initial attack were provided, as were the hard disk images. The hard disk had been carefully partitioned, each partition saved and gzipped, along with information about how to mount the partitions using the Linux loopback mount interface. The fact that the victim system was partitioned made the analysis much easier.

I started by mounting the partitions and looking around much like a system administrator might – especially one who had the benefit of an ID system. Well, this was a pretty good waste of time, although not totally fruitless. I started by looking for a service that listened to port 871/udp. I will tell you right away that RFC1700 does not list any service at that port. But, I knew that the attack followed a scan of rpcinfo at port 111. So I started scanning system startup files, looking to see what would have been started. Only the NFS lock services, statd and lockd, would have been started, and statd has a long history of exploits.

I also noticed that the startup script for syslogd had been deleted. The only ordinary user account was named drosen, and in that directory, I found a .bash_history file with a list of commands in it. That sent me off to /usr/man/.Ci, a "hidden" directory just chock

full of stuff. There were scripts here, "hidden" files, as well as directories with scanning and exploit tools.

It was at this point that I decided to download and install The Coroner's Toolkit (TCT) (<*http://www.porcupine.org/forensics/*>) and take advantage of the tools written by Dan Farmer and Wietse Venema. During the challenge, Brian Carrier of Purdue wrote some additional tools to supplement TCT (<*http://www.cerias.purdue.edu/homes/carrier/forensics.html*>), and it would have been handy to have some of these: for example, one that listed the contents of a directory inode (even if it had been deleted) and showed any filenames corresponding to deleted files. I was accustomed to using od -cx on the directory itself, but the Linux kernel balked at letting me read directories as files.

Before I make things look absurdly difficult, the top-rated investigator in the challenge started off with a much different approach. Thomas Roessler mounted the partitions and then used RedHat's rpm tool to determine, in conjunction with MD5 checksums and the still-installed package file, which files had been modified after the install. Another investigator used Tripwire checksums for a database created on an intact system. Both turned up a list of Trojans that had been installed: ls, ps, top, netstat, tcpd, ifconfig, and in.identd. Most of these are part of standard rootkits (that is, similar to one I was given back in 1994). The fake identd provides a root shell to anyone connecting from one of two "magic" source ports.

Autopsy

The standard RedHat 6.2 server install includes 30,000 files and directories, and we know that the attacker has at least attempted to hide things. Also, it is likely that some of the evidence of the attack was deleted. This is where the TCT comes into play. By running graverobber, you create the body file which is grist for the mactime script, which I'll discuss in a moment. What some of the other investigators did (and I wish I had too) was to use the unrm tool to search for deleted inodes, and then to convert the output of ils (inode ls) into a format suitable for inclusion in the body file. Then, when they run mactime, they can see not only the existing files and directories, but ones that have been deleted (and not yet reused) as well.

mactime is my favorite tool of the bunch. It presents a list sorted by access, modify, and/or inode change time. For example, when a program is executed, you see that its access time is modified, as well as the access times of any libraries loaded at that time. Or, when a file is modified, you can see when that happened. Of course, you can only see the last time a program was executed or a file modified. But, you can see things like a script named clean being executed by the attacker, which in turn called the snap script, which then strips lines that match certain patterns from logfiles.

Roughly speaking, here is what the attacker did. I will use the datestamps taken from the victim system, whose clock was set incorrectly (just to make things more interesting, no doubt).

The initial attack followed a couple of probes. First port 111/tcp was checked to see if it was open and reachable, then an RPC request was made, most likely checking for statd and the port it was listening to (871/udp). Forty-five seconds later, the attack used statd to write a new entry in /etc/inetd.conf that would run a root-owned shell at port 4545/tcp for anyone who connected to it. This part of the attack appeared to be automatic, which was very likely, considering some of the other stuff found on the victim. This happened shortly after 11 p.m.

At about 7:25 the next morning, the attacker connected to the magic port, and cleared /etc/host.deny, a control file for TCP wrappers. It was probably at this point that the attacker deleted the init script that starts syslogd, leaving many orphaned symlinks pointing nowhere. The attacker then used FTP to download some tools and scripts. One of these tools was used to add two new user accounts, own and adm1, and the attacker then disconnected and logged back in as adm1. The own account had a UID of zero and no password, so the attacker could immediately become root. Soon, the attacker cleaned up the inetd.conf file so no one else could take advantage of the original backdoor at port 4545.

Although the attacker deleted or cleaned up many logfiles, lastlog was forgotten. Linux has no tools that will display the content of any lastlogfile except one in the default location, so I wrote one (something that Roessler did as well) and discovered that the attacker had logged in at 7:31 as adm1 from c871553-b.jffsn1.mo.home.com (the initial attack came from an address within home.net).

The very next activity was to install tpack, an IRC bot used to keep control over IRC channels. The attacker later installed a recent version of BitchX (bx), an IRC client. This indicates a possible motive for the attack – to gain another platform for the IRC wars.

The attacker next started running scripts that backup programs, then install and configure the various Trojans mentioned earlier. Most of these Trojans are part of the lurker four (lrk4) package that you can find at sites like <*ftp://ftp.technotronic.com*>, although a couple are apparently from a later version, lrk5. The scripts are not perfect: for example, they use the wrong pathnames for a replacement telnetd and a Trojanned syslogd (that ignore log messages that match certain patterns).

The attacker's script also launches snif, a version of LinSniffer (also part of lrk4), leaving behind snif.pid and an empty logfile (no passwords were sniffed).

Good Sysadmin

I found what the attacker did next remarkable, although I have heard that this has become increasingly common. The attacker downloaded a set of rpms and patched the most commonly used security holes on Linux systems: amd, lpr, nfs-utils, wu-ftpd, and BIND. Another script ruthlessly stripped away set-user-id permissions on many programs, apparently to prevent anyone else from using an exploit to gain root access.

The reason for this paranoia became more apparent when I looked through the /usr/man/.Ci directory. I found scanning and exploit tools for exploiting each of the patched server programs, as well as z0ne and strobe for scanning. In general, each directory contained a program that generated IP addresses when given a 16-bit prefix (e.g., 206.1 to create 206.1.1.1, 206.1.1.2, up to 206.1.254.254), that got passed to a tool that probed for a particular port, BIND version, or RPC service, and finally to an automatic exploit. In other words, the system now became yet another automatic attack platform that, in turn, discovers more vulnerable systems. No wonder it took only 2 1/2 days before this system was exploited.

Like any security-conscious individual, the attacker installed a version of SSH. Unlike most versions, this one not only logs passwords, but it also has a magic password. The attacker logged out of the adm1 account, and logged back in using SSH and the magic password. The Trojan sshd dutifully logged the magic password, tw1Lightz0ne, in the logfile (/usr/tmp/nap).

By the time the attacker logged out, he had spent about 36 minutes downloading files, installing rpms, and running scripts. The adm1 and own accounts were deleted, and Trojans hid most (but not everything) that had been done. The attacker had two backdoors, sshd and identd, to use for future visits.

What the attacker took about a half hour to accomplish takes the average investigator about 34 hours to uncover. I spent close to 30 hours myself (I did not enter the challenge), simply because I did feel the challenge posed by this thoroughly hacked system.

In the usual case, this system would simply have been taken offline, and the OS reinstalled. If security patches had not also been installed, it is very likely that the system would have been hacked again in very short order. At the time I write this, the port being scanned most frequently is 53/tcp, the port used by BIND. If you have systems, even non-Linux ones, that use versions of BIND prior to 8.3 or 9.1, I suggest that you upgrade them soon.

The attack used by the Honeynet Project was not particularly subtle. Swift, a little messy, obviously not someone who was totally clueless, but (by the same token) someone who obviously was into taking over as many systems as possible. Not what I consider a serious attacker, who would have been considerably more subtle and left much less in the way of traces. If all the attacker had wanted was a copy of a certain file, the initial attack could have used a script that would have done everything, including cleaning up afterwards.

All in all, trying The Challenge was a very worthwhile learning experience. I suggest it as a fine way to pass the copious free time that you might have. On a more serious note, if you do take the time to read a couple of the analyses, you will learn a lot about how one would investigate a hacked system. You might also consider scanning your own networks (you can use the tools found in The Challenge) and see how many of your own systems are vulnerable.

And if you don't find any at all, that could be good or it could be bad. If you installed the patches, it is good. If the attacker installed the patches, well, you have a lot of work ahead of you.

using tcpdump and sanitize for system security

Why use freeware and open source for security management?

by Dario Forte

Dario Forte has been a security analyst since 1992. He is a frequent speaker and writer on forensic investigation and information warfare/management, and has worked with several Italian governmental agencies. He is a member of CSI, USENIX, and SAGE.

<*dario.forte@inwind.it*>

Some time ago, I was invited, as a representative of an Italian governmental office, to give a speech on new issues in security management in a forum organized by an important American ISV (commercial, obviously). The only condition I put on it was to be able to speak of the positive aspects of the freeware and open source movement with respect to the ISV-oriented one, with particular reference to security.

My attention was focused particularly on the fundamental principle that there are full-spectrum security tools that are truly valid, and the nice thing about them, apart from the availability of the source code, is the complete lack of licensing costs.

Every system administrator has a favorite toolkit. He or she runs it from a central console (where possible) and, especially in small-to-medium networks, tries to keep an orientation towards public domain tools. Personally, I use a Linux-based environment made up of the tools I am going to describe below.

In the December 2000 issue of *;login:*, I wrote about Trinux, a light distribution of Linux, which shares a broader realm with other mini-UNIXes such as tomsrtbt, LEM, PicoBSD, and others.

Trinux is booted from a single floppy, loads the rest of its modules from a FAT/Ext2 partition, from other floppy disks, or from an HTTP/FTP server, and runs completely in RAM. One of the most important features is that Trinux contains a series of precompiled versions of security tools such as nmap, tcpdump, iptraf, and ntop. Furthermore, this distribution works by default with DHCP.

Tcpdump and Its Companions

Trinux includes a precompiled version of tcpdump, which was created as a network diagnostics tool for UNIX but has gone on to be used in a great variety of ways. The transactions that this tool intercepts are, practically speaking, all the IP, TCP, UDP, and ICMP packets. Without getting too deeply into this topic (check out <*http://www.tcpdump.org*>), we could say that it is a continuously evolving tool that has its sniffer aimed at an increasingly large number of protocols. For this very reason, the amount of packets intercepted is very often so high that only external tools can sift out data and information that are truly interesting from the security point of view.

SANITIZE

Sanitize is one of these data sifting tools. It is a collection of five Bourne shell scripts for reducing tcpdump traces in order to address security and privacy concerns by renumbering hosts and stripping out packet contents. Each script takes as input a tcpdump trace file and generates a reduced, ASCII file in fixed-column format to stdout. Here is a list of the scripts:

- sanitize-tcp – has the task of reducing all TCP packets
- sanitize-syn-fin – does the same reducing on TCP SYN/FIN/RST packets
- sanitize-udp – reduces UDP packets
- sanitize-encap – reduces encapsulated IP packets (usually MBone)
- sanitize-other – reduces any other type of packet

What is important to emphasize is that the performance of Sanitize (<*http://ita.ee.lbl.gov/html/contrib/sanitize.html*>) depends on the type of traffic it is handling. For example, reduced TCP traffic retains the packet size (amount of user data), while other reduced traffic does not. In addition to Bourne shell, the scripts were written using tcpdump, and the common UNIX utilities sed and awk. Regarding the latter, it is a good idea to use the most recent versions.

Unfortunately, Sanitize also has its limits, albeit fewer than its brethren. For example, the contents of the sniffed packets are stripped out, while their size is revealed only for TCP traffic. For encapsulated IP traffic (usually MBone), and for non-TCP, non-UDP, non-encapsulated-IP traffic, only timestamps are generated. The script for reducing TCP SYN/FIN/RST packets is separate from the one for reducing all TCP packets, so the host renumbering performed by each will be independent.

Sanitize in Detail

The five scripts carry out a renumbering of hosts and the extrapolation of the packet contents.

The sanitize-tcp script works on TCP traffic and generates output in six columns:

timestamp of packet arrival

For the first packet in the trace, this is the raw tcpdump timestamp. For the remaining packets, this is the offset from the integer part of that first timestamp. There is a difference between what this script does and what sanitize-syn-fin does. The latter uses as its base time the arrival of the first TCP packet in the file, not the first TCP SYN/FIN/RST packet (this helps when comparing sanitize-syn-fin times with those produced by sanitize-tcp).

(renumbered) source host
(renumbered) destination host

When you use this product you will realize that this renumbering process causes the loss of all the other network information.

source TCP port
destination TCP port

These are the number of data bytes in the packet, or 0 if none (this can happen for packets that only lack data sent by the other side).

The sanitize-syn-fin script reduces TCP SYN/FIN/RST traffic for analysis. Its output is eight columns.

The first five correspond to the same columns as for sanitize-tcp, using the same host renumbering. The remaining three columns are:

TCP flags (e.g., "FP" for a packet with FIN and PSH set)
sequence number
acknowledgement sequence number

For the initial SYN sent to set up a connection, this will be zero. Experience has shown that you should not trust the sequence numbers used in RST packets.

The sanitize-udp script reduces UDP traffic. Output comprises five columns, corresponding to the first five columns for sanitize-tcp (i.e., packet size is not reported).

The sanitize-encap script reduces encapsulated IP packets (these usually are MBone packets). Output is a single column, giving the arrival timestamps.

Finally, sanitize-other analyzes all non-TCP, non-UDP, non-encapsulated traffic. Only a timestamp is reported.

As you can see, there aren't a lot of scripts but they are good ones. Thanks to its extreme granularity, tcpdump contains a great deal of information, which is not always easy to organize. Sanitize may thus be an excellent aid.

A Series of Questions

Can Trinux contain all the tools we've talked about? This is one of the most recurrent questions, partially driven by the fact that the community of Trinux users is rapidly growing. In an email exchange with the maintainer of the project (Matthew Franz), it was concluded that there shouldn't be problems here, especially in light of the heft (5Kb) of the Sanitize package. Nevertheless, whether the sed/grep in BusyBox supports everything in the scripts and whether it will be necessary to add egrep and awk still needs to be seen.

Another question concerns the compatibility of Sanitize with the various versions of tcpdump. According to Vern Paxon (Sanitize's creator), it should be compatible, except perhaps for very old versions of tcpdump (or unofficial releases that have altered its output format).

Conclusions

One hope would be the creation of a management console (obviously freeware/open source) capable of handling a number of installations of the tools discussed here. In the case of Sanitize, this requires script execution with maximum granularity. This might be an interesting idea for a new project. In the meantime, I will be content to use this toolkit in a test environment made up of a small LAN with 30 stations, four hosts, all *nix, hooked up to the public network via an auxiliary internal gateway.

MORE TOOLS

The following are other tools that could be used with tcpdump. Obviously, such tools have their limits, which is why I suggest using them together.

- Tracelook is a Tcl/Tk program for graphically viewing the contents of trace files created using the -w argument to tcpdump. Its latest release is from 1995.
- TCP-Reduce is a collection of Bourne shell scripts for reducing tcpdump traces to one-line summaries of each TCP connection present in the trace. This tool was also written by Vern, but it is less powerful than Sanitize (as I see it, of course).
- Tcpdpriv is a program for eliminating confidential information from packets collected on a network interface (or from trace files created using the -w argument to tcpdump).

ISPadmin

by Robert Haskins

Robert Haskins is currently employed by WorldNET, an ISP based in Norwood, MA. After many years of saying he wouldn't work for a telephone company, he is now affiliated with one.

<rhaskins@usenix.org>

Billing and Provisioning

Introduction

In this installment of ISPadmin, I examine aspects of billing and provisioning in an ISP environment. ISP billing is the process by which the customer pays for his/her service, not unlike billing in other service industries such as electrical or cable TV service. As such, the challenges in the ISP environment with respect to billing are:

- Defining bill plans
- Accumulating usage
- Generating bills
- Posting payments
- Interfacing into the provisioning process

ISP provisioning, while a separate process from billing, is closely related to it. Provisioning is the process by which services are enabled on the back-end systems which provide the actual services to the end customer. For example, when a customer orders a generic ISP dialup account with a POP box and home page, the following actions must be taken:

- An email account is created on the POP mail server, and other related systems are updated as necessary (e.g., mail relay(s)).
- The dial-up RADIUS (PPP) account is created.
- A Web home page account is created (usually of the form *<http://www.isp.net/~username>*).
- Access to the Web home page account is enabled, and optionally, an anonymous FTP area is created.
- Disk quotas are created on every system with customer data.
- A record is created in the billing system which indicates the master account owner, billing information for the master account, associated services billed to the master account, and bill plans for those services.

As you can see, this is not a straightforward process! Provisioning and billing are quite tailored to the ISP business, but billing is less specific to the ISP industry. While people can and do utilize generic billing systems for ISPs, the best results are achieved with a billing system that has been designed expressly for service providers, and especially those written specifically for ISPs.

One challenge faced when considering an ISP billing/provisioning system is the fact that such systems touch on most if not all areas of an ISP's operations, including finance, customer service, technical support, back-end business support systems, etc. In this installment, I will focus on only the billing and provisioning aspects.

Small Provider Goals

A small provider is primarily concerned about cost when it comes to designing and implementing a billing/provisioning system. This means a typical small ISP:

- Utilizes a home-grown or low-cost billing system, without high-end features like realtime billing data or realtime provisioning capability;
- Performs account adds/changes by hand on each back-end system or has written their own limited function non-realtime provisioning system in-house (which may or may not be integrated with the in-house billing system).

A small provider doesn't have the volume of account data that a large provider has, so it is feasible to do many processes via MS Excel spreadsheets (or pencil and paper for that matter) and manual input. The same also goes for billing, where there might be no need for the billing system to input RADIUS usage data in order to produce bills. (Whether or not to utilize RADIUS data is determined largely by what business model the ISP follows.) Not utilizing RADIUS accounting records makes things much easier for the smaller provider who doesn't bill for usage.

Of course, accuracy is very important! No one is going to stay in business if they bill customers incorrectly or aren't billing their customers at all. Another consideration is what happens when an attacker adds an account to a system under automated provisioning control. The account will get deleted the next time the provisioning process runs. Of course, if the hacker breaks into the provisioning system, and there is no way of comparing the accounts which are being billed against the accounts that have been enabled for services, a company won't be in business long. For any system administrator, but especially at ISPs, audit trails and mechanisms are a great asset operationally as well as from a financial control standpoint.

Large Provider Goals

A larger provider who services retail customers has different goals than a smaller provider when it comes to billing and provisioning:

- Functionality (e.g., virtual ISP [VISP] and realtime billing support)
- Flexibility (how easy and expensive is it to maintain and change over time)
- Reporting
- Low cost (but much less sensitive than a smaller provider)

Bigger ISPs are much more likely to purchase one of the many commercial billing and provisioning solutions. The only exception to this could be a large wholesale dialup ISP, which might be able to get away with a simple billing/provisioning system with the following limitations:

- No VISP offering, therefore no need to provide end-user services (such as email, RADIUS authentication, and Web hosting) and/or bill end users
- RADIUS authentication via proxy to downstream customers

Of course, not having a fully functional billing system may cause limitations on the growth of your ISP business (as someone once called it, "revenue limiting"). But it can be (and is currently) done this way at some ISPs.

Small Provider Design

Figure 1 contains a diagram that outlines the major pieces in a small dialup ISP's billing and provisioning process, and systems that provide services to end customers. This diagram is illustrative of a small provider, with an automated provisioning/billing system. However, the basis of the design is valid for larger systems, with appropriate scaling mechanisms.

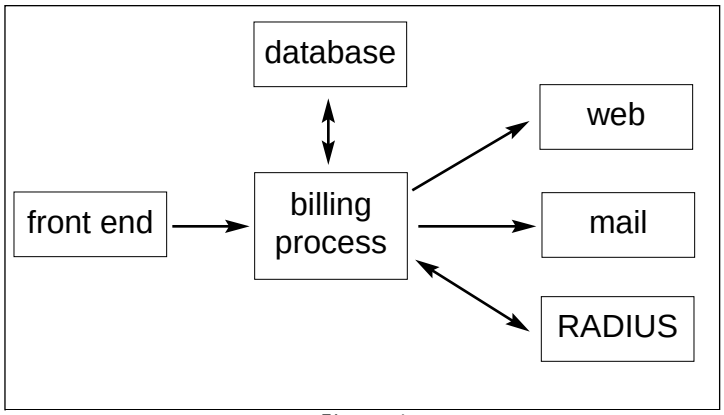

Figure 1

Of course, the center of the system is the billing/provisioning system, marked "billing process" in Figure 1. For a simple system, this would be a single machine. In the case of a commercial application like Portal, this usually takes the shape of a series of machines. In either case, they both

would communicate with an external Oracle, MS SQL, or similar database for a back-end data store. The flow of data would be two-way between the database and billing system.

The box labeled "front end" could be a script local to the billing machine or Web site that inputs data into the provisioning/billing system. It might be on a separate stand-alone system or integrated on the billing platform itself. This front end may also include a Web registration system, bulk-account registration system, specific customer-support screens, and/or other functionality.

The boxes to the right — labeled "web," "mail," and "RADIUS" (flow heading towards the box) — indicate the provisioning output aspect of the system. This is the billing system creating accounts on the back-end systems so customers can access their Web sites and email, and dial into the Internet. The "RADIUS" box with flow heading out of the box shows RADIUS accounting data flowing into the billing system in order to calculate end-subscriber usage.

Large Provider Design

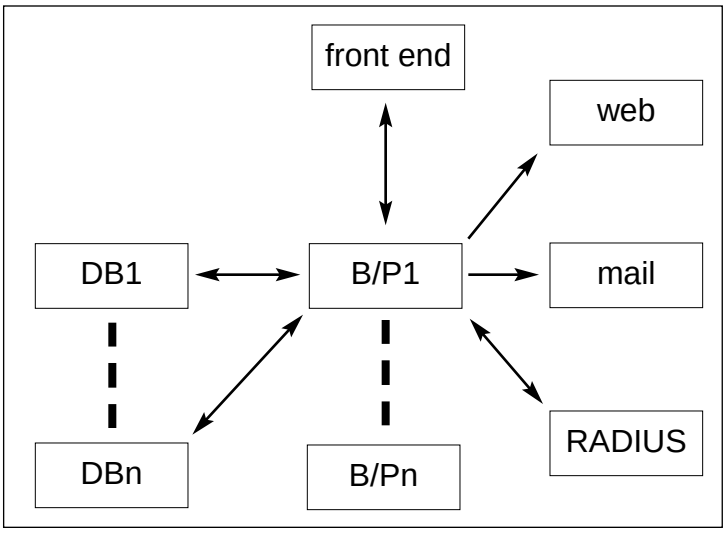

Figure 2

Figure 2 contains an illustration of a possible design of a commercial billing system such as Portal Software's Infranet. Please note that there are many, many ways to implement commercial billing systems, and this discussion is meant as a high-level overview of what is possible. If you are really interested in this topic, I would suggest contacting a major ISP billing company and talking to one of their sales engineers.

The essential difference between a small provider and large commercial implementation is the ability to scale the database and billing/provisioning server machines. In Figure 2, the boxes marked "DB1" and "DBn" indicate multiple machines housing the data store. Of course, large database implementations such as Oracle scale relatively easily. Boxes marked "B/P1" and "B/Pn" indicate the multiple billing/provisioning machines that can be utilized for performance and scaling under heavy-load and high user counts.

Lightweight Directory Access Protocol (LDAP)

No discussion pertaining to provisioning would be complete without at least a mention of LDAP. For those of you who have read previous installments of ISPadmin, you know that LDAP eliminates much of the work when it comes to provisioning. LDAP is a central repository for authentication and account configuration (e.g., what host a customer's POP mail is on, the spool directory location, etc.). The issue with LDAP has been the lack of support at the application level. However, this is slowly changing over time. Full application support of LDAP remains the "holy grail" for many people, both in the ISP business and, to some degree, within the entire IT arena.

Open Source Solutions

A search of Freshmeat and/or SourceForge with the phrase "ISP billing" shows numerous hits. However, many of the listed programs are vaporware, little more than concepts. Some are specific to Web hosting or not related to ISP billing at all. Here is a list of a few open source dialup ISP billing and/or provisioning systems with available source code:

- Freeside from Silicon Interactive Software Design, Inc.
- ISPman from Atif Ghaffar
- gcdb
- ISFree from Brian Wolfe
- ISP daemon

While I don't have space to discuss them all, I will briefly cover Freeside and ISPman.

FREESIDE

Of the open source billing/provisioning applications, Freeside has been around the longest and is probably the most feature-rich and stable. It is a complete billing and provisioning program written in Perl. Freeside has the following features (from the Freeside Web site):

- Utilizes Perl's DBI module; recommended database back ends are PostgreSQL and MySQL
- Web-based interface
- Reseller/agent support, including controlling what agents can sell
- Tax rates by state and locale
- Exports to a multitude of UNIX file formats, including:
  - passwd and shadow (or master.passwd) files
  - ERPCD acp_passwd and acp_dialup files
  - RADIUS users files
- Works with ICRADIUS and Radiator RADIUS servers for native SQL authentication
- Virtual domain support:
  - Exports Sendmail virtusertable and sendmail.cw files
  - Exports Qmail virtualdomains, rcpthosts, and recipientmap files
- Signup server support with MS IE auto-configuration support
- Realtime credit card support

Freeside would be a good choice for an ISP on a limited budget. One major limitation is the lack of LDAP support. Also, scalability might be an issue, unless Oracle or another easily scalable back-end database is utilized.

ISPMAN

ISPman is a relative newcomer to the open source scene and does not have a billing component. It is strictly for provisioning accounts, with LDAP as a back end. It is still under active development and therefore may not be suitable for your needs. However, it is worth investigating, at least as a starting point for your own development. In fact, I firmly believe that the future for ISP provisioning lies with an end-to-end LDAP solution, and Atif Ghaffar (the author of ISPman) is off to a great start. ISPman features include:

- Full LDAP support; all back-end apps are LDAP aware
- Creates DNS configuration files to input into a BIND 8 server
- Web interface for all input
- End-subscriber management

Required back-end software to run ISPman includes:

- OpenLDAP
- Postfix
- Cyrus imapd and SASL

The trend at the higher end of the billing/provisioning market is towards convergence.

- Apache
- Proftpd (GNU's FTP server)
- BIND
- pam_ldap
- IMP 2.2 (PHP Webmail application)

Commercial Billing/Provisioning Applications

There are a number of commercial billing/provisioning applications on the market. A complete discussion of all of the options is out of the scope of this article, but a brief discussion is worthwhile.

At the lower price range, there are NT-based solutions which focus exclusively on ISPs and don't have features like realtime accounting or complete support for wireless, DSL, or other such enhanced services. A few of the better known players in this space are:

- Rodopi
- Boardtown Platypus

At the more expensive end of the spectrum there are a number of players; here are a few examples:

- Portal Infranet
- Amdocs/Solect
- Lucent/Kenan

The trend at the higher end of the billing/provisioning market is towards convergence. This is where telcos and service providers can utilize one platform for billing and provisioning all of the services they offer, including wireless, wireline, ISP/ASP, and others. Historically, telcos and larger service providers have implemented provisioning/billing platforms for each line of their business, which leads to high maintenance costs. (If you investigate this area, you will find that Lucent and other players who came out of the telco market call billing "Business Support Systems," or BSS, and provisioning "Operational Support Systems," or OSS.)

There is an effort to open the (historically closed) interface between providing end-subscriber services and billing for those services by defining an open protocol. IPDR.org, a consortium of leading billing/provisioning software vendors, is developing this standard. CLEC-Planet also has a short article which talks about the current status of billing/provisioning convergence.

Outsourcing

No discussion of ISP billing would be complete without touching on outsourcing. There are several companies who will, for a monthly fee, do your billing, customer care, provisioning, etc. for you. This may be an option for a larger ISP, as a smaller provider usually wouldn't have enough subscribers to make this a viable option.

Typically, these billing companies have implemented Portal, Kenan, or other such commercial billing system in a "virtual" manner. Then, the outsourcing company will customize their system to accommodate you. They typically sell "a la carte" and will provide just the parts you need (only billing, billing plus provisioning, etc.). GlarNet is a company that provides an outsourced ISP billing and provisioning solution, among other services.

Conclusion

Billing and provisioning is a very important aspect of an ISP's operation. For a smaller ISP, billing and provisioning by hand is acceptable. But for a medium or large ISP,

billing and provisioning systems usually are automated, either by building a system in-house, deploying an open source solution like Freeside, or implementing a commercial billing system like Rodopi or Portal. A third option is to utilize an ISP billing outsourcing company and pay a monthly service fee.

Next time, I will look at ways ISPs design and manage their Usenet news infrastructure. In the meantime, please send your feedback on ISP topics and system administration to me!

References

Oracle: <*http://www.oracle.com*>
Portal Software: <*http://www.portal.com*>
LDAP starting point: <*http://www.ldapman.org*>
Freshmeat: <*http://freshmeat.net/*>
Sourceforge: <*http://sourceforge.net/*>
Freeside: <*http://www.sisd.com/freeside/*>
ISPman: <*http://ispman.sourceforge.net/ispman.php3*>
ISPman article by Atif Ghaffar (ISPman author):
<*http://www.linuxfocus.org/English/September2000/article173.shtml*>
gcdb: <*http://sourceforge.net/projects/gcdb/*>
ISFree: <*http://advogato.org/proj/ISFree/*>
ISP daemon: <*http://ispd.eburg.com/*>
Perl DBI: <*http://dbi.symbolstone.org/index.html*>
PostgreSQL: <*http://www.postgresql.org/index.html*>
MySQL: <*http://www.mysql.com/*>
ICRADIUS: <*ftp://ftp.cheapnet.net/pub/icradius/*>
Radiator: <*http://www.open.com.au/radiator/*>
Sendmail: <*http://www.sendmail.org/*>
Qmail: <*http://www.qmail.org/*>
MS Internet Explorer Administration Kit (IEAK):
<*http://www.microsoft.com/windows/ieak/en/default.asp*>
OpenLDAP: <*http://www.openldap.org/*>
Postfix: <*http://www.postfix.org/*>
CMU's Cyrus IMAP server: <*http://asg.Web.cmu.edu/cyrus/imapd/*>
CMU's SASL library: <*http://asg2.Web.cmu.edu/sasl/*>
Apache: <*http://httpd.apache.org/*>
ProFTPD: <*http://proftpd.org/*>
ISC's BIND: <*http://www.isc.org/products/BIND/*>
pam_ldap: <*http://www.padl.com/pam_ldap.html*>
IMP: <*http://www.horde.org/imp/*>
Rodopi: <*http://www.rodopi.com/*>
Boardtown's Platypus: <*http://www.boardtown.com/*>
Solect from Amdocs: <*http://www.solect.com/*>
Amdocs: <*http://www.amdocs.com/*>
Lucent: <*http://www.lucent.com/software/*>
IPDR.Org: <*http://www.ipdr.org/index.htm*>
CLEC billing convergence article: <*http://www.clec-planet.com/tech/0004phifer.htm*>
GlarNet: <*http://www.glarnet.com*>

designing an external infrastructure

by John Sellens

John Sellens is the general manager for Certainty Solutions (formerly GNAC) in Canada, based in Toronto, and is proud to be husband to one and father to two.

<jsellens@certaintysolutions.com>

This article discusses some of the things that you should consider when you are designing the systems infrastructure for an "external site."

What do I mean by an "external site"? An external site is a computing installation that is intended primarily to provide services for people outside your organization, or which is located at a physically remote location. The most common example of an external site is, of course, a Web server (or servers) located at a co-location facility, but there are other kinds of sites that are "external," such as the point of sale and store-management systems used in a retail store chain, the systems used to transfer purchase orders to suppliers, or an incoming fax server. Most of this discussion is going to be phrased in the context of a Web-serving infrastructure, but much of the discussion will be applicable to other kinds of external sites.

Introduction

Before I start talking about specifics, let's spend some time on a slightly more abstract discussion.

Characterization

External sites usually have some or all of the following attributes:

- Remote location – typically located in a co-location facility of some form
- External users – usually primarily used by customers (or potential customers) or business partners
- 7x24 operation – expected to be "always" available
- Flexibility – needs to be able to cope with changing demands or traffic levels

Internal sites, by contrast, often have fewer demands placed on them. The service loads tend to be more predictable, off-hours downtime is easier to schedule, users are often easier to notify, and the equipment is more likely to be just down the hall, rather than across town or across the country.

Basic Concepts

In this discussion, I'll make use of the following concepts:

Redundancy: The use of multiple instances of particular components to reduce the impact of hardware failure. The most common example is using a pair of disks in a mirrored configuration to guard against disk failures.

Replication: Similar to redundancy, but refers more to the duplication of services than to the server hardware components themselves. The use of multiple SMTP or DNS servers is a common example of replication.

Separation of Functions: The use of separate servers for different services. This is also the "don't put all your eggs in one basket" concept.

Reliability: The ability of a system to cope effectively with failures or unusual conditions.

Accessibility: The ability to gain the appropriate access to your systems, even in times of service, system, or component failure. For example, a modem and a telephone line can be very useful when your regular network connection is dead.

Recovery: The process of restoring an impaired or failed service or system to its normal state.

Security: The appropriate access and other controls that protect your systems and services from attacks (intentional or unintentional). Security in this context is more like a topic than a concept.

I will cover how these concepts can be applied to design an appropriate external infrastructure in order to reduce the risk of site failure, and to shorten the expected time to repair a failure should one occur.

It's worthwhile to note that all of these concepts (and more) can also be applied in the design of internal sites and will result in a better infrastructure. But the nature of external sites, most notably their external visibility and often remote location, makes these concepts especially relevant for external sites.

Definitions

I've already defined what I mean by an "external site." Here are a few more definitions:

Service: The actual facility or process provided by your site. We most often think of services like Web content, mail, or FTP, but other common services include such things as calendaring, product catalogs, file storage, and so on.

Server: Most often a computer, used to provide a service, or a part of a service.

Component: A device (typically) that provides some function or ability. This includes such things as disks, network equipment, power bars, and yes, servers.

System: The collection of components that work in concert to provide a service.

Some of the definitions attempted here make some subtle or slight (or even somewhat obtuse and obscure) distinctions, but I thought it would be useful to attempt to distinguish between the four different terms.

Balance

Effective system and infrastructure design often involves a number of tradeoffs – the time-honored "cheap, fast, good – choose two" has a certain ring of truth. In external infrastructure design, we're often faced with conflicts between the three following goals:

Simplicity – Cost effectiveness – Reliability

In other words, adding reliability to a site usually adds additional complexity and cost.

Adding reliability to a site usually adds additional complexity and cost.

In most cases, we design site infrastructures to provide an appropriate level of reliability, while keeping an appropriate balance between cost and the risks and consequences of site failure. What is "appropriate" will depend on your specific situation – your budget, peace of mind, and level of aversion to public relations problems all enter into the appropriateness equation.

Infrastructure Elements

Most external sites are designed for a particular purpose, making use of particular components, software applications, and size and design criteria. But no matter what the final overall design, a site usually contains some (or all) of the following types of components:

Computing Systems: They come in different sizes and different complexities, and they run different operating systems. But they're all intended to run some form of software that performs a function or provides a service.

Storage Disks: Disks, tape, optical, etc.

Redundancy is the primary tool for guarding against server hardware failure.

Networking: Usually some form of hub or switch, host network interfaces, and an uplink to the Internet or a wide area network.

Firewall or Filtering Gateway Router: A barrier of some kind to limit what forms of access to the site are allowed.

Load Balancing: Larger Web sites almost always utilize some form of load balancing to share the service load across multiple servers. Load balancing can be provided through software, DNS-based mechanisms, or by load-balancing hardware.

Support Elements: The power bars, uninterruptible power supplies, console servers, modems, monitoring and environmental devices, etc.

Applying the Concepts

Redundancy

Redundancy is the primary tool for guarding against server hardware failure. The idea is simple – if you've got two (or more) parts performing a single function, things will (probably) keep working if one of the parts breaks.

It wasn't very long ago when redundant parts were relatively expensive, and redundancy was used only in high-end sites. Nowadays, with commodity components, redundant components should be included in just about every server.

The most common application of redundancy is with disk drives, where two or more drives are configured for mirroring (RAID 1) or parity striping (RAID 5). RAID configurations allow systems to keep functioning with no data loss even if a disk fails. With disks as cheap as they are these days, and with RAID software included with just about every operating system, you should almost always configure your servers to use some form of RAID configuration.

Other components that are often configured for redundancy are: power supplies, CPUs – you didn't think dual processor systems were just for added speed, did you? – memory boards, and network interfaces. Most servers these days can be easily configured with redundant components for most of the critical parts.

Redundancy is a great technique and can be cost-effectively applied in just about every situation, regardless of your application or environment.

Replication

Replication is used for two purposes: resilience of the service/system in the face of server failure and scalability of the service. In external sites, the most common example of replication is in the use of multiple "identical" Web servers, with the load shared across an entire "server farm."

If your application or service can be built (or configured) to work across multiple services, you'll end up with a much more reliable and scalable system; a server farm of 10 (or 100) Web servers can cope much more effectively with a server failure (due to OS bugs, catastrophic hardware failures, etc.) than a single monolithic server, and provides an obvious method for upgrading overall system capacity.

However, replication can't be achieved simply by plugging in another server (obviously); your application and system need to have been designed and/or configured to be able to make use of replicated servers. For example, multiple Web servers won't do much good if your Web site's address is assigned to just one of the boxes.

Server replication can usually only be useful in concert with some external help. Replication is sometimes available in services through the application or its interfaces. For example, some "middleware" software programs provide Web server plug-ins or other interfaces that select one server in a replicated pool from a list of currently available servers (e.g., Apple's WebObjects software).

In most cases, however, some form of external load balancing is used. The most common examples of external load balancing are "round-robin" DNS configurations or load-balancing hardware devices.

Round robin DNS is the simplest (and cheapest) method of load balancing, but it has a number of drawbacks. It is implemented by defining a service name with a number of IP addresses, one for each server in your server farm. DNS servers will provide the list of IP addresses in response to lookups, but will rotate through the list of addresses, putting each different address at the head of the list in turn. (At the time of writing, `nslookup www.microsoft.com` provides an example of the use of round-robin DNS.) The primary drawback of round-robin DNS is in times of server failure, when some percentage of users will be given the IP address of the failed server as the first one to try, and will either fail to connect or get a delayed response while the initial connection to the failed server times out. It also performs poorly when the server maintains some form of "session state" between connections; if you connect to a different server next time, it may not have the correct context to continue your service.

These problems can be addressed by using intelligent load-balancing appliances, such as those made by Alteon, F5 Labs, and others. These are devices that (typically) look like an Ethernet switch, but which "re-write" IP addresses in packets passing through them to load-balance traffic across a group of servers. They do this by performing periodic "health checks" on the servers in their pool for availability, response time, etc., and by maintaining some form of state table that matches the two ends of a load-balanced connection. A real discussion of load balancing is beyond the scope of this article, but suffice it to say that these devices can be quite sophisticated, powerful, and very effective in keeping services running and available.

One final aspect of replication that needs to be considered is that of data or content distribution, i.e., keeping everything consistent across an entire server farm. This can be a complicated problem, with lots of revision control and timing issues, but can usually be handled effectively in software for services that use relatively static data, using tools such as `rdist`, `rsync`, `CVS`, or various commercial software packages. Replicating database servers is usually a much more complicated exercise, especially when database updates are driven by external sources (such as Web site customers submitting orders). Up to a certain size, database server replication can be handled by clustering or database replication by the database software itself, both of which can be quite intricate. But beyond that, things can get very complex, with large amounts of custom software keeping things consistent.

The underlying and as yet unspoken message is this: plan ahead for replicability and understand how the components of your application will interact when there's more than one of each.

Separation of Functions

As mentioned above, this is the "don't put all your eggs in one basket" concept. By this, I mean use separate servers for each service or function – don't make your Web server, application server, and database server all the same box. There are several reasons why

Don't make your Web server, application server, and database server all the same box.

A reliable system is one that is designed to be effectively maintainable, resilient in the face of change or failure, and quickly recoverable should something go wrong.

this is a good idea. The most obvious reason is that when a server fails, you only have one service impacted, rather than two (or more!). It also makes troubleshooting much easier if you don't have to worry about the possible interactions of two competing applications on the same server. Finally, separating functions makes capacity planning and upgrades much simpler (in most cases). Given the wide range of server capacities (and prices) these days, it's almost always possible to cost-effectively separate your services onto separate servers.

Reliability

Reliability in this context is a grab bag of things to consider in the context of the overall system design, implementation, and ongoing operation.

A reliable system is one that is designed to be effectively maintainable, resilient in the face of change or failure, and quickly recoverable should something go wrong. Reliability encompasses a wide range of "best practices": proper planning, considered design, effective documentation, and consistent execution.

I won't say more than that about reliability here; I will instead refer you to my (upcoming) SAGE booklet, *System and Network Administration for Higher Reliability*.

Accessibility

For external sites, often located where you aren't, accessibility is a much more important issue than it is for a server that's just down the hall (or under your desk).

By "accessibility" I mean the ability to get appropriate access to the components of your site both during normal operation and during times of failure. The most obvious example of "access" is some form of network login (e.g., telnet, SSH), but access also includes "out of band" access for when your firewall or router is confused, or when your network connectivity is absent; console access when a system is at the "boot" prompt or has crashed; access to the "big red" (power) switch when necessary; and access to the hardware when it's necessary to replace a failed component. Let's look at each in turn.

Network Login: This is most commonly "just software." Most UNIX folks are familiar with SSH these days (and if you aren't, you absolutely should be), which provides secure remote logins, file copies, and command execution (among other interesting things). Other approaches to network login include plain old telnet, and "remote control" applications such as VNC or PCAnywhere. These applications allow access for normal everyday maintenance activities, as well as emergency repair in situations where most components are still functioning.

Out of Band: When your network access is broken, having an alternate path into the internal network of your external site can be a blessing. Anyone who has ever changed the configuration of a remote router or firewall has likely (or perhaps, hopefully) understood the possible complications of a "slip of the finger."

Out-of-band access is most commonly implemented with some sort of dialup connection, usually a modem on a serial port of a computer or router, but it can also be implemented in other, more complicated ways. Make sure that when you're planning your site, you consider what will happen when your primary connectivity fails.

Console Access Network: login works just fine as long as a system is running normally. When it crashes and is sitting in single-user mode, or it's at the boot prompt, or the BIOS is waiting with the message "keyboard missing, press F1 to continue," you'll need some form of console access. Some servers and devices have serial console ports, which

can be connected to a modem, terminal server, or the serial port of another device. Other servers require a keyboard and display in order to deal with some problems. The latter typically require a remotely accessible KVM (keyboard, video, mouse) switch, or someone onsite to deal with the problem.

Power Control: A number of companies make remotely controllable power bars that you can connect to via serial connection, telnet, SNMP, or Web browser, and that allow you to turn devices on and off remotely when they get completely wedged. Some of these devices even have environmental monitoring built in, so you'll be able to tell when your air-conditioning has failed (or your server is on fire). Great tools, very useful, and usually well worth the investment.

Remote Hands: Depending on the problem, and where your system is located, having the (pre-arranged) ability to call someone at the remote site and tell them which button to push or cable to swap can save a lot of aggravation. Many co-location facilities offer this service, as do a number of third-party service companies. The key here is "pre-arranged" — you've got to make sure that everything is in place before you need it. And that's not just having someone's name in your phone list; it's having an agreement, effective contact information, and proper documentation of both the procedures and the configuration of your site.

Recovery

When things are broken, it's no time to start wondering how your site was built and configured, and what options were chosen when.

There are two primary considerations to recovery: having a well-defined (and well-rehearsed) process for restoring the system to a fully operational state and the ability to do just that on a timely basis. Consider the following: using standard mechanisms for configuring servers, such as Sun's "jumpstart" mechanism, to ensure a fast and consistent recovery process; having effective vendor support contracts in place, so that repairs can be completed within an appropriate time frame; keeping your documentation complete and up-to-date, with rack elevations, network diagrams, copies of configuration files, disk partition information, equipment model and serial numbers, and anything else you can think of; and having an effective backup (and restore) process.

Security

Security is a topic that is far too involved to be covered effectively here, so I will say only this: make sure that you have both an effective security policy and an effective security implementation that together provide the appropriate balance between the risk of a security exposure and the overall effectiveness of your system.

Closing

Designing an effective external site infrastructure can be a very complex process of trying to balance conflicting needs and priorities while delivering a working and cost-effective system. The discussion here has not covered every possible alternative or every possible problem area, but I hope that it has given you some things to think about and will help you implement systems that don't keep you up at night.

When things are broken, it's no time to start wondering how your site was built and configured, and what options were chosen when.

the loneliness of the long-distance sysadmin, or where geeks gather

by Mike Knell

Mike Knell is a system administrator in the Department of Computer Science at Trinity College, Dublin. He likes loud music, long walks, and UNIX.

<*username@vanitydomain.org*>

I'm a sysadmin. This, as I'm sure many people reading this will be pleased to hear, is a cool job. Sure, it's a job that drives you nuts at times, but there are still the perks: no management interference, the ability to delete user files and kill user processes with impunity, all the training you can eat, a social life that anyone else can only dream of having . . . .

Uh, one moment. That was the dream I had last night. Sorry.

I'm going to hold on to that assertion about system administration being a cool job, though, because compared to a hell of a lot of other jobs, it is. But I'm also going to talk about a couple of other things that, if paid attention to, would help make it even cooler.

A few months ago, a sysadmin friend by the name of Dónal Cunningham made an astonishing announcement – he was going to LISA. We all thought that was very cool, and I promise that I won't use the word "cool" again after this because it's getting a bit tired now. At this point, I bet a few people are thinking "Huh? Why is going to LISA astonishing?"

The reason for this is simple. I live and work in Dublin, Ireland. It's a nice place to live, but there's one drawback as far as professional development for the system administrator is concerned – distance. Getting almost anywhere from here involves a flight to somewhere, or a ferry trip to the UK followed by a trip across the UK, followed by another ferry trip to . . . well, you get my drift. Going anywhere outside the country, in short, starts getting really expensive really fast. The comparatively small size of the sysadmin population here means that people often don't know that there are other sysadmins out there. There are a few OS-specific user groups, or informal gatherings of people that have sprung up in various places. But there's nothing for, say, the new sysadmin, or even as in my case a couple of years ago, the new-to-the-area sysadmin, to point to and say, "That's where the sysadmins hang out."

The result of this is that wheels end up getting reinvented quite regularly. One person has a problem, and they never get to sit down with other people in the pub or at a meeting and say, "I've had this problem. Anyone know anything about it?" So they have to work it out for themselves from scratch, even when it's quite possible that someone working just down the road from them has had the same problem and would have been able to answer in about five seconds what will otherwise take a couple of days of tedious slogging to work out. In summary, what's lacking is information exchange.

But why am I moaning here about an issue that's specific to me? The simple answer is that it's not. I'm willing to bet that all over the world there are thousands and thousands of sysadmins who aren't SAGE members, who may never get to hear about SAGE, and who will never attend LISA or any other formal system administration conference. It's even quite unlikely that many sysadmins will get to talk informally with other sysadmins from outside their own place of work. There are a few reasons for this – limited travel and training budgets, limited publicity for those groups that actually *are* out there, and

possibly even lack of motivation to go out and find out who else is doing a similar job, but the most pervasive problem is a shortage of local communication.

For a profession that spends much of its time dealing with the exchange of information in one form or another, sysadmins in many parts of the world are remarkably poor at exchanging information among themselves. For every problem or obstacle that's encountered while trying to get something done at work, there's almost certainly a solution out there already.[1] The problem is getting hold of that solution.

Vendor-specific certifications are one way of finding out how to solve problems. However, as soon as the problem is something remotely out of the ordinary, that expensive MCSE or CCNA suddenly becomes less useful. Such certifications don't really contribute much to professional development unless you equate "professional development" with "maximum obtainable salary." I'd prefer professional development to be more about gaining a reputation as a Damn Good Sysadmin than as an Expensive Sysadmin. Don't get me wrong – certifications have their value, but what's most important to me in the process of becoming a better sysadmin is good old-fashioned peer-to-peer networking.

To my mind, the best and usually the cheapest way to become a better sysadmin is to talk to other sysadmins. Face-to-face conversation is by far the most valuable, as it's possible to learn a lot just by listening to things that you never even knew you needed to know (and you often get to drink beer while you're doing it). But there are plenty of other ways in which people could be communicating that are currently either underused or well used but unpublicized.

Most sysadmins these days are fortunate enough to have an Internet connection, and there are numerous services available out in the electronic world that could be used for connecting sysadmins. There's IRC for those questions that need to be answered right now but not necessarily reliably, Usenet if you can wait a little longer and still not have the right answer, the Web if you can find what you're looking for in the search engines, and mailing lists if you know which one to look for.

However, when it comes to the crunch there's little to beat local knowledge and contacts even when all the resources of the Global Village are available at your fingertips. The idle "Hey, has anyone encountered . . ." question asked at a meeting or in the pub often draws a quicker and more accurate answer than posting on Usenet or asking on some anonymous IRC channel. If you know the person answering the question, you're more likely to be confident that they've given you the right answer (or not) than if the answer has come from a total stranger out on the Net.

Anyway, Dónal went to LISA last December and came back laden with enthusiasm, new knowledge, and some really nifty freebies. As often happens, he's passed on some of this new knowledge to me, some of the enthusiasm, and even a couple of the freebies.

As a result of this new-found enthusiasm, we're looking to improve the lack of communication here a little by finally getting around to starting a SAGE-IE, as an offshoot of the recently formed UK-based SAGE-WISE. We're hoping to have a press launch within the next few months, followed by a technical event of some kind to kick things off. This isn't just because we want more freebies (although I might have to actually buy some t-shirts soon otherwise) but because somewhere out there are bound to be sysadmins

For a profession that spends much of its time dealing with the exchange of information in one form or another, sysadmins in many parts of the world are remarkably poor at exchanging information among themselves.

1. Usually something along the lines of "Add 'host zing arf' to /etc/foo!" or "Use a crossover cable!" or "Shoot the user concerned!" I didn't say that it was always the right solution, but any solution is a start.

who are interested in becoming better sysadmins but who don't know how to go about it.

If every city had a local SAGE group or even just an informal gathering, and every country had a national SAGE, system administration as a profession would benefit in ways we can only begin to think of. It would be, uh, cool.

[The SAGE-IE mailing list is hosted by SAGE-WISE – to subscribe, send mail to <*ireland-subscribe@sage-wise.org*>. The SAGE-IE website can be found at <*http://www.sage-ie.org/*>].

customers organization employees or people accessing company systems via the Internet? If the answer is "both," you're not quite listening – who are the PRIMARY customers? Who gets priority? If the answer is still "both," and you are not a company of under 50 people, you are probably trying to staff an entire department with one position. Rethink your position.

Indicate up front if the position requires cell phone or pager access outside normal hours. If there isn't a formal on-call rotation, you might want to say that "while the position does not involve shift work, applicants should be aware that the company has a dynamic, fast-paced environment and will at times require support outside normal business hours." Of course, this is often shorthand for "we haven't codified our requirements into something firm enough to stand behind." Danger, Will Robinson!

Duties and Required Expertise

Don't describe the job duties with vague phrases like "manage infrastructure" or "ensure smooth operation of production environment." If you include such phrases, they should be followed immediately by clarifying statements describing the specific technologies and duties. For instance, "manage infrastructure of EMC, NetApp, and HP storage arrays." Even better, "monitor, design, and upgrade network of EMC, NetApp, and HP storage arrays to support critical financial modeling effort."

Giving an explicit SAGE Level in the job description will help set expectations appropriately. Since not everyone is familiar with the SAGE Levels, don't elide the job description to just that. Refer to the SAGE Level as part of a conventional description of required experience, stressing those parts which are most important to the position. Clearly distinguish what is required for the position from what would merely be nice to have. Your most qualified candidates are often those who are the most realistic about their skills; don't scare away good candidates by appearing to require The Perfect Candidate.

The Constraints

Contact Details

Do give a specific role or job-description email address to respond to and keep it "live" later, but don't make it a person's address. These things take on a life of their own; they are harvested on the Web, are saved by people to refer folks to later, are passed around between agencies as favors, etc. You'd like the address to be a good long-term one, so you can check periodically for folks contacting you, but you don't want it to be anybody's personal mailbox because you will get junk in it.

Do not rule out a candidate based on the format of their resume, but do suggest one or two preferred formats. If you make no specific requests, do not be surprised if you receive resumes in LaTex, nroff, PostScript, or other formats which your HR department may be ill-equipped to deal with. Rather than deep-sixing a candidate whose resume HR does not understand, make certain that the resume is sent to someone technical for parsing or that HR requests a plain-text resume from the candidate.

Money Matters

Do not say "salary commensurate with experience" if you have a specific salary cap that is not informed by current salary survey data from your geographical area and industry. Give a range instead, and indicate that salary will be keyed into that range based on experience.

Giving an explicit SAGE Level in the job description will help set expectations appropriately. Since not everyone is familiar with the SAGE Levels, don't elide the job description to just that.

Do include mention of any non-salary compensation accompanying the position, such as medical and vision benefits, day care, stock-option plans, and so on. If your organization provides domestic partner benefits, a special dental plan, and other benefits which are not customary, those can also be important enticements to mention to candidates.

The Fine Print

Do say "principals only" if you don't want agencies contacting you, i.e., would rather not pay 10% to 30% of salary as a finder's fee. If you are happy to work with agencies, get explicit documentation of their fees before they send you a single candidate. If you have your own HR staff, find out in advance the interaction of financial responsibility for outside agencies presenting candidates.

Do say "This is a full-time, salaried position only. Contract, hourly, or part-time applications will not be considered" if that is the case – or vice versa. If your intent is to do a contract-to-permanent position, do not advertise it solely as a contract position. Conversely, if you are seeking a contractor but are open to a possible retainment, use a standard phrase such as "Contract position only, but employee possibilities for the right person."

If you are publishing this directly, rather than handing it off to your HR department, remember to include whatever standard legal disclaimers are required or advisable. If you have specific requirements which are not specific to job duties, such as eligibility for a security clearance, or cannot accommodate certain situations, such as assistance with a visa or work permit, indicate those requirements clearly. I sincerely advise you to have your legal or HR department (or both) sign off on any requirements of this type before publishing them in any venue.

Summing Up

As you can see, the basic guidelines are simple. If you address each of the points above with a line or two of text, it will take a relatively small amount of editing to turn the results into an advertisement suitable for publication. Think of things that you would want to know about a position, and let that be your guide. Then run it past the lawyers just to make sure.

easy management

Giving Good Report, or I Keep Doing Work, Why Do They Keep Yelling at Me?

by Richard Threadgill

Richard Threadgill is the cofounder of Ponte Communications. He is also a decade-long sysadmin and project manager.

<richardt@ponte.com>

Lots of techies give really lousy progress reports and are really hard on their managers for no good reason. This is particularly a problem for sysadmin types, systems programmers, and other people who love math too much. I spent several hours with a coworker last week discussing "how not to be an employee of doom," and these are my notes from that conversation.

First, an aside: these notes offer advice both for techies in general, who often have a poor model for the pressures on, and motivations of, their management, and for systems and math people in particular. Math people have two classes of problems: trivial problems, which merely require identifying an existing solution; and unsolved problems, which require thinking, hypothesis, and, potentially, experimentation. This often leads them to front-load their work, going through the list of their tasks and performing the "hard" tasks first because the others are "just work."

Systems people have a strong tendency to suffer from "searchlight focus" as well, because it's a really useful trait in a high-interrupt environment where you need to context-switch pretty completely. Unfortunately, it leads to some work habits which make your behavior (our behavior) really unpredictable. Management can tell how often they're getting complained to about things you haven't gotten done, and how often you're reporting finishing tasks which they cared a lot about personally, but that's about all they know.

This unpredictability makes writing job-requirement justifications basically impossible. And that's bad because it means you get fewer raises and spend all of your time being overworked. It also means that development managers basically can't deal with you in any constructive way, because your behavior is inexplicable and unpredictable.

Finally, this article is most intended for people who have a soft handle on how much time they spend on tasks, because they think about tasks from the perspective of difficulty rather than from the perspective of expected time-to-accomplish.

So, some rules for being easier to manage.

First and foremost, NEVER go radio silent. This is your manager's worst nightmare: they don't know what you're doing, they can't defend spending their resources on it, and they don't know when you'll finish. So, if you are about to embark upon a task which might cause you to go quiet for a while, discuss it with your manager first. Be prepared for them to direct you to attack a different problem first, so that they (and you) can build some capital to defend you while you're silent. Think of this as giving your manager a good answer to the question "What has that employee done for you lately?" when they get asked by their peers and their management. This makes their life easier.

Give status early and often. This makes your manager's life easier. Most of the rest of this document will talk about how you can order your work and reporting to make your work more predictable and thereby more visibly valuable.

Attempt to show consistent levels of output. This creates a perception of predictability and changes the conversation your boss has with management from "Has Dave gone silent again? Do we know what he's working on this time?" to "How's the really huge

When beginning a project, make a list of tasks. Then make a list of questions which must be answered to perform those tasks, including who needs to answer those questions.

project we asked Dave to deal with coming? Are we interrupting him with too many other little tasks?"

Learn to report when you're overloaded. It's much, much better for your manager to know that you are not going to be able to accomplish something at request time than to discover it at the expected time-of-completion.

Order your tasks so that you generate usable, partial, visible results often, thus enabling other people to get leverage from your work quickly, and making your manager's life easier. This hurtles headlong into the typical math-geek work ordering model, which tends to start with "Do the hard bits, because we don't know if those are possible, and that's the most important thing to learn before we get into this too deeply." Unfortunately, this behavior gets interpreted by a lot of managers as "Dave just wants to do the fun bits and never the actual work." So while it will make your teeth itch, gang, when you do your task breakdown, plan to do a bunch of the simple ones in parallel with the hard, thinking-about bits. I know this will sometimes mean you run down a rat hole, building trivial bits of an intractable task. But you'll be showing progress while you lose, which is vastly better than not-showing progress while you lose more quickly. Your manager will almost never get points for you finding out that a solution is intractable faster than you might have. Check – if that's actually your job, much of this document is not for you.

When beginning a project, make a list of tasks. Then make a list of questions which must be answered to perform those tasks, including who needs to answer those questions. Note which ones you have already answered. I know it sounds crazy, but work with me on this one. Now, in another document, note what those answers are that you already have. Do not spend time trying to determine new answers at this stage – either you have already got the answer or it should be listed as a "collect somehow" question. Send a copy to your manager, this makes their life easier, and forward selected portions of the list to each answerer. This gets answering your questions into their task queue. Each one of those "collect answer" questions should be treated as a task. Now begin performing tasks.

Make daily logs. Most of you get parts of many more tasks done every day than anyone actually realizes (including you). Don't expect to remember what you've been doing; write down completed subtasks as you work on things so that you can forward it at regular intervals. It's easier for your manager to throw away data (if you've organized it well for them) than it is for them to extract it from you if you can't remember things. BTW, this is one of the skills which makes admins really love a manager – the near-psychic ability to figure out what their staff are actually working on, even though their staff aren't very communicative. As former admins themselves, they might be good at interpreting those reticent grunts you give out when you're slogging through a lame name-service problem for the fourth day in a row, but you're still making progress and so aren't at the "just firebomb the vendor and get it over with" stage. But being able to perceive that requires a lot of domain experience on the part of your manager. And that much insight is really expensive to maintain, so try not to bet your career on your manager always being able to bring it to bear on your behalf.

If you are hit with inspiration, work on that task until you run out of steam. Take good notes while you're doing so. Then complete a trivial task.

Do not work on more than one complex task per day, unless you (1) have finished a complex task or (2) are inspired. Don't let a unit-of-time go by without finishing at least one task.

Try to make your list of tasks contain tasks of comparable amounts of temporal effort. Perform those tasks by strictly alternating trivial tasks and complex tasks within a unit-of-time (day/week/whatever).

Once per mega-unit-of-time, ask people who you need information from (see the task-listing task, above) to answer the questions you need them to answer. Getting information from someone is itself a (not always trivial) task. Do not attempt to complete getting information from more than one person per day; keep trying to get info from different people until *someone* gives you at least one answer, but stop when you've succeeded with one of them. If other people send you answers, that's gravy, but you don't want to go radio silent because you're spending days on end appearing to block while you're trying to extract information from other people. If someone tells you to go find the information in a named location, that should be construed as an answer for the purposes of this discussion, although it creates a "collect information from a known document" task. "I don't know" is not an answer, but changes your list of people to ask. If you get an answer or an "I don't know," write down the answer in your answers list.

Finally, let me reiterate the cardinal rule: Silence is bad. Management cannot differentiate between someone who's in over their head, someone who is malingering, someone who's trying to solve an intractable problem, and someone who is making progress on a hard design issue. You'll note that many of those options are bad. If you don't tell your manager what you're doing in a way that management can easily communicate to their peers, you're creating a lot of new work for your manager in two ways: first, by creating a need for them to defend you to their peers, and second, by making it actually difficult for them to do so. Good managers will review and evaluate their own focus and resource allocation continually. Making it easy for them to do so is good for both of you.

Silence is bad.

kick those BUTs

by Steve Johnson

Steve Johnson has been a technical manager on and off for nearly two decades. At AT&T, he's best known for writing Yacc, Lint, and the Portable Compiler.

<scj@transmeta.com>

and Dusty White

Dusty White works as a management consultant in Silicon Valley, where she acts as a trainer, coach, and troubleshooter for technical companies.

<dustywhite@earthlink.net>

Last issue we discussed OR. We continue our tour of short words by today discussing BUT.

"I liked the movie. But the ending was awful."

"You're doing a good job. But I wish you would show more leadership."

So, did I really like the movie? As soon as the word BUT comes through the door, YES and NO go out the window. If someone later asks you whether I liked the movie, it's hard for you to say YES (even though I said I did), because the effect of the BUT was to pretty much negate anything that came before it.

"Wait a minute," you may say, "what's wrong with trying to be precise by talking in detail about my reaction, rather than just saying YES or NO?" Are you really being precise by saying two things that contradict each other? We would argue that you aren't. You are simply making statements that are very much open to confusion and misinterpretation. Depending on your biases, you may hear and quote me that I liked the movie, or that I thought the ending was awful.

More precision looks like this:

"I liked the first three quarters of the movie. The ending was awful."

This gets rid of the direct logical contradiction, but is still emotionally ambiguous – it doesn't tell how you felt about the movie as a whole. Even more precise would be:

"The first three quarters of the movie were great. The ending was awful. On the whole, I was disappointed with the movie."

Not too many people may care whether I liked a particular movie or not. However, people tend to be very interested when the sentences involve their performance doing their job. In our experience, the word BUT appears in many performance reviews, and almost always should be eliminated. The same problems of logical contradiction and emotional ambiguity that were irritations when discussing the movies can be profoundly upsetting when people are getting their job reviews.

How can you get rid of BUTs? One rule is to replace most BUTs by AND. This makes it clear that you stand behind both sentences, and aren't using the second sentence to take away the first. This is typically linguistically correct. It may feel awkward, especially if the two sentences have different emotional tone.

A good way to get rid of the emotional contradictions implicit in BUT statements is to displace the communication in time. Typically, in a performance review you are more interested in improving future performance than in punishing someone for past mistakes. So you can phrase suggestions for improvement as "if . . . then" statements about their future performance and point out the advantages that would come to everyone if the person changed their behavior:

"If you could show more leadership this coming year, it would make my life easier and position you well for a future promotion."

or

"By showing more leadership next year, you'll find that newer employees will be able to benefit more from your experience..."

Let's face it – as a manager, if we think someone should have been doing something and we didn't tell them to do it, we don't have any moral right to spring it on them in the performance review. We should have been talking about it all year.

Another way of defusing the emotional ambiguity of BUT, especially when there are serious problems that need correction, is to use the "feedback sandwich." Talk about the positive, then talk about the negative, then overall talk about what is positive. Assuming you aren't actually firing the person, keeping the emphasis on more positive results in the future is a good strategy. So you might write something like:

> "You have come up to speed quickly and accomplished more than we expected in the first year. Unfortunately, you made a couple of key mistakes that cost us the Fletcher account, leading to an overall negative assessment of your performance this year. We expect that you can put these mistakes behind you and be more successful next year.

This conveys four messages. There were some good things about the year. There were some problems. Overall, the problems predominated. And in the future we expect you will grow past this difficult year. By contrast, the typical BUT sentence:

> "You had a good year, but your mistakes cost us the Fletcher account."

is much more ambiguous, and the overall tone more negative.

So when writing reviews, or in general when expressing your opinion, use your word processor to look for BUTs, and get rid of them. There is always a better way to be more precise.

Research Exchange Program Update from the Field

by Wilfred Dittmer

Scientific Programmer, Vrije Universiteit, Amsterdam, Netherlands
<wdittmer@cs.vu.nl>

A report on the ReX exchange program jointly supported by USENIX and NLnet. See *<http://www.usenix.org/about/rex.html>* and *<http://www.nlnet.nl/projects/rex/>* for information about this valuable program.

Vrije Universiteit and the University of California, San Diego, Cooperative Association for Internet Data Analysis

USENIX and NLnet offered me the possibility to work for the Cooperative Association for Internet Data Analysis (CAIDA) in San Diego, CA, for a period of six months via the Research Exchange (ReX) program. At the time, I had just graduated and was going to work for the Vrije Universiteit (VU) in Amsterdam. VU didn't have a job opening, so it was great to participate in the exchange in the meantime. The purpose of the exchange was to enhance and develop tools for measuring Internet traffic in order to obtain better insight into the physical organization of the network through the Skitter project at CAIDA. First, I will tell a bit more about VU and CAIDA.

VU *<http://www.vu.nl>* is where I graduated with a degree in computer science. As a student, I worked on the GLOBE *<http://www.cs.vu.nl/~steen/globe>* project, which is a novel scalable infrastructure for a massive worldwide distributed system. I wrote an application that uses GLOBE to locate and contact users over the Internet, called Loc8 *<http://www.cs.vu.nl/~baggio/loc8.html>*.

CAIDA *<http://www.caida.org>* is an organization that uses its skitter tool to actively probe the Internet in order to analyze topology and performance. Skitter measures forward IP paths by recording each hop to many destinations. Using ICMP echo requests, skitter also collects Round-Trip Time (RTT) to those destinations. The data collected provides indications of low-frequency, persistent routing changes, and correlations between RTT and time of day may reveal a change in either forward- or reverse-path routing. The skitter data can also be used by their Otter tool to visualize the directed graph from a source to much of the Internet. Otter can handle visualization tasks for a wide variety of Internet data, including datasets on topology, workload, performance, and routing. CAIDA has written many more tools to collect, analyze, and visualize data: for example, CoralReef and Walrus. Now on to the projects I worked on for CAIDA.

The first project I worked on was to map reverse-traceroute and looking-glass servers onto a world map. Using Internet search engines and emails sent by people maintaining reverse-traceroute and looking-glass servers, I collected as many reverse-traceroute and looking-glass pages as I could. Using IP addresses and NetGeo, I could find their latitude and longitude as well as city, state, and country. With this information I then used GeoPlot to display the servers on a world map.

The servers appear as little dots on the map; clicking on them brings up a page containing the URLs for the traceroute and looking-glass servers for that node or will zoom in to that particular region. The nodes are colored to indicate what services are available for that location.

The information collected was also used to obtain the AS numbers for all the traceroute and looking-glass pages (again using NetGeo) and to build a Web page that allows users to search our database for traceroute and/or looking-glass servers by AS number. This page also allows users to search by AS name, latitude, longitude, city, state, or country.

The purpose of these pages is to give users with network problems the ability to find the nearest traceroute/looking-glass service, in a convenient way based on the location they desire to trace from. The traceroute/looking-glass services can then be used to view their own network and can provide hints to what causes their networking problem. The page can be found at: <*http://www.caida.org/analysis/routing/reversetrace*>.

The second project was to enhance the Web pages that provide users with daily summaries of data collected by the skitter boxes around the world. I first had to familiarize myself with the way the scripts currently processed the data for Web usage before I was able to expand their Web pages. In particular, I added pie charts to display the percentage of paths going through a particular country or AS. Because a user can select multiple skitter boxes and dates to generate this pie chart, I had to merge the data of the skitter boxes and dates before I could generate the pie charts.

I also added AS connectivity graph images to the Web pages. An AS graph image shows AS nodes arranged in a circle with polar coordinates, with their angle based on the location of their AS headquarters and their distance from the center reflecting the richness of their observed connectivity to other ASes. Those nodes closest to the center are the most connected ASes as observed for those particular days and skitter boxes. The graph is drawn by Otter. Users can download a data file for Otter that allows the user to view the graph on their own machine with more detail. The page can be found at: <*http://www.caida.org/cgi-bin/skitter_summary/main.pl*>.

The third project was to enhance the Otter network mapping/visualization tool so that it has arrows at the end of the links between nodes to indicate the direction of the link. We would want to print the arrows too, which required a better understanding of how PostScript works, since Otter saves images in PostScript format. After this, I implemented the arrangement of the links entering or leaving the node. Instead of all links merging in the center of the node, the lines are now spread out over the four sides (a node is displayed as a square) depending on the location of the node it is connected to. Links are also sorted by up/down or left/right to the same node.

The final project, which I was unable to finish because of lack of time, was generating animations of AS graph images for specific servers aggregated over a particular number of days. I used Otter to generate the graphs. Otter will use the maximum size of a page when it's printing to file. For animations this is not preferable, because the images will vary in size and common nodes between frames will jump over the screen. So we define four anchor points in every frame, with the minimum value set to the minimum outdegree found in all frames. After this we normalize all outdegrees to be a percentage of the maximum outdegree found in all frames.

I extended the program to use only a limited number of nodes in the animation to improve visibility of the most connected nodes. After doing a day-by-day animation for 364 days in 2000, the nodes were still jumping a little. The way to solve this problem is to generate intermediate frames that smooth out the movement of the nodes between frames. Because of time constraints, this has not yet been implemented.

Future projects at CAIDA include the integration of bandwidth estimation data into a database with Otter and/or Walrus as a front end. The data will be collected by a tool suite called netchar that uses features from Bruce Mah's pchar, Allen Downey's clink, and Constantinos Dovrolis' pathrate. The data-processing back end refines and extends capabilities in Allen Downey's clink, estimates the bandwidth and latency of each link appearing in a path, and will assemble a view of the subnets connecting the hosts.

REFERENCES TO PROGRAMS USED/MENTIONED:

Reverse traceroute servers allow the user to traceroute from the server's perspective. The user can see the return path from that server to his or her own machine or to other machines.

Looking-glass servers allow users to see BGP tables at a particular point in the Internet.

NetGeo uses "whois" records to determine the location given a domain name. See <*http://www.caida.org/tools/utilities/netgeo*>.

Geoplot is a light-weight java applet that allows users to create a geographical image of a data set. See:
<*http://www.caida.org/tools/visualization/geoplot*>.

Otter is a tool used for visualizing arbitrary network data that can be expressed as a set of nodes, links, or paths.See <*http://www.caida.org/tools/visualization/otter*>.

pchar:
<*http://www.employees.org/~bmah/Software/pchar/*>.

clink: <*http://rocky.wellesley.edu/downey/clink/*>.

Finally, the graphical user interface will lay out the graph generated by the back end and allow the user to drill down into the collected data and estimated characteristics.

To further strengthen the relation between CAIDA and VU, CAIDA provided us with a GLOBE server in San Diego; in return, VU will set up a skitter machine in Amsterdam. Also, by using CAIDA's data, we can find the best locations for our GLOBE boxes to provide good service coverage, and their data on bandwidth and round-trip time will help us to optimize GLOBE.

Of course the other notable aspects of the exchange were the experiences I had and the new contacts I made. During my work at CAIDA, I met a lot of fantastic people and brought back a number of memories (a rollercoaster park, a real American Thanksgiving, bonfires, parties, etc.) which made the exchange a success.

All in all I found it was a great exchange and would like to thank the following people and institutions for making it all possible: ReX, a USENIX/NLnet endeavor <*rex@usenix.org; rex@nlnet.nl*>; Gale Berkowitz; Frances Brazier; Evi Nemeth; kc claffy; and Maarten van Steen.

women in computing

Collaborative Research Experience for Women in Undergraduate Computer Science and Engineering (CREW) Program

by Jennifer P. Rubenstein

Computing Research Association
Committee on the Status of
Women in Computing Research
(CRA-W)
<jpr@cra.org>

A nerd hunches in a cubicle pounding away at her keyboard 24/7. She is successful, but is that what most women beginning a computing career want?

This image may discourage many undergraduate students from pursuing a career in computing research.

"The stereotype of the loner computer scientist can be especially deterrent to women," said Jan Cuny, 1997-2000 co-chair of the Computing Research Association Committee on the Status of Women in Computing Research (CRA-W). "Women tend to be more motivated by interaction, and so may be rebuffed by the isolation assumed to be associated with research."

To address this stereotype, the Computing Research Association's Committee on the Status of Women (CRA-W) has been implementing a variety of programs.

With support from USENIX and the National Science Foundation, the Collaborative Research Experience for Women in Undergraduate Computer Science and Engineering (CREW) program gives undergraduate women the opportunity to experience a year-long research project. It is hoped that this experience will show students a more realistic view of a career in computing and encourage them to attend graduate school. CREW supports the formation of teams of undergraduate women who collaborate on joint research projects under the direction of a faculty member at their home institutions during the academic year.

Dr. Lynn Stauffer from Sonoma State University describes the research experience she had with her students this way: "I have found the CREW program to be a wonderful way to introduce talented CS students to research at our small university. Without a graduate program and with a demanding local high-technology industry, our computer science department has a tough time interesting students in research work that does not have any monetary benefits (i.e., talented CS students can easily find well-paying internships nearby). Also, the simplified application process makes putting a proposal together more doable for already very busy faculty. In fact, I found the proposal writing part of the project particularly enjoyable, and I believe the student researchers learned a lot from the experience."

In the last few years, CREW teams have studied robot navigation and vision, parallel processor communication, Web navigation, and integrated circuit design. At the end of their projects, students write summaries of their work and are encouraged to submit papers and present their work to other appropriate journals and conferences. Students have presented their work at CHI 2001, SIGCSE 2001 and the National Conference on Undergraduate Research.

There were eight projects funded in 1998-1999, 10 projects in 1999-2000, and 11 projects in 2000-2001. The fourth year's awards will be announced June 30 for the 2001-2002 academic year. Because of additional funding from USENIX, twice as many projects as previous years will be funded for next year. It is hoped that by securing even

THANK YOU TO SHEILA E. CASTANEDA, CREW PROJECT DIRECTOR AND CHAIR, ASSOCIATE PROFESSOR, COMPUTER SCIENCE DEPARTMENT, CLARKE COLLEGE, DUBUQUE, IA FOR HER CONTRIBUTIONS TO THIS ARTICLE AND PROGRAM

more funding, as many as 100 projects for the 2002-2003 academic year can be supported. An average of three students collaborate on each project, so the impact on individuals and institutions is substantial.

Evaluation results from the first three years of the program have been encouraging. Based on student surveys:

- 75% indicated that this was their first experience doing research
- 67% indicated that they plan to attend graduate school
- 25% credited CREW for their increased interest in and preparation for graduate school

If CREW can increase graduate school enrollment for women by 25% as it scales up to 100 projects, that would mean an additional 75 or more women choosing to attend graduate school every year.

CRA-W was established in 1991 with the goal of taking positive action to increase the number and success of women in CS&E research. The committee is comprised of leaders in computing research from academia and industry. It is an action-based committee, implementing projects that aim to eliminate barriers to the full participation of women. More information about CRA-W and CREW can be found at <*http://www.cra.org/craw/*>.

The Computing Research Association is a tax-exempt (501c3) association of more than 180 North American academic departments of computer science and engineering (CS&E); 25 research laboratories and centers in industry, government, and academia engaging in basic computing research; and six affiliated professional societies [USENIX being one]. CRA works to strengthen research and education in the computing fields, expand opportunities for women and minorities, and improve public understanding of the importance of computing and computing research in our society. More information about CRA is available at <*www.cra.org/*>.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He is Editorial Director at Matrix.net. He owns neither a dog nor a cat.

<peter@pedant.com>

Every so often, rather than lots of brief comments, I spend time on a book or two. This month I'm devoting the column to three items, which are, quite simply, outstanding.

The Universe of the Net

Mapping Cyberspace is a brilliant attempt at grappling with the representation of what has been called "cyberspace." I say representation because Dodge and Kitchin are not merely geographers or cartographers. They are also concerned with the sociocultural concepts that Baudrillard (or Greimas or Derrida) would endorse.

Maps are attempts at presenting the surface of the world in two-dimensional symbolic form, according to a pre-WWII encyclopedia. Over the past decades, we have used "map" in a far broader sense. In the early 1970s, Gould and White wrote *Mental Maps*, a study of the geography of perception of the images we form of places, and Joan Foley of the University of Toronto was examining the ways students and faculty related campus locations to one another.

The term "cyberspace" was used by Bill Gibson in an article in *Omni* in 1982, and in 1984 it appeared in his *Neuromancer*. The OED *Additions Series* (vol. 3, 1997, s.v.) defines cyberspace as "The notional environment within which electronic communication occurs, esp. when represented as the inside of a computer system . . . the space of virtual reality."

This last definition takes us to Howard Rheingold's 1991 *Virtual Reality* and his 1993 *The Virtual Community: Homesteading on the Electronic Frontier*.

We generally think of maps on flat pieces of paper, or wrapped on a sphere (a globe). But much more is involved in our attempts at mapping information and communications technologies (Chapter 5), mapping asynchronous media (Chapter 7), or mapping synchronous social spaces (Chapter 8), to say nothing of "imaginative mappings" (Chapter 10) or the future (Chapter 11). By preceding these chapters with four introductory chapters (1-4) on cyberspace, geography, and cartography, Dodge and Kitchin have produced a genuine "must read" for sociologists, political scientists, and network engineers.

Think of it, how does one go about mapping a MUD or a MOO? What is entailed in mapping a chat room? Yet I would guess that no one reading this has any doubt as to the conceptual reality of the places we meet online, in the sites we converse in, in the spaces we bargain and market in.

Dodge and Kitchin devote a good deal of time to their discussion of Usenet. And well they might. From the three groups of 1979, to the 300 of 1986, to the 20,000 or so as of last year, the growth of the news groups has reflected the growth in the numbers of users.

I have long found the notion that the vast reticulum that comprises the Internet contains a culture like that of the "coffeehouse," a fascinating one. Smith's views of communities in cyberspace are important here, and Rheingold has used the coffeehouse as a metaphor for Usenet, so we can reflect upon Baudrillard and the notion of the "electronic coffeehouse."

After email and the Web, Usenet is the next most frequent use of the Internet. Dodge and Kitchin limn the "spatial

BOOKS REVIEWED IN THIS COLUMN

MAPPING CYBERSPACE
MARTIN DODGE AND ROB KITCHIN
London & New York: Routledge, 2001. Pp. 260. ISBN 0-415-19884-4.

DIGITAL COPYRIGHT
JESSICA LITMAN
Amherst, NY: Prometheus Books, 2001. Pp. 208. ISBN 1-57392-889-5.

WHITE HAT SECURITY ARSENAL
AVIEL D. RUBIN
Boston, MA: Addison-Wesley, 2001. Pp. 510. ISBN 0-201-71114-1.

INTRUSION DETECTION
REBECCA GURLEY BACE
Indianapolis, IN: Macmillan Technical Publishing, 2000. Pp. 339.ISBN 1-57870-185-6.

structure" as composed of two features: groups and articles. It's important, they point out (using Smith's Netscan data), to note the geographical diffusion of postings: of 238 TLDs, there were postings from 205 of them in 1997. Of these, 41% were from hosts in the US. The conceptual cartography of Usenet is thus vastly different from geographical cartography. In fact, the percentages don't reflect numbers of hosts, either: in 1999, over 50% of hosts were in the US. Some areas are more voluble than their representation reflects.

Cyberspace is non-planar, so we must deal more with conceptual maps than with bi-dimensionality. What Dodge and Kitchin have done is not so much map cyberspace as give us impetus to find hyper-cartographical means to represent connectivity in the 21st century. Their chapters on mapping asynchronous and synchronous spaces, "spatial cognition of cyberspace," and "future mappings of cyberspace" are of value here.

Napster, etc.

In 1998, lobbyists (largely from the film/TV and recorded music industries) persuaded the US Congress to pass the Digital Millennium Copyright Act (DMCA), which sharply restricts private use of works that are under copyright. The concepts of pay-per-view and pay-per-listen follow, as does the ongoing war on Napster. (It's not clear to me whether the forces of Mammon know about Gnutella, yet.)

Jessica Litman, a law professor at Wayne State University, has turned out a brilliant book, *Digital Copyright*, on this topic.

Litman skims through the nearly 300 years of copyright law and goes into some detail where the contributions of copyright lawyers, greedy media moguls, and avaricious congressional representatives combined to create a stupid law with nearly no technical input. Nor any consideration for the user, the scholar, or the ordinary citizen.

Glued to the traditions of distribution, contemporary recording, and production, executives still think in terms of centralized manufacture and distribution. To a certain extent, this militates against companies like Napster, but shared files over the Internet are decentralized, and Gnutella and Freenet provide the recording and distribution companies with no easy target to haul into court.

There is already some recognition that the DMCA hasn't worked and that some new legislation is required (see Ham & Atkinson, "Napster and Online Piracy," Progressive Policy Institute report, May 2000). As Litman points out: "Unless the stakeholders do something very different this time around, though, that law won't work either" (p. 170).

Litman enabled me to understand just how the DMCA protects neither authors nor artists but, rather, the corporate media masters.

Shrink-wrapped software licenses in microscopic print may constitute the greatest support there could be for the FSF, for Linux, for the Open Source movement. *Digital Copyright* will explain just why the DMCA is hopeless.

Countering the Interlopers

Avi Rubin has been doing good work about security matters for a long time. His article on one-time passwords was one of the very best in the final volume of *Computing Systems*. So I approached his book on how to handle Internet threats with great interest. I was well-rewarded.

This is not your standard how-to security book. This is a well-designed, well-written volume on just what the threats are, how they work, and what you have on hand to resist these threats.

Viruses, worms, denial of service attacks are just the beginning of this. Most interestingly, Rubin dissects the Morris Work, Melissa, I Love You, and several other malicious invertebrates. His explanations of just how these infiltrative beasties work is just brilliant.

I enjoyed his section on secure transfer and on setting up session keys, too. His chapters on SSL and on encrypted email are also fine.

This is a "different" security book, and it's one you really need.

(I feel I ought to mention Bace's *Intrusion Detection* here. While DDoS attacks block, most other attacks are intrusive. In well under 300 pages (plus appendices), Bace fully informed me of problems and methods.)

SAGE news

New SAGE Officers

by Trey Harris

Secretary, SAGE Executive Committee

<trey@sage.org>

At the first meeting of the new SAGE Executive Committee, held 9-10 March 2001, in Emeryville, CA, the officers for the upcoming year were chosen. They are:

President: David Parter, University of Wisconsin (<*parter@sage.org*>)

Vice President: Geoff Halprin, The SysAdmin Group (<*geoff@sage.org*>)

Secretary: Trey Harris, VA Linux Systems (<*trey@sage.org*>)

Treasurer: Peg Schafer, Harvard University (<*peg@sage.org*>)

Additional members of the Executive Committee are:

Strata Rose Chalup, VirtualNet Consulting (<*strata@sage.org*>)

Barb Dijker, NeTrack (<*barb@sage.org*>)

Tim Gassaway, Auspex Systems (<*gassaway@sage.org*>)

Andrew Hume (<*andrew@usenix.org*>) serves as the USENIX board's liaison to the Committee.

Each member of the Executive Committee was also given projects to oversee or "champion." This is not meant to be an exhaustive list of the projects SAGE or the members of the Executive Committee are involved in, but it may give you an idea of their current work:

International SAGE groups: Strata Rose Chalup, Geoff Halprin, Barb Dijker

Memo-to-Members: Trey Harris

Public Relations: David Parter

Review of SAGE policies: David Parter

SAGE Certification: Barb Dijker

SAGE Certification Test Development Committee: Trey Harris

SAGE Code of Ethics: Barb Dijker

SAGE Mentoring: Strata Rose Chalup

SAGE Publications: Peg Schafer

SAGE/USENIX restructuring: Barb Dijker, Peg Schafer

SAGE Web Site revamping: Strata Rose Chalup, Geoff Halprin, David Parter

Salary Survey: Peg Schafer

Sysadmin Internship Project: Peg Schafer

Conference Liaisons: Strata Rose Chalup (LISA 2001), Trey Harris (SNAC 2002), Geoff Halprin (BSD 2002)

The Executive Committee eagerly seeks input from members. You can contact them individually at the email addresses above, or as a group at <*sage-exec@sage.org*>.

SAGE Certification Update

by Lois Bennett

Member, SAGE Certification Committee

<lois@deas.harvard.edu>

Since the last *;login:* article, work has continued on the certification front. The Policy Committee has had several meetings via conference call, as they continue to develop the policies and procedures needed to implement the certification program. The SAGE Executive Committee had a lengthy discussion about certification with the purpose of refining the direction of the program. The Test Development Committee met to review and modify the questions that have been submitted to the exam item bank. Additional questions will be drafted over the next two months to complete the exam item bank needed for the first certification exam.

The certification will have practical as well as theoretical components. There will be a multiple choice test and some form of practical test. The multiple choice test that is being written now will be vendor neutral and platform independent. The practical test, still in the planning stages, will be platform specific. More information on this effort can be found on the SAGE Certification Web page: <*http://www.usenix.org/sage/cert*>.

USENIX news

Membership

by Daniel Geer
President, USENIX Board of Directors

<geer@usenix.org>

"Membership in USENIX has been growing at a sustained double-digit rate for years."

(a) Good news!
(b) Define "membership"
(c) Both

The correct answer is (c), *it is* good news but the definition of member is the crucial detail. This is not unique to USENIX – if there is anything I have learned in Washington it is that when it comes to regulations, the game is over after the definitions page; ditto employment contracts; ditto patents; ditto pre-nuptial agreements; ditto software licenses. Nearly every durable declaration of fact depends critically on what the definition of terms is. (For further reflection, re-read Humpty Dumpty's remarks in *Through the Looking Glass.*)

So what is a USENIX Member and why does it matter? A USENIX Member, as we mean it here, is someone who pays the membership fee. They get benefits, some of which are tangible and detailed on our homepage, and some of which are intangible. The intangible, like making colleagues out of people you would never otherwise even meet, tends to grow in importance to you the longer you are here. As with any membership organization, we like to see people stay around a longer time, but if you are reading this you are (likely) already a Member. We hope you stay a long time; I hope you someday find yourself leading this organization.

What USENIX does better than anything else is concentrate (information about) technical progress in the building and running of systems. We bring together people and ideas. In that tired analogy, we assemble critical mass. Sometimes, the number of people is small, but the ideas are grand, sometimes the reverse is true. Every conference, symposium and workshop we run is a risk, a risk that is exactly proportional to how good we all (you) are at deciding what is important versus merely showy. The reason we are still here and growing is simply that we have been pretty good handicappers over a sustained period. I'd claim that what Warren Buffet is to investing, USENIX is to building real systems – not always glamorous, but forever driven by results, by what actually works under load and what does not.

A former member of the USENIX Board somewhat famously said that what USENIX does is "Move information from where it is to where it isn't." I like that. As some would say, "The future is already here, just unevenly distributed." I like that, too. Putting those two together is close to what USENIX is fundamentally about – catching first light of the future and accelerating the work of our members, people both motivated by and unafraid of results, by moving information from where it is to where it is not.

So what is a Member? We have to stick with the definition we have on "paper" for who has suffrage for our elections. We'll continue to more or less make it automatic to join if you attend one of our meetings. But I want to ask you a straightforward question: If over half the attendees at any meeting are first-timers, it's clear that many of them don't come back. Is this natural selection or something else? I happen to think that it *is* natural selection, and we *are* doing the right thing when we make it easy to try

USENIX MEMBER BENEFITS

As a member of the USENIX Association, you receive the following benefits:

FREE SUBSCRIPTION TO *;login:*, the Association's magazine, published eight times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on security, Tcl, Perl, Java, and operating systems, book and software reviews, summaries of sessions at USENIX conferences, and reports on various standards activities.

ACCESS TO *;login:* online from October 1997 to last month <*www.usenix.org/publications/login/login.html*>.

ACCESS TO PAPERS from the USENIX Conferences online starting with 1993 <*www.usenix.org/publications/library/index.html*>.

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

OPTIONAL MEMBERSHIP in SAGE, the System Administrators Guild.

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. See <*http://www.usenix.org/membership/specialdisc.html*> for details.

FOR MORE INFORMATION REGARDING MEMBERSHIP OR BENEFITS, PLEASE SEE

<*http://www.usenix.org/membership/membership.html*>

OR CONTACT

<*office@usenix.org*>

Phone: +1 510 528 8649

FOR INFORMATION ABOUT CONFERENCES, PLEASE SEE

<*http://www.usenix.org/events/events.html*>

OR CONTACT

<*conference@usenix.org*>

Phone: +1 510 528 8649

out being a Member – sort of submitting ourselves to market forces by defining our Members as those who find what we do here to be valuable to them on a sustaining basis.

Now this leads to a detail: If I am right – that our *raison d'être* is to move information from where it is to where it needs to be and that the reason Members (you) are members is that this is actually valuable to you and your careers – then how much do you want your membership to be an advantage of time and place to you? How much do you want to be a USENIX Member in order to Get It First? How much is your being part of this really rather unique organization about gaining an advantage to you personally in what you do yourself? How much do you want to make membership a ticket to things that are simply not otherwise just lying about for anyone to use? Does what you want align with what it takes to keep USENIX viable for the bulk of your productive life?

Rousseau suggested that democracy survives so long as the electorate cannot give away the treasury. If I am right about what makes USENIX valuable, our treasury is measured in information. How we (you) strike this balance between what is special for our Members and what is for just anyone only seems trivial. Until you think about it.

What's Up with Charity?

by Andrew Hume

Vice-President, USENIX Board of Directors

<andrew@usenix.org>

Over the last seven years, USENIX has funded a modest number of projects that I would describe as "charitable." By this I mean projects primarily benefiting the community at large, rather than the specific technical community to which nearly all our members belong. Much to my surprise, the current USENIX board now has a majority opposed to such projects. This opposition is ideological in nature and has little to do with the amount of funds involved. Of course, the board is quite responsive to "what the membership wants," and frankly, I want you to let the board know if you agree with me, so that we can continue to fund community projects when they present themselves. It would also help the board develop a strategy for such proposals so that decisions can be made on a less ad hoc basis.

I believe that every person and organization has a responsibility to help those in need; this is a significant part of what makes us a society. This is why I personally donate money to the United Way, and it's why AT&T, amongst its other charitable works, contributes the overhead for my donations. Of course, everyone has to decide for themselves how to express that responsibility. I want to give a couple of (I think good) examples of how USENIX has done this in the past.

In 1996, we spent about $50K on a project centered on a settlement house in lower Manhattan. "Settlement houses" are basically community centers in poorer urban neighborhoods, albeit with a long history of service. The two main activities affecting this discussion are after-school programs for kids and evening outreach for adults. The settlement houses already had some computer labs set up, but they were not being utilized well. In collaboration with a local university, we paid for a better network connection (T1) to the Internet, and we funded undergraduates to teach/tutor people in the computer labs in the use of computer technology. This program has been very successful, and we are being asked to fund a similar effort in a settlement house in Boston.

In August 1999, we funded a request for $40K from SOS Children's Village near Chicago. The village is part of the Illinois adoption system; it is the destination of last resort for children who have been bounced through several placements. Physically, the village is a set of several houses around a cul-de-sac. The project involved setting up a LAN between the houses, and buying a couple of computers for each of the houses (each of which has 6–10 children), to be used mainly for homework. We also involved a SAGE volunteer to help with configuring, ordering, setting up, and maintaining the network, and teaching the "villagers" how to run the network themselves.

I think both these cases exemplify effective charity; we helped those in real need, we did so in areas that we are familiar with, and we set up connections between the recipients and others in their local community (students and volunteers). I think continuing to support this type of project is a no-brainer, especially at the level of funding (~3% of our good-works budget) we have done in the past. Actually, I think it is so obviously a good idea I am gobsmacked that any reasonable person would oppose it! Dan Geer revealed in the last *;login:* that he thinks we ought to put all our resources into benefiting our members directly. Certainly, that should be our main thrust, but to not set aside a few percent seems mean-spirited and inconsistent with the generosity and camaraderie displayed at our conferences by our members.

If you agree with me, or even if you don't, please do let the board know!

Twenty Years Ago in U[SE]NIX

by Peter H. Salus
USENIX Historian
<peter@matrix.net>

There was no June 1976 meeting/conference. Lew Law hosted meetings at Harvard in April (with 60 attendees) and October (with double that number). So 25 comes up empty for me.

The June 1981 conference was hosted by Wally Wedel at the University of Texas. While it was well-attended (500), the numbers were but half of those at San Francisco (Tom Ferrin) and Santa Monica (Mike O'Brien, long before Mr. Protocol) which bracketed it.

The community was still bi-coastal: Urbana, IL, Toronto, Denver, Austin all attracted far fewer attendees than Cambridge, MA, New York, Berkeley, San Francisco, or San Diego.

But June 1981 saw the released of 4.1BSD, which contained a large number of performance improvements, support for a new VAX model, and auto-configuration. The summer also saw Kirk McKusick join the CSRG.

Kirk says: "4.1BSD was the high point of performance, in the sense that it was really 4.0BSD [October 1980] but tuned to a fine hone, especially for the 750. The 4.1BSD system was taken back into Bell Labs to become the 8th Edition UNIX."

Armando Stettner remarked to me that given the choice between 4.1BSD and V7, "no one ever wanted to go back to V7."

In June 1981, the new Internet was growing, too. It had 200 host sites and well over 2,500 users.

USENIX Funds Electronic Frontier Foundation

The USENIX Board of Directors recently decided to fund the Electronic Frontier Foundation's efforts to protect copyright and fair use rights related to the Digital Millennium Copyright Act (DMCA) legal cases. The Electronic Frontier Foundation (EFF) is pursuing several legal cases to protect copyright and fair use rights by opposing the anticircumvention rules of the Digital Millennium Copyright Act (DMCA) as violating constitutional rights to free expression. The cases build on EFF's earlier precedent-setting victory, Bernstein vs. U.S. Department of Justice, where a federal appeals court ruled that code is free speech and, therefore, protected by the Constitution.

The USENIX Association also helped fund the Bernstein case in 2000.

The USENIX Association recently renewed its support for the Electronic Frontier Foundation (EFF) by committing $150,000 over the next three years to protect copyright and fair use rights related to the Digital Millennium Copyright Act (DMCA) legal cases. EFF expects that approximately $1.5 million will be needed over the next three years to fully support the project. See their website <*http://www.eff.org/support/20010419_eff_fund_drive.html*> for information about contributing to this effort.

ABOUT THE ELECTRONIC FRONTIER FOUNDATION:

The Electronic Frontier Foundation is the leading civil liberties organization working to protect rights in the digital world. Founded in 1990, EFF actively encourages and challenges industry and government to support free expression, privacy, and openness in the information society. EFF's objectives are to ensure that fundamental rights are at least as well secured online as they are offline; to educate the press, policymakers, and the general public about online civil liberties; and to act as a defender of those liberties when they are attacked. For more information about EFF, please see: <*http://www.eff.org*>.

Notice of Annual Meeting

The USENIX Association's Annual Meeting of the Board of Directors with the membership will be held during the 2001 USENIX Annual Technical Conference in Boston in June. The exact date, time and place will be announced on site.

We welcome all to come and participate.

USACO News

by Rob Kolstad
;login: Editor
<kolstad@usenix.org>

The first phase of the USACO contest season is wrapping up. We've held the US Open and chosen the finalists for training camp in addition to recognizing dozen and a half "USACO All Americans." From the finalists, four will be chosen to attend the IOI contest in Tampere, Finland in July.

The US Open had 358 total contest entries from 40 different countries, including 187 from the USA (which is an improvement in USA participation!). The contest was run both in a proctored version (for USA and other participants) and a week-long (but five-hour limited) non-proctored version. Contestants chose to enter either of two divisions:

- The Green division with really hard problems
- The Orange division with somewhat difficult problems

This year, the Green problems were really, really challenging. Scores were much lower than usual and the contestants made it clear that they thought the problems were much harder. It's a challenging task to get the problems "just right."

The top five proctored USA winners along with their HS graduation years, states, and schools, were:

Thuc Vu, 2001, Fairmont Preparatory School, CA.
Songzi Du, 2003, Ben Davis HS, IN.
Reid Barton, 2001, Home Schooled, MA.
Jacob Burnim, 2002, Montgomery Blair HS, MD.
Yuran Lu, 2001 Presque Isle HS, ME.

Thuc Vu placed second last year, nine points shy of perfect.

The not-necessarily-proctored International Division top eight:

Doan Ngoc Minh, 2001, Viet Nam
Li Yiming, 2001, China
Ivan Georgiev, 2001, Bulgaria
Li Rui, 2002, China
Pengxu Li, 2002, China
Wenjie Fu, 2002, China
Martin Pettai, 2002, Estonia
Jozef Tvarozek, 2002, Slovakia

Vietnam and China continue their winning ways. China often takes four gold medals at the IOI.

From the results of this year's four contests, the USACO chose 18 students recognize as the USACO All American Programming team. These students participated strongly and scored consistently high on the contests:

Reid Barton, 2001, Home Schooled, MA
Jacob Burnim, 2002, Montgomery Blair HS, MD.
Kevin Chen, 2002,Western Branch, VA
Jeff Cohen, 2002, TJHSST, VA
Adam D'Angelo, 2002, Phillips Exeter, CT
Songzi Du, 2003, Ben Davis HS, IN
Richard Eager, 2001, TJHSST, VA
Joseph Jaewhan Lim, 2004,Taejeon Christian School
Yuran Lu, 2001, Presque Isle HS, ME
Vladimir Novakovski, 2002 TJHSST, VA
Anatoly Preygel, 2003, Montgomery Blair HS, MD
Gregory Price, 2002, TJHSST, VA
Alex Schwendner, 2005, Homeschool, TX
Gary Sivek, 2002, TJHSST, VA
Steven Sivek, 2002, TJHSST, VA

Thuc Vu, 2001, Fairmont Preparatory School, CA
Tom Widland, 2001, Albuquerque Academy, NM
John Zhu, 2001, Gilbert HS, AZ

The criteria for the final contest rounds to be held at USACO programming camp were slightly different. Among the criteria was a requirement to be able to attend the IOI! This results in slightly different selection for training camp:

Reid Barton, 2001, Home Schooled, MA
Jacob Burnim, 2002, Montgomery Blair HS, MD.
Jeff Cohen, 2002, TJHSST, VA
Adam D'Angelo, 2002, Phillips Exeter, CT
Songzi Du, 2003, Ben Davis HS, IN
Neil Herriot, 2002,Palo Alto HS, CA
Joseph Jaewhan Lim, 2004,Taejeon Christian School
Yuran Lu, 2001, Presque Isle HS, ME
Vladimir Novakovski, 2002 TJHSST, VA
Anatoly Preygel, 2003, Montgomery Blair HS, MD
Alex Schwendner, 2005, Homeschool, TX
Gary Sivek, 2002, TJHSST, VA
Steven Sivek, 2002, TJHSST, VA
Thuc Vu, 2001, Fairmont Preparatory School, CA
Tom Widland, 2001, Albuquerque Academy, NM

Problem four presented a serious challenge to the contestants.

Designed by Greg Galperin, formerly of MIT and now at an airline scheduling startup, it required the students to reorder mixed up cow signs in order to form an advertisement:

Farmer John decided to make the most out of his highwayside farm and sell advertising. He found a client and hiked to his fence which runs along the highway and painted the message on the sides of the cows grazing along the fence. He painted exactly K letters (2 <= K <= 4) on each of C (2 <= C <= 20) cows and ignored spaces.

Although cows are fairly lazy creatures, they are not immobile. After milking the next morning, FJ noticed that the message he was being paid to display was now in pieces in his barn. Worse still, FJ also forgot the original message.

Help FJ reconstruct the original message. Given K, the cows' letter groupings (which all happen to be unique), and a dictionary that contains all the possible words, figure out all the possible messages (note that the message ABCD EF is different from AB CDEF). Each dictionary word is unique and can be used more than once in any given message. All C cows are used exactly once. Being advertising, it is important to ignore grammar and any possible meaning of the message.

It is guaranteed that the number of possible original messages will not exceed 2,000,000,000. Furthermore, all data in this problem is supplied in upper-case.

PROBLEM NAME: sign

INPUT FORMAT:

Line 1: Three integers, K, C, and D (1 ≤ D ≤ 150, the number of words in the dictionary)

Lines 2..C+1: Each line contains K letters of the scrambled message

Lines C+2..C+D+1: Each line contains a single word from a dictionary; no word is longer than ten characters

SAMPLE INPUT (file sign.in):

```
3 5 7
TEN
ATT
NAT
BAR
ACK
AT
ATTACK
BARN
CHICKENS
CHOPPERS
COWS
TEN
```

OUTPUT FORMAT:

Line 1: The first (in alphabetical order) possible message. Note that "A B" precedes "AB" alphabetically.

Line 2: A single integer which is the number of possible messages

If no solutions exist, print a single line containing the word "NOSOLUTIONS".

SAMPLE OUTPUT (file sign.out):

```
ATTACK BARN AT TEN
6
```

[in case you were wondering, here are the six: TEN ATTACK BARN AT, TEN BARN AT ATTACK, ATTACK TEN BARN AT, ATTACK BARN AT TEN, BARN AT TEN ATTACK, and BARN AT ATTACK TEN]

Problem eight, in the orange division, challenged the next tier of students. Designed by Don Piele, USACO director, it asks the students to find numbers with certain properties to enable a combination lock to be opened:

To unlock a combination lock you must rotate a dial alternately right and left through a sequence of positive numbers. For this problem, a sequence (C1, C2, ..., Cn) is called a combination for M if it has the following properties:

1) n > 1
2) Ci - C(i-1) = 1 for all i
3) C1 + C2 + ... Cn = M

Create a program which will find all possible combinations for a given M. Express each combination by giving the first and last number in the combination.

Example: 1998 2002 denotes the following combination for M=10,000:

1998+1999+2000+2001+2002 = 10000

PROBLEM NAME: combo

INPUT FORMAT:

A single line with the integer M (10 ≤ M ≤ 2,000,000).

SAMPLE INPUT (file combo.in):

10000

OUTPUT FORMAT:

A set of lines with two numbers as described above. Order the lines by increasing first number (with ties broken by consulting the second number). It is guaranteed that at least one answer exists.

SAMPLE OUTPUT (file combo.out):

18 142
297 328
388 412
1998 2002

All in all, the year has gone extraordinarily well. A new contest grading system enables the competitors to be confident that their programs read and write the proper files in addition to compiling correctly. The program execution system has moved from DOS (with an average of one reboot per minute) to Linux. Multiple grading machines have enabled contest grading time to be reduced from several days to several minutes. The maximum memory allowed has increased from 640KB to 16MB. The number of grading protests has dropped to the low single digits.

If you know a pre-college programmer who would like to compete, potentially for various prizes, or even just learn more about procedure-oriented programming, please have them stop be at <*http://www.usaco.org*>.

USENIX Needs You

People often ask how they can contribute to the USENIX organization. Here is a list of needs for which USENIX hopes to find volunteers (some contributions reap not only the rewards of fame and the good feeling of having helped but also a slight honorarium). Each issue we hope to have a list of openings and opportunities.

- The *;login:* staff seeks good writers (and readers!) who would like to write reviews of books on topics of interest to our membership. Write to <*peter@matrix.net*>.
- The *;login:* editors seek interesting individuals for interviews. Please submit your ideas to <*login@usenix.org*>.
- *;login:* is seeking attendees of non-USENIX conferences who can write lucid conference summaries. Contact Tina Darmohray, <*tmd@usenix.org*> for eligibility and remuneration info. Conferences of interest include (but are not limited to): Interop, Internet World, Comdex, CES, SOSP, Linux World, O'Reilly Perl Conference, Blackhat (multiple venues), SANS, and IEEE networking conferences. Financial assistance to cover expenses may be available. Contact <*login@usenix.org*>.
- *;login:* always needs conference summarizers for USENIX conferences too! Contact Alain Hénon <*ah@usenix.org*> if you'd like to help.
- The *;login:* staff seeks columnists for:
  - Perl
  - Large site issues (Giga-LISA),
  - Hardware technology (e.g., the future of rotating storage)
  - General technology (e.g., the new triple-wide plasma screens, quantum computing, printing, portable computing)
  - Paradigms that work for you (PDAs, RCS vs. CVS, using laptops during commutes, how you store voluminous mail, file organization, policies of all sorts)
  - Comics/cartoons (need to find them, not necessarily draw them).

  Contact <*login@usenix.org*>.
- The *;login:* staff seeks editors for "special topics" issues or partial issues. If you would like to assemble a collection of articles on a particular topic please contact Rob Kolstad, <*kolstad@usenix.org*>. (See, for example, the November 2000 issue on Security). This is a paid position.

Announcement and Call for Papers **USENIX**

FAST 2002
Conference on File and Storage Technologies

Co-Sponsored by USENIX and ACM SIGOPS (pending)

http://www.usenix.org/events/fast

January 28-29, 2002 **Monterey, California, USA**

Important Dates

Paper submissions due: *July 13, 2001*
Notification of acceptance: *September 14, 2001*
Camera-ready final papers due: *November 15, 2001*
Conference begins: *January 28, 2002*

Conference Organizers

Program Chair
Darrell Long, *UC Santa Cruz*
Program Committee
David Black, *EMC*
Walter Burkhard, *UC San Diego*
Randal Burns, *IBM Research*
Jeff Chase, *Duke*
Richard Golding, *Panasas*
Roger Haskin, *IBM Research*
Peter Honeyman, *University of Michigan*
David Kotz, *Dartmouth*
Mike Leis, *Quantum*
Ethan Miller, *UC Santa Cruz*
David Nagle, *Carnegie Mellon University*
Rodney Van Meter, *Nokia*
Randy Wang, *Princeton University*
John Wilkes, *Hewlett-Packard*

Overview

File and storage systems are critical to business and society, holding the "crown jewels" of most Information Age organizations and dictating the performance of most computer systems. FAST brings together the top storage systems researchers and practitioners, providing a premier forum for discussing the design, implementation, and uses of storage systems.

The conference will consist of two days of technical presentations, including refereed papers, invited talks, and an introductory keynote address. A session of work-in-progress presentations is planned, and informal Birds-of-a-Feather sessions may be organized by attendees. Refereed papers will be published in the Proceedings, provided free to technical session attendees, and available for purchase from USENIX.

Topics

Papers may be on any topic related to file systems and data storage; we are particularly interested in receiving submissions from industry. Topics of interest include (but are not limited to):

- Data layout, organization and metadata
- Caching, replication, and wide area file systems
- Scalable storage systems
- Security in storage systems
- Novel applications for storage systems
- Mobile storage
- Storage system reliability, availability, and integrity
- Storage management for databases
- Network-attached storage & storage area networks
- New storage technologies
- Storage system and device internals
- Compiler support for storage and I/O
- Performance evaluation of storage systems & devices
- File and storage system workloads

For details about paper submissions check the Web site at: *http://www.usenix.org/events/fast*. The submissions deadline is July 13, 2001.

CONNECT WITH USENIX & SAGE

MEMBERSHIP, PUBLICATIONS, AND CONFERENCES

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: +1 510 528 8649
FAX: +1 510 548 5738
Email: <office@usenix.org>
<login@usenix.org>
<conference@usenix.org>

WEB SITES

<http://www.usenix.org>

<http://www.sage.org>

EMAIL

<login@usenix.org>

COMMENTS? SUGGESTIONS?

Send email to <ah@usenix.org>

CONTRIBUTIONS SOLICITED

You are encouraged to contribute articles, book reviews, photographs, cartoons, and announcements to *;login:*. Send them via email to *<login@usenix.org>* or through the postal system to the Association office.

The Association reserves the right to edit submitted material. Any reproduction of this magazine in part or in its entirety requires the permission of the Association and the author(s).

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

;login:

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

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