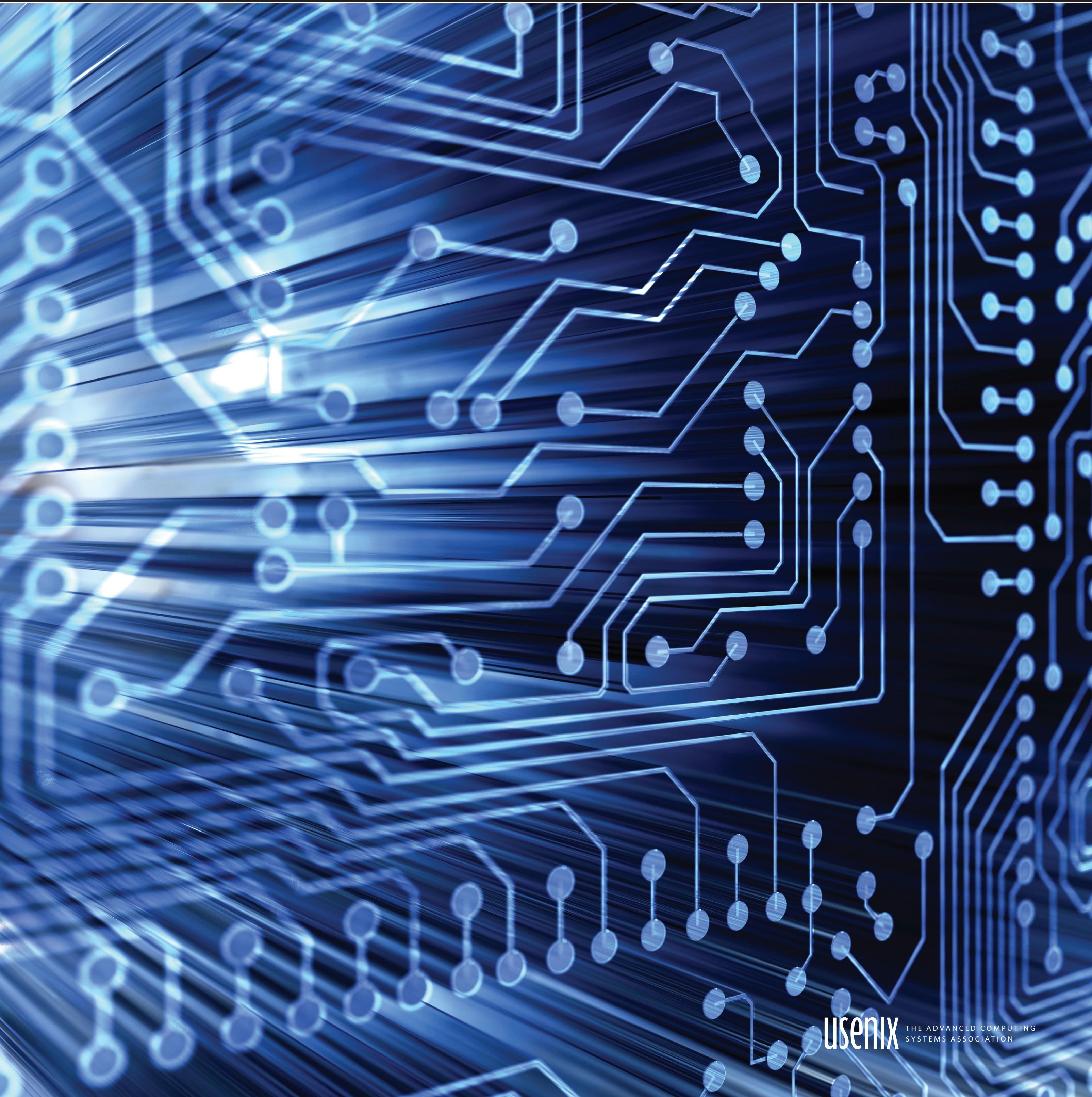
usenix ;login:

FEBRUARY 2011 VOL. 36, NO. 1



usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION usenix UPCOMING EVENTS

9th USENIX Conference on File and Storage Technologies (FAST '11)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGOPS

February 15–17, 2011, San Jose, CA, USA
http://www.usenix.org/fast11

Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '11)

CO-LOCATED WITH NSDI '11

March 29, 2011, Boston, MA, USA
http://www.usenix.org/hotice11

4th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '11)

CO-LOCATED WITH NSDI '11

March 29, 2011, Boston, MA, USA
http://www.usenix.org/leet11

8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGCOMM AND ACM SIGOPS

March 30–April 1, 2011, Boston, MA, USA
http://www.usenix.org/nsdi11

European Conference on Computer Systems (EuroSys 2011)

SPONSORED BY ACM SIGOPS IN COOPERATION WITH USENIX

April 10–13, 2011, Salzburg, Austria
http://eurosys2011.cs.uni-salzburg.at

13th Workshop on Hot Topics in Operating Systems (HotOS XIII)

SPONSORED BY USENIX IN COOPERATION WITH THE IEEE TECHNICAL COMMITTEE ON OPERATING SYSTEMS (TCOS)

May 8–10, 2011, Napa, CA, USA
http://www.usenix.org/hotos11

3rd USENIX Workshop on Hot Topics in Parallelism (HotPar '11)

May 26–27, 2011, Berkeley, CA, USA
http://www.usenix.org/hotpar11

2011 USENIX Federated Conferences Week

June 12–17, 2011, Portland, OR, USA
Events include:

3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '11)

June 14, 2011
http://www.usenix.org/hotcloud11
Submissions due: March 7, 2011

3rd Workshop on Hot Topics in Storage and File Systems (HotStorage '11)

June 14, 2011
http://www.usenix.org/hotstorage11
Submissions due: March 9, 2011

3rd Workshop on I/O Virtualization (WIOV '11)

June 14, 2011

2011 USENIX Annual Technical Conference (USENIX ATC '11)

June 15–17, 2011
http://www.usenix.org/atc11

2nd USENIX Conference on Web Application Development (WebApps '11)

June 15–16, 2011
http://www.usenix.org/webapps11

2011 Electronic Voting Technology Workshop/ Workshop on Trustworthy Elections (EVT/WOTE '11)

CO-LOCATED WITH USENIX SECURITY '11 AND SPONSORED BY USENIX, ACCURATE, AND IAVOSS

August 8–9, 2011, San Francisco, CA, USA
http://www.usenix.org/evtwote11
Submissions due: April 20, 2011

20th USENIX Security Symposium (USENIX Security '11)

August 10–12, 2011, San Francisco, CA, USA
http://www.usenix.org/sec11
Submissions due: February 10, 2011

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

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 a 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.

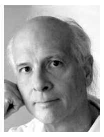
usenix ;login:

FEBRUARY 2011, VOL. 36, NO. 1

OPINION

Musings

RIK FARROW



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

Walking near where I live on a warm winter afternoon, it hit me. Layers. Everywhere I looked there were layers of rocks, leftovers from millions of years ago, now exposed by weathering. How apropos to many of the topics covered in this issue. There are layers in virtualization, layers in file systems, layers in networking, as well as pluses and minuses with having so many layers.

Layers are not necessarily a bad thing. I was once asked to create drop-down, cascading menus using a primitive drawing library. Popping open a menu of choices was fairly easy, as was drawing a second-level menu. I just created a bunch of text rectangles of the correct size, making certain they stacked perfectly. But I also needed to deal with saving the pixels that were present before I drew the menu and with replacing those pixels once a selection was made. Today, you would just use a library, a list of menu items, a corresponding list of functions to call, and everything would get taken care of. In other words, you would take advantage of higher layers of software that made a task that was once difficult easy to do.

Layers

Layers occur everywhere in computers. As I type this sentence, each keypress gets converted into a set of bits, sent over a serial link, received by a character device driver, passed through the line handling code, placed in a queue associated with a particular virtual TTY and copied to the program I am using, which then interprets the character and writes it to an X Window library routine, which eventually calls a kernel routine so that it can display some bits on my screen. And, of course, I am simplifying things quite a lot. But it is the many layers that make what appears to be, and really needs to be, a simple action appear to work in a trivial fashion.

Simple devices have fewer layers. Your microwave may have a menu of items it "knows" how to cook, displayed in funky letters on its display. On such simple systems there are few layers, and a lot of programming effort is required just to spell out BEEF. Move up to a smartphone, and even though the device is smaller, the computing power is immensely bigger, and the layers have grown as well. Then move onto a modern PC running Linux or Windows 7, and the number of layers grows even faster. You might think that a smartphone running Linux or Windows would have just as many layers as a desktop running the same OS, but you'd be wrong. The smartphone is more limited, with a simplified API that programmers are required to use.

Semantic File Systems

Instead of thinking about characters on a screen, let's consider file systems. At the hardware interface, the OS presents sector-sized blocks to devices along with directions about where to write the sector. Modern disk drives may ignore the location directions and just write the sector in the first free block available, keeping track of where the OS thinks the block resides. So even disks have their own layers—internal, hidden levels of indirection.

Now let's get really squirrelly and pop way up the stack to an operating system running within a VM. This virtualized OS "thinks" it is writing to a disk, but really it is sitting atop a hypervisor which may take the block write and convert it into a network write to some remote storage device. From the perspective of the virtualized OS, it is convenient not to have to consider what really happens when the OS writes a sector. But from other perspectives, blindly treating blocks as blocks wastes lots of information.

Including semantics in file systems is not a new idea. File metadata has provided some level of semantics in just about every file system, with the exception of mainframe OSes. But virtualization rips away the assumption that a block written on a disk includes some semantic information, because VMMs today are, for the most part, blind to this information.

Storage companies just love this. I really wondered why EMC bought VMware, until I realized just how much virtualization features, like migration, rely on SANs. And with the semantic information about what is being written lost, the opportunity to do clever things is greatly reduced. Sure, a smart filer can handle deduplication, as data is just data. But from a system administrator's perspective, the blocks on those filers are just blocks. They no longer represent anything meaningful. Instead, the amount of storage required increases.

The Lineup

During the enormous poster session at OSDI (75 posters!), Dutch Meyer managed to catch my eye. Perhaps it was because I knew Meyer from his work as a summarizer, but I think it was really because he and his co-authors are looking at the issue of layers in virtualization in their research. In their article they point out just how much is lost, and how much there is to gain, by preserving file semantics below the level of a VM.

I also met Rob Sherwood during OSDI. Sherwood presented a paper on FlowVisor, a prototype implementation of network slicing that relies on OpenFlow. FlowVisor allows new services to be tested on live networks by partitioning the network based on how traffic is switched. OpenFlow by itself stands to be a game-breaking technology for the operators of large clusters of systems.

During USENIX Security '10, David Barrera proposed sharing work he had done with Glenn Wurster and Paul Van Oorschot on improving a part of IPv6 that has security implications. In IPv6, the lower 48 bits of an address are, by default, the MAC address of the network interface. But that address is supposed to be unique. And that implies that an adversary could track the mobile devices as they move from one IPv6 network to another. Barrera shares their approach to fixing this issue, along with a very nice explanation of IPv6 host addresses.

Mona Attariyan and Jason Flynn (also met during OSDI) share their work on providing an automated way of solving configuration error problems. Their project involves statically tracing execution flow, then monitoring execution until an error occurs. They can revisit the execution, trying out different paths, until they determine which variables, identified with taint, were most likely to have caused the error. Very cool and useful work.

Josh Fiske shares his experience using Linux virtualization from his work at Clarkson University. Fiske takes advantage of layers, by automating the process of spinning up new VMs and configuring them, as well as installing and configuring a set of application packages.

Ole Tange shows off his own software project, GNU Parallel. GNU Parallel is a replacement for xargs with a focus on forking as many processes in parallel as desired, allowing you to take better advantage of multicore or multi-threaded systems. Tange has also designed GNU Parallel to avoid some weaknesses in how xargs processes its arguments, making it an excellent replacement.

David Blank-Edelman gets right into the theme of file systems by exploring some of the included Perl libraries for copying and renaming files. He also takes a look at CPAN modules that go well beyond the basics, such as using FTP, SFTP, SCP, and the wrappers for rsync.

Peter Galvin compares virtualization options in Solaris, AIX, and RHEL. Expanding on the comparison in his December 2010 column, Galvin explores the pluses and minuses of these three enterprise-ready operating systems. Not surprisingly, hardware support does make a difference here.

Dave Josephsen continues his exploration of Ganglia. In this column, Josephsen demonstrates how to write plug-ins in C for the data-collecting daemon, gmond. While writing C programs may not be something everyone feels comfortable with, for often repeated tasks on critical servers their performance cannot be beat. And, as Josephsen points out, using C means that other packages do not need to be installed for this trick to work.

Robert Ferrell explains how the threat of worms like Stuxnet requires us to think outside the box, or at least the comic book, to find new solutions.

Elizabeth Zwicky explains how she can review so many books each issue, tells us about her experience reading eBooks, then presents us with her views of three new books. I take a quick look at a book about building your own PCs, and I like what I see enough to order the recommended list of parts for my new desktop. Sam Stover waxes enthusiastic over a book about lock picking, a great hobby for any geek, as well as a useful skill for physical penetration testers.

This issue includes summaries from OSDI '10. LISA summaries were not complete when I turned this issue in for printing (really!), so they will be out in April 2011. We also have summaries from four workshops, including some excellent advice from the Diversity Workshop to anyone either in grad school or planning to work toward an advanced degree.

I can look out my office window and see the layers in the rock, similar to the Coconino Sandstone and Hermit Shale layers seen in the Grand Canyon [1]. If I move my chair a little, I can see basalt that capped the Mogollon Rim with hard rock from volcanic eruptions millions of years ago. These layers make for spectacular views, as well as supporting the local economy by attracting hordes of tourists.

Our computer systems are composed of many more layers, rapidly deposited over a period of just decades. If our computers performed as slowly as they did in 1969, where a dual-CPU MULTICS system peaked out at six million instructions per second, we wouldn't have so many layers—they would be too performance-intensive.

Layers make programming and, to some extent, using computers much simpler. They also have other implications. More lines of code means more bugs. And deep software stacks also allow operating system vendors to lock in customers, as the only way to bypass these layers is to port software to other operating systems' layers. I've been thinking and writing about the implications of these layers for many years, and you can take a look at where I hope we are going both in security and in dealing with these layers [2].

References

[1] Grand Canyon Layers: http://www.bobspixels.com/kaibab.org/geology/gc_layer.htm and http://education.usgs.gov/schoolyard/IMAGES/GrandCanyon.jpg.

[2] Rik Farrow, "Software Insecurity," *IQT Quarterly,* vol. 2, no. 2: http://www.rikfarrow.com/IQT_Quarterly_2010.pdf.

Namespace Management in Virtual Desktops

DUTCH T. MEYER, JAKE WIRES, NORMAN C. HUTCHINSON, AND ANDREW WARFIELD

Dutch Meyer is a PhD student, under the supervision of Andrew Warfield, at the University of British Columbia. His research investigates the impacts of virtualization on network-available storage systems.
dmeyer@cs.ubc.ca

Jake Wires received his MS in computer science from the University of British Columbia. He currently works in the Datacenter and Cloud Division at Citrix, where his focus is storage virtualization.
Jake.Wires@Citrix.com

Norman Hutchinson is an associate professor at the University of British Columbia. His research interests center on programming languages and operating systems, with particular interests in object-oriented systems, distributed systems, and file systems.
norm@cs.ubc.ca

Andrew Warfield is an assistant professor at the University of British Columbia. He advises students on a wide range of topics, including virtualization, storage, and security.
andy@cs.ubc.ca

Even as virtualization has promised to ease cluster scale and management, it presents system administrators and storage system designers with opaque blobs of data that represent entire virtual volumes. In these environments, application and file-level semantics are abandoned long before data reaches the disk. Our research borrows from past work and is creating virtual storage interfaces that preserve file-level information in order to improve the management and efficiency of storage.

Virtualization has been widely used to reduce operational costs of mid- and large-scale server farms. Now it is making headway in desktop computing, where it has played a central role in recent efforts to migrate users from individual workstations to centrally administered servers. Virtual Desktop Infrastructure (VDI) is the latest manifestation of the well-known thin-client paradigm. It attempts to lure end users—who have previously been reluctant to embrace thin clients—by providing a computing environment almost identical to the familiar desktop PC. There are good reasons to believe this approach is working.

Gartner predicts that 40 percent of all worldwide desktops—49 million in total—will be virtualized by 2013 [2]. Already, many organizations have deployed VDI to tens of thousands of users [6]. This surge is being driven by administrators who have long seen the value of centralizing PC resources. They find that a significant economy of scale comes from the reduced operating costs provided by a VDI environment. There are, however, big challenges posed by such large and centralized installations, particularly with respect to storage.

In current VDI implementations, virtual disks are stored as opaque files on a central network server. File formats like Microsoft's VHD and VMware's VMDK encapsulate entire disk images using a read-only base image (or Gold Master) as a template disk. Modifications to the base image are stored in one or more separate overlay images allocated for each VM. This allows the rapid creation of new VMs with minimal initial overhead. But as VMs mature, their overlay images diverge, leading to increased storage consumption and maintenance burden. The block level approach used by these file formats, while simple to implement, lacks the contextual information necessary to begin addressing this divergence problem. Administrators are presently faced with the choice of either allowing images to diverge without bound, resulting in a serious management problem for tasks like upgrade, or resetting images to the original master at daily or weekly frequencies, which frustrates users who desire to install their own software.

This information-poor block interface also extends through much of the storage stack in enterprise VM environments. In a traditional PC the transformation from file to block requests occurs very low in the stack, so many storage features operate at the file level. However, in virtualization environments this transformation occurs at the top. Below the guest VM's block level, the VMM will map the virtual drive to a file format for virtual disks. Since a shared storage system is required for live VM migration and recovery in case of a physical server failure, the block requests must then travel over the network. They are processed by a centralized storage system that aggregates many physical drives, often storing the images on yet another file system. In total, an enterprise virtualization storage stack will have easily twice the distinct layers of a desktop PC, most of which are unaware of the original file semantics.

Semantics Lost

This loss of semantics limits file-oriented performance optimizations. For example, it is often the case that different VMs on the same physical host read and store identical files that happen to be at different logical disk offsets. Common storage optimizations around caching, deduplication, and placement—often implemented within OS and file system code with the benefit of object boundaries—must be approximated at the block layer.

Block semantics also diminish the administrator's ability to administer. In time-sharing systems, administrators could see user files and their accesses. If a configuration file was incorrect, it could be inspected and even changed. If a file management policy was not being followed, it could be detected directly. Administrators could scan the whole system for all files of a given type or name. The corresponding view in a contemporary virtualization system is a stream of block requests passing to an opaque virtual disk file.

This can make simple tasks, such as changing a user's security settings in Internet Explorer, unnecessarily complicated. The administrator can use Remote Desktop or Terminal Services to modify the machines directly, but must access each VM individually. With scripts they may do the job faster, but this requires knowledge of specialized syntax. If the VMs can be turned off, the administrator can mount the disks for inspection. Or perhaps she could email the VM's owner and ask politely for help. These restrictions are largely consequences of using opaque containers and protocols. They seem ridiculous, given that the files are already being hosted on a single shared storage system.

Users do not directly see these layer intricacies, of course, but they also don't get many explicit benefits. Instead, they are forced into an anachronism—PC-era isolation, despite mainframe-style consolidation. Consider file sharing in this environment: users can copy files to a network drive, create a file server on their local VM, or email files as attachments. Those options seem natural for a PC, but in a VDI any shared data is already hosted by a dedicated file server. The barriers to collaborating on this file are vestiges of an era when hard drives were physically isolated—and virtual disks preserve these barriers without offering any real benefits. A better approach would be to share the file without creating copies and without the complexity and overhead of creating what is in effect a proxy file server.

Obviously, today's file systems were not designed for use in virtual environments. But what would a new, virtualization-aware file system look like? We feel that some very good ideas can be reappropriated, refurbished, and redeployed from past research to help address these issues.

Namespace Composition

Many of the these problems stem from the forfeiture of file semantics at the top of the predominantly block-oriented virtual storage stack. However, there's no fundamental reason that most of the storage stack can't instead use a file interface. File-based network protocols like CIFS and NFS are in widespread use, and virtual disk management based on a file interface was introduced with Ventana in 2006 [3].

Like Venti [4], which inspired it, Ventana used a single global store for all files. Individual disk images were created by selecting the necessary files from this pool, and shared access was protected with copy-on-write. Conceptually, this composition could be considered similar to a very fine-grained use of UnionFS. Unfortunately, the Ventana implementation never saw much use or distribution.

Systems like Ventana require that the file interface extend all the way from the guest operating system to the network storage system. In practice, there are technical limitations that make this difficult. Most notably, the Windows boot process requires a block device, which precludes using a file interface. However, this problem is not impossible to overcome. Linux already provides NFS boot, which would be a sufficient solution for Windows. In our own lab we use a custom Windows file system driver to transition to a CIFS interface during boot, which has much the same result. Although this approach requires synthesizing a block interface for the early boot stages, it is much simpler than recreating file semantics from a stream of block requests [1].

Whether one uses CIFS, NFS, or another protocol in place of a block interface, the benefits are significant. At the VM level, it removes the need for any type of block-level processing. The VMM benefits from the file interface, because it opens up opportunities for caching when multiple VMs are reading from the same files. The network interface to centralized storage can also be file-based, possibly NFS or CIFS, which is easier to reason about than iSCSI and available on more affordable hardware. Finally, the cluster administrator is put back in the position of dealing with file access and management. This helps in technical administration, such as troubleshooting a client misconfiguration and managing diverging disks. It opens the door to replacing inefficient per-client services, such as virus scanners, with centralized alternatives. This could be used to solve the "Antivirus Storm" problem, where a number of idle clients, unaware that they are all sharing storage resources, engage their antivirus software and place stress on centralized storage.

A file interface would also help administrators simply understand what their system is doing. Often, it is too easy for an administrator to be unaware that a considerable portion of their storage resources is handling completely unnecessary tasks such as defragmenting virtual disks or writing IE temp files over the network to highly redundant and expensive storage. Given the current structure of block requests and opaque disk images, such waste can go unnoticed.

Virtual Directories

Restoring file semantics to the storage stack may not be enough. Many believe that today's file systems are already too complex to manage [5], so navigating a large cluster of such hierarchies would probably challenge an administrator. In other environments, virtual directories have been the subject of some interest in combating complex file hierarchies. For virtualization, we think the idea could be extended to provide even more benefits.

The virtual directory mechanism traces back to Gifford's 1991 paper proposing semantic file systems, and perhaps even earlier, to UNIX systems which first displayed devices through a file interface. In the current context, complex searches for files can be represented persistently as a virtual directory. This allows users to create directories that display semantic information rather than filesystem location information. As an example, users might want their music collection displayed in the file system as a directory containing all files from a certain artist, regardless of the hierarchical location of those files.

Combined with an enterprise-wide storage system like the one proposed above, this mechanism would be a powerful management and collaboration aid. For end users, this would provide support for three fundamental workflows.

First, for users operating on multiple VMs, the process of circumventing the unnecessarily strong barriers between VM file systems could be eliminated. Rather than creating a new file server or copying the file over a network, a user could merely request that the file be mapped into both file systems. Of course, there are complex notions of user-identity and access control that need to be addressed. Similar problems have been addressed in the past [7], although in different contexts, making this a ripe area for further research.

Second, to facilitate information finding, one could use virtual directories for persistent queries, such as "find me all spreadsheet files from the accounting group." Currently, one could search for such files, but the illusion of decentralized storage requires that users first locate the appropriate network servers and then aggregate results from multiple sources. Furthermore, persistent queries are more powerful than searches, because they can stay current with publish/subscribe notifications.

Third, virtual directories and namespace composition could work together to empower file publishers, while simplifying the steps required to collaborate on a file. Rather than sending an email to relevant parties containing a network address or copy of the file, a publisher could (with the help of a Microsoft Outlook plug-in) include a capability to access the file in an email. Shared access to this file could be coordinated with copy-on-write, writer locks, or integrated version control software. No doubt, each of these options should probably be available, since different collaboration models are appropriate for different files. In any case, this approach shifts the age-old problem of maintaining and merging multiple file copies from an ad hoc management approach to one that is consistent and centralized in a single enterprise-wide file system. Certainly, merging will occasionally be required, but that's okay. Most non-expert computer users are already familiar with the need to merge files, since they do this over email already. What they aren't aware of is the fact that other management options exist for this problem.

Virtualization administrators, similarly, would benefit greatly from virtual directories. Aggregating files from VM file systems into a single namespace could provide opportunities to view a user's files in terms of their similarity or dissimilarity to those of their peers. The latter may provide opportunities to locate misconfigured machines via outlier detection. The former would allow administrators to considerably collapse the large space of files in a large enterprise. Virtual files may also be useful: consider, for example, creating a master log for a cluster by reading Windows Events through the logging facilities of each VM and merging them. Again, these mechanisms provide new opportunities to diagnose problems or to catch warnings before they become problems.

Towards a Virtualization-Friendly File System

Cluster-wide virtualization is disruptive to internal network, compute, and storage infrastructure. However, corresponding changes have yet to propagate to our file systems. Our research experiences suggest to us that deep stacks without semantic information lead to misconfiguration and inefficiency. Namespace composition offers one organizing principle, but many issues remain. Simplicity and platform-agnosticity at the block level have served us well, but ensuring those traits in file- and object-based protocols is more difficult. There are also questions about layering in the storage stack. It's not yet clear how much functionality should be placed in the client file system. Alternatively, the VMM's role in hosting many guest file systems suggests performance benefits to co-locating similar VMs and providing features at that level. Then again, centralizing storage in back-end filers is appealing for simplicity.

For end users, there is already widespread awareness that we need better tools to organize and navigate data, but virtualization may be important in shaping those solutions. Virtualized desktops and datacenters act much like PCs, but their architecture is closer to time-sharing systems. We need to find a balance between isolation and ease of sharing in these environments. Even for individual users, creating VMs in order to isolate known-good OS and application configurations is beneficial. However, sharing and synchronizing files between these isolated systems is not easy or robust. With support for file sharing between VMs, virtual directories offer a compelling alternative. Semantic file organization may also improve our ability to find what we want among larger collections of file systems. However, that approach implies that namespace location is no longer a useful guide for the physical disk locations for our files.

While many open questions remain, we see great promise in these semantic-rich storage stacks and file system structures. Administrators need a file-oriented view of storage to efficiently understand and assist their users, while users may see benefits to novel file organizations and simpler work flows through explicit sharing. Storage is, for many good reasons, slow to change, but if we are to address the shifts in PC and cluster design, changes are on the way.

References

[1] A.C. Arpaci-Dusseau and R.H. Arpaci-Dusseau, "Information and Control in Gray-Box Systems," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles,* 2001, pp. 43–56.

[2] C. Pettey and H. Stevens, "Gartner Says Worldwide Hosted Virtual Desktop Market to Surpass $65 billion in 2013," March 2009: http://www.gartner.com/it/page.jsp?id=920814.

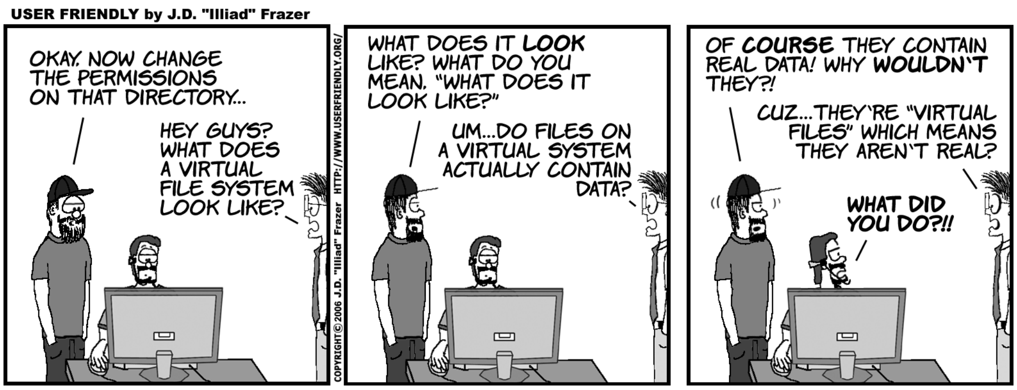
[3] B. Pfaff, T. Garfinkel, and M. Rosenblum, "Virtualization Aware File Systems: Getting Beyond the Limitations of Virtual Disks," in *3rd Symposium on Networked Systems Design and Implementation (NSDI '06),* 2006.

[4] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage," in *Proceedings of the Conference on File and Storage Technologies,* 2002, pp. 89–101.

[5] M.I. Seltzer and N. Murphy, “Hierarchical File Systems Are Dead,” *12th Workshop on Hot Topics in Operating Systems (HotOS XII)*: http://www.usenix.org/event/hotos09/tech/full_papers/seltzer/seltzer.pdf.

[6] VMWare, Customer Case Studies by Product, November 2010: http://www.vmware.com/products/view/casestudies.html.

[7] E. Wobber, M. Abadi, M. Burrows, and B. Lampson, “Authentication in the Taos Operating System,” in *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles,* 1993, pp. 256–269.


USER FRIENDLY by J.D. "Illiad" Frazer
OKAY. NOW CHANGE THE PERMISSIONS ON THAT DIRECTORY...
HEY GUYS? WHAT DOES A VIRTUAL FILE SYSTEM LOOK LIKE?
COPYRIGHT© 2006 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
WHAT DOES IT LOOK LIKE? WHAT DO YOU MEAN, "WHAT DOES IT LOOK LIKE?"
UM...DO FILES ON A VIRTUAL SYSTEM ACTUALLY CONTAIN DATA?
OF COURSE THEY CONTAIN REAL DATA! WHY WOULDN'T THEY?!
CUZ...THEY'RE "VIRTUAL FILES" WHICH MEANS THEY AREN'T REAL?
WHAT DID YOU DO?!!

Safely Using Your Production Network as a Testbed

ROB SHERWOOD

Rob Sherwood is a Senior Research Scientist at Deutsche Telekom Inc.'s R&D Lab in Los Altos, California. Additionally, Rob is a member of the Clean Slate Lab at Stanford University.
r.sherwood@telekom.com

FlowVisor is a prototype implementation of *network slicing,* a technique for allowing production and experimental network protocols to safely share the same physical infrastructure. FlowVisor relies on OpenFlow, a new protocol for managing switches and routers that controls network traffic using patterns found in packets.

Network administrators must strike a careful balance between providing a solid, reliable network and a network that has the cutting-edge services demanded by its users (e.g., multicast, IPv6, IP mobility, or something more radical such as a traffic load balancing service). This is particularly challenging in a research or university setting, as some of the requested services are themselves experimental and therefore already disruption-prone in nature. Further complicating this issue are up-and-coming technologies like OpenFlow [3] that allow network administrators, operators, and researchers to program custom services into the network. OpenFlow provides programmatic control of how packets are forwarded, and with it researchers are prototyping novel network services [1, 2]. But while OpenFlow allows interesting new network services, this additional freedom comes at the cost of additional potential sources of network instability.

Common practice is to deploy new services in a smaller testbed or isolated VLAN and then, after some time has passed and confidence has been built, transition the service to the production network. This approach has two main shortcomings. First, typically for reasons of cost, testbeds rarely have the same size, number of users, heterogeneity, or, more generally, complexity as the real production networks. As a result, it's not uncommon for a service to work correctly in the testbed but still have problems in the production network. Second, once the service has passed its testbed evaluation, there is typically not an incremental and controlled (e.g., user-by-user) way of deploying the service in the production network. For example, on most routers, multicast support is a binary feature: it is either enabled or disabled, so some error in the service could affect all users, not just the ones that elect to use multicast.

Our approach, as demonstrated by our first prototype, called FlowVisor [7], is to divide the network into logical partitions, called *slices*, and to give each service control over packet forwarding in its own network slice. Users then *opt in* to one or more network services: that is, they delegate control of subsets of their traffic to specific services. The existing production network services (e.g., Spanning Tree, OSPF, etc.) run in a slice, and by default, all users are opted into the production slice. Critically, the FlowVisor ensures strong isolation between slices: that is, actions in one slice do not affect another slice.

For example, a network administrator, Alice, could use FlowVisor to divide her network into three slices (Figure 1). The first slice the administrator keeps for herself as a *production* slice and in it runs standard, well-vetted network protocols, e.g., OSPF, Spanning Tree, or basic MAC-address learning/switching using available open source software such as Quagga [6] or NOX [4]. The administrator then delegates the second and third slices to two network researchers, Bob and Cathy. Bob is developing a network service optimized for high throughput, and Cathy is running a service optimized for low packet-loss. Then the network's users are able to pick and choose (e.g., via an authenticated Web page) which slices control their traffic. For example, users not trusting the research slices might elect to have all of their traffic controlled by Alice's production slice. By contrast, an early adopter, Doug, might be more interested in the new slices and elect to have his gaming traffic be controlled by Cathy's slice, his HTTP traffic be controlled by Bob's slice, and the rest of his traffic (e.g., VoIP) controlled by Alice's production slice. The key point is that the FlowVisor would enforce isolation between these slices so that if one slice had a problem (e.g., created a forwarding loop), it would not affect the other slices even though they shared the same physical hardware. Further, new services would be deployed more incrementally, i.e., user by user, creating a more graceful service introduction process.

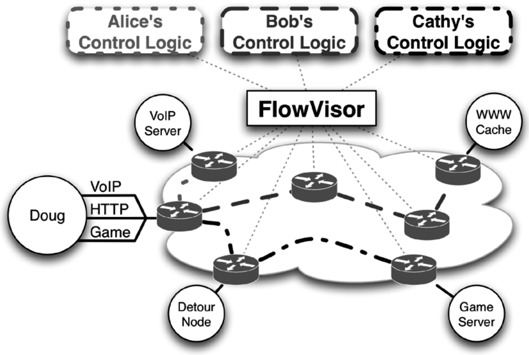


Figure 1: Network slicing allows multiple network protocols to run safely together on the same physical infrastructure.

Slicing Control and Data Planes

Network slicing is a way of allowing multiple services to share control of the same physical network resources. At a high level, the internals of modern switches, routers, base stations, etc. are typically divided into a control plane and a data plane (also called the slow path and the fast path, respectively). The control plane is a collection of software applications typically running on a general-purpose CPU, where the data plane is one or more application-specific integrated circuits (ASICs). The control plane is responsible for formulating higher-level forwarding rules of the form "if a packet matches *pattern,* then apply *actions,"* which are then pushed down to, and enforced by, the data plane. The exact nature of the pattern and actions varies by device: for example, on a router, the pattern might be a CIDR-prefix and actions would be "decrement TTL and then forward out port 14." The important point is that there is a communications channel between the control and data planes.

Similar to how a hypervisor sits between multiple virtual machine operating systems and the underlying hardware, network slicing is a layer between the multiple control planes and the underlying data planes. Each slice runs its own logical control plane and makes its own packet-forwarding rules. The slicing layer ensures isolation between slices by verifying that forwarding rules from different control planes/services do not conflict. Note that once a rule is pushed into the data plane, packets are forwarded at full line speed, so network slicing has no packet forwarding performance penalty. Network slicing can even isolate bandwidth between slices by mapping a slice's actions onto a per-interface quality of service (QoS) queue.

Our network slicing implementation, FlowVisor, is implemented on top of OpenFlow. OpenFlow is an open standard for controlling data planes of existing network hardware. An already existing and deployed switch or router can be OpenFlow-enabled with a firmware upgrade. Once a network device supports OpenFlow, network administrators or researchers can write their own control logic to make low-level packets forwarding decisions, e.g., writing a new routing algorithm. In OpenFlow, the control plane is moved off the network device to an external *controller* (typically, a commodity PC); the controller talks to the data plane (over the network itself) using the OpenFlow protocol. The controller is simply a user-space process that speaks the OpenFlow protocol to OpenFlow-enabled devices.

The FlowVisor acts as a transparent OpenFlow proxy, sitting between the switch and a set of OpenFlow controllers. The FlowVisor intercepts messages as they pass between switch and controller, and rewrites or drops them to ensure that no service violates its slice configuration. FlowVisor's configuration file specifies which sets of resources are controlled by each slice, including topology and bandwidth, and which classes of packets each slice manages.

Deployment and Scalability

Even though it is still a research prototype, FlowVisor has been deployed in various capacities on eight campuses and on one national backbone provider's network. At Stanford University, for example, FlowVisor runs on two different VLANs of the physical production network, including 15 wired switches and 30 wireless access points. It has been in place for over one year and slices the network that the authors use for their daily email, Web traffic, etc. On each of the seven other campuses (including Georgia Tech, University of Washington, Clemson University, Princeton, Rutgers, the University of Wisconsin, and the University of Indiana), FlowVisor manages a testbed network, but there are plans underway to move to the production network. Additionally, the National Lambda Rail (NLR)—a national backbone provider—has deployed OpenFlow and FlowVisor on a dedicated five-node nationwide circuit.

Recently, FlowVisor-sliced networks were showcased at the ninth GENI Engineering Conference. Five distinct OpenFlow-based projects ran simultaneously on the same physical network nodes, as contributed by the eight campuses and NLR. This demonstration is evidence that FlowVisor-style network slicing has the necessary isolation capabilities to test new research on commercially available production equipment.

We also evaluated the FlowVisor in terms of its scalability and overhead. The FlowVisor's total workload is the product of the number of switches, times the average number of messages per switch, times the number of slices, times the average number of rules per slice. Our Stanford deployment does not produce a measurable load on our deployed FlowVisor, so we instead created a synthetic workload that is comparable to the *peak* rate of a published real-world 8000-node enterprise network [5]. Using this synthetically high workload, the FlowVisor maintains under 50% CPU utilization on a single process on a modern server. In terms of performance, we find that FlowVisor adds an average of 16 milliseconds of latency for setting up a new flow and no overhead for additional packets in a flow. Thus, we believe that a single FlowVisor instance could manage a large enterprise network with minimal overhead.

Conclusion

FlowVisor-style slicing combined with OpenFlow offers potential relief to operators and researchers looking to deploy new network services without sacrificing network stability. Our current efforts are focused on expanding our deployments and better "bullet-proofing" isolation between slices. The source code for FlowVisor is freely available from http://www.openflow.org/wk/index.php/FlowVisor.

References

[1] D. Erickson et al., "A Demonstration of Virtual Machine Mobility in an Openflow Network," in *Proceedings of ACM SIGCOMM (Demo),* August 2008, p. 513.

[2] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10).*

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, April 2008, pp. 69–74.

[4] http://www.noxrepo.org.

[5] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A First Look at Modern Enterprise Traffic," in *Proceedings of the Internet Measurement Conference 2005,* pp. 15–28.

[6] http://www.quagga.net.

[7] R. Sherwood, G. Gibb, K.-K. Yap, M. Cassado, G. Appenzeller, N. McKeown, and G. Parulkar, "Can the Production Network Be the Test-Bed?" in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI '10),* pp. 1–14.

Back to the Future: Revisiting IPv6 Privacy Extensions

DAVID BARRERA, GLENN WURSTER, AND P.C. VAN OORSCHOT

David Barrera is a PhD student in computer science at Carleton University under the direction of Paul Van Oorschot. His research interests include network security, data visualization, and smartphone security.
dbarrera@ccsl.carleton.ca

Glenn Wurster completed his PhD in computer science (2010) at Carleton University under the direction of Paul Van Oorschot. His interests include software security, system administration, operating systems, and Web security.
gwurster@scs.carleton.ca

Paul Van Oorschot is a professor of computer science at Carleton University in Ottawa, where he is Canada Research Chair in Authentication and Computer Security. He was Program Chair of USENIX Security '08, Program Co-Chair of NDSS 2001 and 2002, and co-author of the *Handbook of Applied Cryptography* (1996).
paulv@scs.carleton.ca

Network stacks on most operating systems are configured by default to use the interface MAC address as part of the IPv6 address. This allows adversaries to track systems as they roam between networks. The proposed solution to this problem—IPv6 privacy extensions—suffers from design and implementation issues that limit its potential benefits. Our solution creates a more usable and configurable approach to IPv6 privacy extensions that helps protect users from being tracked.

With more people adopting IPv6, some features of the protocol are slowly being explored by a small user-base. Security issues related to IP packet fragmentation and malicious route headers [4] have been identified, and new RFCs addressing those issues have been published (e.g., RFC 5095 and RFC 5722). Over many years, the iterative process of identifying flaws and creating fixes led to IPv4 becoming a stable and mature protocol. Since IPv6 is much newer and only now being broadly deployed, many of its features have not enjoyed broad testing or security analysis. In this article we concentrate on one such feature: IPv6 privacy extensions.

IPv6 provides the option for clients to assign themselves an IPv6 address based on a 64-bit prefix periodically broadcast by a local server, and a 64-bit value derived from the network interface identifier—usually the MAC address of the network card. Having a globally routable IP address which includes (and therefore reveals to remote servers) the MAC address of a client was regarded as a potential privacy issue, leading to the development of IPv6 privacy extensions (RFC 4941). Through the use of these extensions, a host can generate and assign itself a partially randomized (but still valid) IP address at fixed intervals, allowing connectivity without revealing its MAC address. The existing IPv6 privacy extensions are not only important for personal privacy, but also for hiding information which can otherwise allow wide-scale targeted malware attacks such as Internet-scale hit lists.

Paradise Lost

The initial IPv6 address design choices have had a detrimental impact on privacy. The proposed privacy extensions can also fail to provide the benefits they were designed for. Having recorded multiple IPv6 addresses, an adversary can trivially map two or more of these addresses to the same client, sometimes even when privacy extensions are in use. In practice, the adversary could be a corporation wishing to provide targeted services only to users that fit a specific profile (e.g., users who have visited more than three coffee shops in the past week or have been at five airports in the past month). Other adversaries may include governments or eavesdroppers who wish to follow users as they roam through multiple locations. While the tracking of users in IPv6 is partially addressed by the IPv6 privacy extensions, the specification has a number of design issues which can cause implementations to fall short of the goal of keeping the client-to-IP-address mapping private across locations.

Without privacy extensions, tracking is possible because the last 64 bits of a client's IPv6 address are constant: if a client has IPv6 address 1::2345:6789 while on network 1, the client may have IPv6 address 2::2345:6789 when using network 2. This provides the means to track users as they move between IPv6-capable networks. Existing privacy extensions essentially randomize the last 64 bits every *x* seconds.

Paradise Regained

We propose a new technique and prototype implementation for generating private IPv6 addresses. Our proposal differs from current IPv6 privacy extensions in that it can be configured for different privacy requirements and is capable of providing private addresses even if an administrator has configured the network to deter their use. Our proposal provides as much privacy as IPv4 and has minimal overhead. We also describe the implementation of a Linux kernel prototype of our proposal.

In this article we identify issues with the current state of IPv6 privacy extensions that could lead to a downgrade attack, enabling eavesdroppers to track IPv6 users as they move through networks. We also identify issues with currently deployed implementations of IPv6 privacy extensions in modern operating systems, and we propose a more flexible and robust algorithm for generating private IPv6 addresses.

IPv6 Background

Before we explain the details of our proposal, we will review relevant terminology and background on how clients are assigned IPv6 addresses and on the originally proposed privacy extensions. We will use the generally accepted terminology.

- **Prefix:** the first (most significant) 64 bits of an IPv6 address. A prefix can be learned through periodic router advertisements, assigned by DHCPv6, or self-assigned (e.g., for loopback and link local addresses).
- **Interface identifier:** the least significant 64 bits of an IPv6 address. The prefix and interface identifier together fully specify an IPv6 address.
- **Preferred lifetime:** a lifetime associated with a particular IPv6 address during which the address should be used to initiate connections. Once the lifetime expires, the IPv6 address is deprecated, but still active for the remaining open connections. The address remains deprecated until the valid lifetime expires.
- **Valid lifetime:** a lifetime associated with a particular IPv6 address. When it expires, the IPv6 address is removed from the network interface by the kernel and no longer used.

Obtaining IPv6 Addresses

Clients can obtain IPv6 addresses through one of three methods: (1) the user manually assigns a valid IPv6 address to an interface; (2) a periodically advertised prefix is prepended to the self-generated interface identifier; or (3) a DHCP server is queried and the received response used. We review methods (2) and (3).

STATELESS ADDRESS AUTO-CONFIGURATION

IPv6 provides a method for clients to automatically assign themselves a valid IPv6 address based on periodically broadcast router advertisements. In a typical setup, a router broadcasts the IPv6 prefix that all clients should prepend to their auto-configured interface identifier. In 1998 the authors of RFC 2462 suggested that clients use their network MAC address in the generation of the interface identifier. The rationale was that this provided sufficient uniqueness and would require no persistent storage. Nine years later, RFC 4862 removed the MAC address suggestion, allowing hosts to choose their own method for generating interface identifiers. The interface identifier is always appended to the network prefix, after which the Duplicate Address Detection (DAD) algorithm is run by the client to ensure that the address is unique to the network segment, and therefore globally unique (as prefixes are also unique).

In cases where a MAC address is used as the interface identifier (still currently the default behavior of Linux and Mac OS), the IPv6 address reveals information which can be used to identify the client hardware. This ability to determine the hardware configuration of a machine may lead to additional information about the client being revealed on the network. For example, Mac OS runs on a specific underlying hardware platform, allowing the identification of Apple users based only on MAC addresses. This ability to determine the characteristics of a client through the MAC address can be used in a targeted attack on a user or organization (e.g., sending a malicious PDF that only exploits Mac OS). Bellovin et al. [3] argue that the MAC address could also be used by IPv6 worms to target specific hosts.

INADVERTENT IPV6 USERS

We define *inadvertent IPv6 users* as users who unknowingly use IPv6 to connect to remote servers. While the vast majority of Internet users currently use IPv4, modern OSes attempt to use IPv6 by default when resolving hosts. In a typical network, connecting (and therefore revealing the source IP address of the connection) to a remote server over IPv4 will not typically allow the server to track the individual or identify the network hardware. Connecting to the same server over IPv6 may reveal sufficient information to track the individual and identify hardware. Because stateless address auto-configuration does not depend on additional client software (other than an updated kernel), it is likely to cause inadvertent IPv6 use. This increases the importance of IPv6 privacy extensions that truly provide protection and information hiding.

DHCPV6

With the addition of stateless address auto-configuration for IPv6, hosts can obtain network information and learn how to route packets without installing additional software. While this may seem ideal from a network management standpoint (e.g., set up a route prefix advertisement daemon and IPv6 just works), there may be other configuration parameters needed by hosts in order to actually communicate with external hosts. These parameters will vary from network to network, but some include WINS, NTP, NETBIOS, and DNS.

There are cases where administrators may choose to replace stateless auto-configuration with DHCPv6, or use both simultaneously. When using DHCPv6 for address assignment, the server keeps track of assigned addresses and the hosts using them, as DHCP did in IPv4. When using both, a host obtains its IPv6 address through stateless auto-configuration and other information through a server on the local network. The issue of tracking clients using IPv6 is specific to those who obtain an address through stateless address auto-configuration.

Original Privacy Extensions

IPv6 privacy extensions for stateless address auto-configuration were proposed specifically to address privacy concerns with having a static and globally unique interface identifier. Concerns that a well-placed sniffer (or prolific ad network) might track users as they roam through different networks are partially mitigated by privacy extensions through using periodically changing random interface identifiers. RFC 4941 specifies the algorithm used to generate a random identifier, as well as when to update it. As shown in Figure 1, a hash function (MD5 is suggested in the RFC) is used to generate the interface identifier. The first 64 bits of output are used as the interface identifier, while the last 64 are stored for the next iteration of the algorithm, which takes place every *x* seconds (or when a duplicate address is detected by the client). The first iteration of the function uses a random value as the history value.

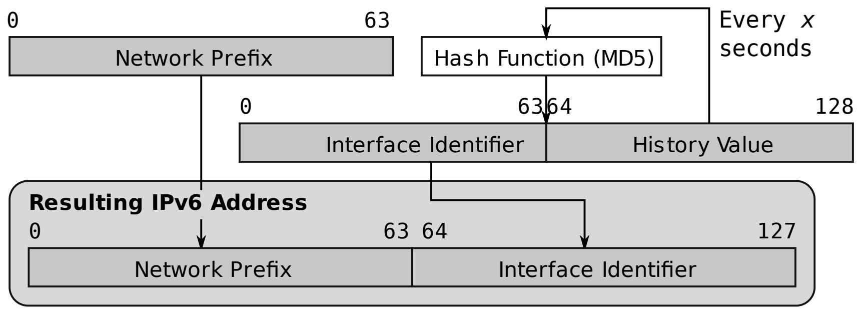


Figure 1: Original privacy extension address generation

The current specification has two important limitations. The only configurable parameter is the interval at which new random interface identifiers are generated. The default interval is to generate a new identifier every 24 hours. This still allows a user moving between two or more IPv6 networks in a 24-hour window to be tracked by an adversary (since the client's interface identifier will not change during this time, even if the network prefix does). The expert user can configure the regeneration interval to be smaller, at the expense of no longer maintaining long-lasting connections (e.g., SSH or movie downloads).

The intervals are dependent on the configuration of the network. If a user has configured the interval for regeneration of addresses to be small, but the network advertises smaller intervals, the smallest takes precedence. This means that if the network is configured to advertise prefixes with valid lifetimes of 60 seconds, a user with privacy extensions enabled will generate a new and different IPv6 address roughly every 60 seconds. This will severely impact user experience: no connection made will last more than 60 seconds.

The latest RFC for privacy extensions also specifies that system implementers should add an option for the user to enable or disable random interface identifiers on a per-prefix basis. This is similar to our proposal in that a new full IPv6 address is generated when the prefix changes (the user changes networks), but differs in that they rely on the client to maintain a list of networks for which privacy extensions should be enabled (or disabled) and do not use the prefix directly in the generation of random interface identifiers.

New Privacy Extensions Proposal

In this proposal we focus on protecting clients who configure their IPv6 address through router advertisements from being tracked as they move between IPv6-enabled networks. We do not address clients configured through DHCPv6 or clients with static IPv6 addresses. We assume that each IPv6 network visited by a client is associated with a distinct prefix (routing problems result if two networks share the same IPv6 prefix).

We assume that the attacker does not have access to the LAN segment, and hence cannot associate IPv6 addresses with MAC addresses, but we do not assume that the network administrator is totally benign. We assume the network administrator is capable of modifying router advertisements, forcing users to renew their IPv6 address often. We assume an attacker attempting to track the client can see traffic generated with each IPv6 address the client uses. We do not attempt to protect against tracking clients using higher-level protocols such as HTTP [5].

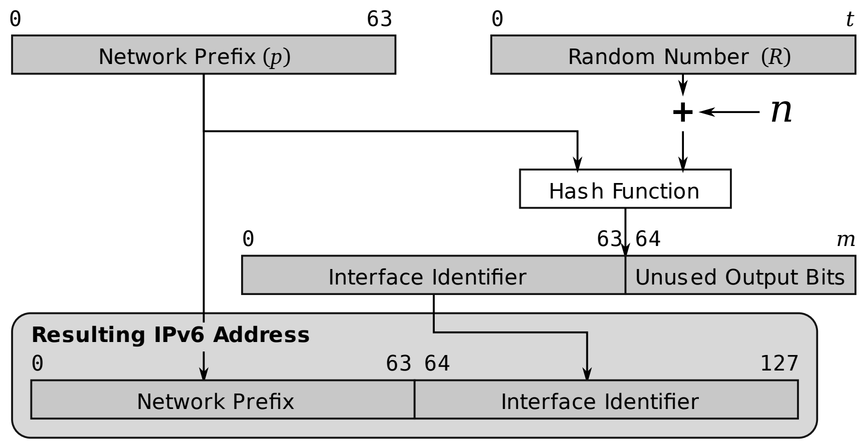


Figure 2: Generation of new IPv6 addresses

The new proposal is for clients to generate IPv6 address interface identifiers *(I)* through hashing the IPv6 prefix advertised by the router advertisement daemon *(p)* with a t-bit random number *(R)* incremented by *n* (in order to resolve duplicate addresses). *R* is generated locally and does not leave the client. The generated interface address is composed of the first 64 bits of the resulting m-bit hash value, as illustrated in Figure 2.

$$I = H(\, p \mid (R + n)\,)$$

We require a pre-image resistant hash function H [8] so that given the prefix and interface identifier, an attacker cannot determine R. Keeping R hidden prevents an attacker from determining that two distinct IPv6 addresses correspond to the same client. There are no compatibility or interoperability concerns should two clients choose to use different hash functions in generating the interface identifier.

To ensure that a new I is generated for every new network prefix (which is not possible in the current privacy extensions), p is included in the hash. There are several options regarding when to change R, allowing the client control over when to start using a new interface identifier. To prevent known attacks [11, 8] against guessing R, its length (t) should be sufficient (e.g., $t = 128$ or 256 bits should certainly be enough), and it should be set from a cryptographically secure random number generator.

Should a client discover (through duplicate address detection) that it is attempting to use the same generated IPv6 address as another client on the local network (an unlikely scenario), the client should generate a new I (and hence IPv6 address) by incrementing n and recomputing the hash. The client should reset n to 0 on reboot and whenever p or R changes.

Generation of New Random Numbers

Our proposal includes several options on when to (re)generate R, resulting in a changed IPv6 address. The different options provide different levels of privacy protection, which we now discuss.

1. **Generate *R* on OS install**. If R is generated during system install and then never changed, I will change when the network prefix advertised by the router changes. As long as p remains constant, so will I. This option is useful for laptops in enterprise environments. As long as the laptop is on the corporate network, the IPv6 address will be fixed. When the laptop is removed from the network (e.g., the employee goes to a USENIX workshop), the interface identifier I will change, preventing the employee from being tracked as they roam between networks. When they rejoin the enterprise environment, they will re-obtain the original interface identifier.
2. **Generate *R* on OS reboot**. This option results in a new IPv6 address every time the computer is rebooted, even if the client receives the same IPv6 prefix from the broadcast daemon.
3. **Generate *R* on network interface change**. This option results in a new interface identifier I being generated whenever the client computer brings up the network interface. Since interfaces are brought up on boot and when connecting to a wireless network, a client will use a different I each time it joins a network broadcasting the same IPv6 prefix.
4. **Generate *R* when the user chooses**. This option results in a new interface identifier I being generated based on user involvement (e.g., the user regenerates R when transitioning between tasks). While we include this option for completeness, we do not suggest defaulting to this option.
5. **Generate *R* every *x* seconds**. In this option, the client generates a new IPv6 address every x seconds. This approach closely parallels the current IPv6 privacy scheme. Unlike the approaches discussed above, a new IPv6 address may be generated while network connections are open, causing these connections to be dropped. To reduce the number of dropped connections, the kernel can avoid deleting old IPv6 temporary addresses associated with active network connec-

tions. As long as the active network prefix is the same as that contained in the old temporary address, the temporary address can continue to be used. While we note that *x* does not need to stay fixed (i.e., a new *x* can also be chosen when the random identifier is updated), we currently see no additional benefit in randomly changing *x*. A default *x* of one day mirrors the current default with IPv6 privacy extensions.

We suggest the generation of a new random number (and hence interface identifier) whenever the network interface is brought up (option 3). This method generates IPv6 addresses as frequently as possible without interrupting open connections (since connections are terminated when the interface goes down).

Our proposal is designed so that given two distinct IPv6 addresses, it should be hard for an attacker to answer the question, "Did the same client use both IPv6 addresses?" To answer this question, the attacker must be able to determine that the same *R* value was used in the generation of both addresses (since two clients sharing the same *R* value is extremely unlikely).

In answering this question, we assume that the attacker has access to the generated interface identifier as well as to the prefix. The security, therefore, depends on the difficulty of determining the random number provided to the pre-image resistant hash function—which is assumed to be hard for a sufficiently large *R*. An attacker attempting to track a client would need to keep trying random values for *R* until finding one which generates multiple distinct and observed interface identifiers; therefore a birthday attack [2, 8] does not seem to help.

Because the interface identifier changes whenever the prefix changes, a client connecting through two networks with different prefixes will also connect with different interface identifiers, leaking no information in the IPv6 address.

One potential attack against our proposal involves a network administrator (as attacker) broadcasting target prefixes in an effort to detect what interface identifier would be used by the client on that network (e.g., the administrator broadcasts prefix 1:2:3::/64 to determine what IPv6 address the client would attempt to use on that network). The attack would be successful if *R* was not updated by the client before visiting the target network. One way to defend against this attack is to configure the client to generate a new *R* whenever it enables or makes a change to the network interface.

The proposal for generating interface identifiers relies on an appropriate random number generator. If *R* can be guessed, an attacker can determine whether a client using the same *R* value generated multiple distinct IPv6 addresses using distinct prefixes. The proposal also depends on a pre-image resistant hash function (SHA-2 should suffice).

Our approach does not protect against an attacker identifying two addresses used by a client through correlating the time at which one IPv6 address stops being used and another starts. We expect that as IPv6 privacy extensions are deployed, the volume of IPv6 address churn will make correlations more difficult.

As an extension to the core approach, it may be possible to use multiple IPv6 temporary addresses concurrently on a host. As an example, a new IPv6 address could be used for every application running on the client (e.g., Web browsing would use one IPv6 address, DNS queries would use another, and an active SSH connection may use a third). While not directly privacy related, a server may also choose to use multiple IPv6 addresses (e.g., as a method for distributing firewalls [12]).

Implementation

For our prototype implementation, we used version 2.6.34 of the Linux kernel. We modified the currently implemented IPv6 privacy extensions. The modified kernel provides several sys-controls which can be read and written to by user-space programs, controlling the operation of IPv6 privacy extensions. These sys-controls are as follows:

use_tempaddr: controls whether or not to enable privacy extensions. Possible values are listed in Table 1.

temp_valid_lft: the maximum amount of time a temporary address is valid. In the original approach, a new distinct temporary address would be created. In this proposal, the lifetime of an already existing temporary address will be extended when router advertisements are received if both p and R are unchanged.

temp_prefered_lft: the maximum amount of time a temporary address will be the preferred address for the interface. As with temp_valid_lft, the lifetime will be extended if both p and R are unchanged when a router advertisement is received.

temp_random (new): a 32 byte (256 bit) random value R used as input to the hash function. This sys-control is specific to our proposal.

We tested our prototype by switching between several IPv6 networks and verifying that the generated IPv6 addresses were different at each network. We did not notice any impact on activities such as Web browsing and SSH. During the course of implementing the proposed IPv6 privacy extensions in Linux, we found several bugs which cause IPv6 privacy extensions to be disabled and/or have all temporary addresses deleted. We have a patch in version 2.6.37 of the Linux kernel which addresses these implementation deficiencies. We will be submitting our proposed revisions to privacy extensions to the Linux kernel in hopes that this will help improve adoption.

Value	*Meaning*
0	Privacy extension disabled
1	Original privacy extension enabled but not used by default for new outgoing connections
2	Original privacy extension enabled and used by default for new outgoing connections
5	Proposed privacy extension enabled but not used by default for new outgoing connections
6	Proposed privacy extension enabled and used by default for new outgoing connections

Table 1: Possible values and their meanings for the *use_tempaddr* sys-control in the modified kernel.

Other Related Work

Cryptographically Generated Addresses [1] (CGA) were proposed to prevent stealing and/or spoofing IPv6 addresses. CGAs define a method for securely associating a public key to an IPv6 address. The interface identifier of the IPv6 address is a cryptographic hash of the public key, which can later be verified by the recipient of the packet or message. Because CGAs tie a public key to an IPv6 address, even as hosts switch networks, they are uniquely identifiable through use of the public key. CGAs, like our proposal, use a cryptographic hash to generate the interface identifier, but the purpose of CGAs is contrary to ours and our proposal does not involve public keys.

Mobility extensions (RFC 3775) define ways in which a mobile host can move between networks while maintaining open connections, even if the networks use different link layer technologies (WiMAX, LTE, WiFi). This is accomplished by establishing a tunnel (usually with IPSec) to the *home network*. The mobile host is then reachable through the proxy home network. Route optimization (RFC 4866) allows the correspondent node (server) to communicate directly with the mobile host, even though the mobile host's IPv6 address may continue to change. IPv6 Mobile extensions with route optimization are illustrated in Figure 3.

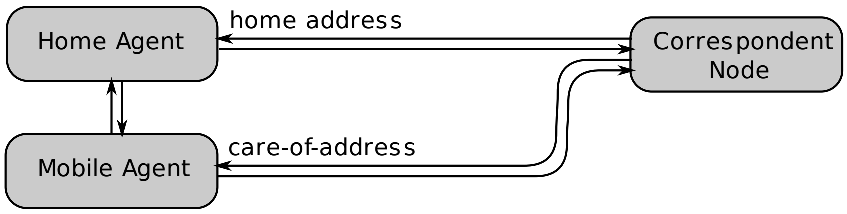


Figure 3: Communication with Mobile IPv6 agents

For mobile agents, implementing route optimization is mandatory. The correspondent node receives both the home address and care-of address and can track the location of the mobile agent as it moves between IPv6 networks. Because the mobile agent is also tied to the fixed home address, enabling privacy extensions at the mobile agent does not prevent the correspondent node from tracking the mobile agent. To prevent tracking the mobile agent, the home address and care-of address must be changed at the same time. Otherwise, the correspondent node can tie the old address to the new address and continue to track the mobile node.

Mobility extensions aim to address connection persistence problems rather than privacy concerns. The former have also been solved independently in the cell phone industry at the link layer, where cell towers hand off connections to prevent active phone calls from being dropped when a device switches towers.

Tracking at Other Layers

Guha and Francis [6] demonstrate how to track users through DNS. Their analysis shows that dynamic DNS updates combined with geolocation [9] can provide a passive attacker with all the approximate locations visited by a victim. Geolocation in IPv6 may also reveal more accurate results compared to IPv4 due to address allocation recommendations (http://www.apnic.net/policy/ipv6-address-policy).

Users with private IPv6 addresses may be tracked at the application layer through cookies [7] or browser characteristics [5]. Protecting privacy at all layers is clearly a difficult problem and beyond our scope, but we argue that in order to have privacy at higher-level protocols, underlying protocols must also be private.

While we do not address the problem of tracking users on the LAN specifically in this article, we note that tracking users at the Ethernet level is possible due to clients broadcasting their MAC addresses [10]. Because MAC addresses in the Ethernet header are overwritten on a hop-by-hop basis, attackers outside the LAN do not obtain the MAC address of a client. This article focuses on the network layer, where tracking can be performed across the Internet.

Conclusion

We have proposed a new way of generating the interface identifier used in temporary IPv6 addresses. The use of temporary addresses prevents tracking clients as they move between IPv6 networks. Our approach does not use MAC addresses, which can be used to identify client hardware.

Our proposal has several benefits over the current IPv6 privacy extension scheme, including: (1) the ability to maintain a consistent IPv6 address over an extended period regardless of the lifetime specified by a router advertisement (as long as the prefix being advertised does not change); (2) the ability to always use the same interface identifier while connected to a network broadcasting an unchanging prefix; (3) the ability to configure when a new interface identifier should be created (e.g., whenever the network interface is brought up); and (4) not being able to track a client through the use of a common interface identifier across networks broadcasting different IPv6 prefixes. We have implemented and tested the approach in Linux and found that it generates new interface identifiers as designed while not impacting Internet activities.

A version of this article is also available as Technical Report TR-10-17 (September 9, 2010), Carleton University, School of Computer Science.

References

[1] T. Aura, “Cryptographically Generated Addresses (CGA),” in *Proceedings of the 6th International Information Security Conference (ISC ’03)*, pp. 29–43.

[2] M. Bellare, O. Goldreich, and H. Krawczyk, “Stateless Evaluation of Pseudorandom Functions: Security beyond the Birthday Barrier,” 19th International Conference on Cryptology (Crypto ’99).

[3] S. Bellovin, B. Cheswick, and A. Keromytis, “Worm Propagation Strategies in an IPv6 Internet,” *;login:*, vol. 31, no. 1 (February 2006), pp. 70–76.

[4] P. Biondi, A. Ebalard, M. Balmer, and V. Manral, “Ipv6 Protocol Type 0 Route Header Denial of Service Vulnerability,” April 23, 2007: http://www.securityfocus.com/bid/23615.

[5] P. Eckersley, “A Primer on Information Theory and Privacy”: https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy.

[6] S. Guha and P. Francis, “Identity Trail: Covert Surveillance Using DNS,” in *Proceedings of the Privacy Enhancing Technologies Symposium*, 2007.

[7] D. Kristol and L. Montulli, “HTTP State Management Mechanism,” RFC 2965 (Proposed Standard), October 2000.

[8] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography* (CRC Press, 1996).

[9] J.A. Muir and P.C. Van Oorschot, “Internet Geolocation: Evasion and Counter-evasion,” *ACM Computing Surveys* (CSUR), vol. 42, no. 1 (2009), pp. 1–23.

[10] L. Peterson and B. Davie, *Computer Networks: A Systems Approach* (Morgan Kaufmann, 2007).

[11] B. Preneel and P.C. van Oorschot, “On the Security of Iterated Message Authentication Codes,” *IEEE-IT*, vol. 45, no. 1 (January 1999), pp. 188–99.

[12] H. Zhao, C.-K. Chau, and S.M. Bellovin, “ROFL: Routing as the Firewall Layer,” in New Security Paradigms Workshop (NSPW), 2008.

Automating Configuration Troubleshooting with ConfAid

MONA ATTARIYAN AND JASON FLINN

Mona Attariyan is a PhD candidate at the Computer Science and Engineering Department of the University of Michigan. Her research interests broadly include software systems, especially operating systems and distributed systems.
monattar@umich.edu

Jason Flinn is an associate professor of computer science and engineering at the University of Michigan. His research interests include operating systems, distributed systems, storage, and mobile computing.
jflinn@umich.edu

Complex software systems are difficult to configure and manage. When problems inevitably arise, operators spend considerable time troubleshooting those problems by identifying root causes and correcting them. The cost of troubleshooting is substantial. Technical support contributes 17% of the total cost of ownership of today's desktop computers [3], and troubleshooting misconfigurations is a large part of technical support. Even for casual computer users, troubleshooting is often enormously frustrating.

Our research group is exploring how operating system support for dynamic information flow analysis can substantially simplify and reduce the human effort needed to troubleshoot software systems. We are focusing specifically on configuration errors, in which the application code is correct, but the software has been installed, configured, or updated incorrectly so that it does not behave as desired. For instance, a mistake in a configuration file may lead software to crash, assert, or simply produce erroneous output.

Consider how users and administrators typically debug configuration problems. Misconfigurations are often exhibited by an application unexpectedly terminating or producing undesired output. While an ideal application would always output a helpful error message when such events occur, it is unfortunately the case that such messages are often cryptic, misleading, or even non-existent. Thus, the person using the application must ask colleagues and search manuals, FAQs, and online forums to find potential solutions to the problem.

ConfAid helps mitigate such problems. ConfAid is run offline, once erroneous behavior has been observed. A ConfAid user reproduces the problem by executing the application while ConfAid monitors the application's behavior. The user specifies the application she wishes to troubleshoot and its sources of configuration data (e.g., httpd.conf for the Apache Web server). ConfAid automatically diagnoses the root causes of self-evident errors, such as assertion failures and exits with non-zero return codes. ConfAid also allows its user to specify undesired output (e.g., specific error strings); it monitors application output to files, network, and other external devices for such user-specified error conditions.

When ConfAid observes erroneous application behavior, it outputs an ordered list of probable root causes. Each entry in the list is a token from a configuration source; our results show that ConfAid typically outputs the actual root cause as the first or second entry in the list. This allows the ConfAid user to focus on one or two specific configuration tokens when deciding how to fix the problem, which can dramatically improve the total time to recovery for the system.

The rest of this article briefly describes ConfAid's design and implementation, and it gives a short summary of our experiments with ConfAid. More details can be found in our OSDI '10 paper [1].

Design Principles of ConfAid

We begin by describing ConfAid's design principles.

Use White-Box Analysis

The genesis of ConfAid arose from AutoBash [8], our prior work in configuration troubleshooting. AutoBash tracks causality at process and file granularity in order to diagnose configuration errors. It treats each process as a black box, such that all outputs of the process are considered to be dependent on all prior inputs. We found AutoBash to be very successful in identifying the root cause of problems, but the success was limited in that AutoBash would often identify a complex configuration file, such as Apache's httpd.conf, as the source of an error. When such files contain hundreds of options, the root cause identification of the entire file is often too nebulous to be of great use.

Our take-away lessons from AutoBash were: (1) causality tracking is an effective tool for identifying root causes, and (2) causality should be tracked at a finer granularity than an entire process to troubleshoot applications with complex configuration files. These observations led us to use a white box approach in ConfAid that tracks causality within each process at byte granularity.

Operate on Application Binaries

We next considered whether ConfAid should require application source code for operation. While using source code would make analysis easier, source code is unavailable for many important applications, which would limit the applicability of our tool. Also, we felt it likely that we would have to choose a subset of programming languages to support, which would also limit the number of applications we could analyze. For these reasons, we decided to design ConfAid to not require source code; ConfAid instead operates on program binaries.

Embrace Imprecise Analysis

Our final design decision was to embrace an imprecise analysis of causality that relies on heuristics rather than using a sound or complete analysis of information flow. Using an early prototype of ConfAid, we found that for any reasonably complex configuration problem, a strict definition of causal dependencies led to our tool outputting almost all configuration values as the root cause of the problem. Thus, our current version of ConfAid uses several heuristics to limit the spread of causal dependencies. For instance, ConfAid does not consider all dependencies to be equal. It considers data flow dependencies to be more likely to lead to the root cause than control flow dependencies. It also considers control flow dependencies introduced closer to the error exhibition to be more likely to lead to the root cause than more distant ones. In some cases, ConfAid's heuristics can lead to false negatives and false positives. However, our results show that in most cases, they are quite effective in narrowing the search for the root cause and reducing execution time.

ConfAid's Information Flow Analysis

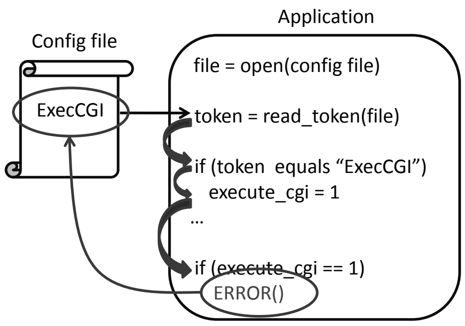


Figure 1: ConfAid propagates configuration tokens throughout the application using information flow analysis. When an error happens, ConfAid uses the propagated information to determine the root cause of the undesired outcome.

We use the example in Figure 1 to illustrate the mechanics of ConfAid. Assume that the application exhibits an error if the configuration token ExecCGI exists in the config file. When the application runs, ConfAid uses taint tracking [7] to dynamically monitor the propagation of configuration tokens and to determine how the erroneous outcome depends on the configuration data. When the application reads the value of the ExecCGI token from the configuration file, ConfAid taints the memory location that stores that value of token to indicate that its value could change if the user were to modify the value of ExecCGI in the configuration file. As the application executes, ConfAid observes that the value of execute_cgi depends on the value of the token, so it also taints that memory location. When the error happens, ConfAid sees that the error could have been avoided if the branch that tests execute_cgi had a different outcome. Since execute_cgi is tainted by the ExecCGI option, ConfAid identifies that configuration option as the root cause of the error.

To analyze the information flow, ConfAid adds custom logic, referred to as instrumentation, to each application binary using Pin [6]. The instrumentation monitors each system call, such as read or pread, that could potentially read data from a configuration source. If the source of the data returned by a system call is a configuration file, ConfAid annotates the registers and memory addresses modified by the system call with a marker that indicates a dependency on a specific configuration token. Borrowing terminology from the taint tracking literature, we refer to this marking as the taint of the memory location. If an address or register is tainted by a token, ConfAid believes that the value at that location might be different if the value of the token in the original configuration source were to change.

```
/* a, b, c and d are read from the config file*/
if (c == 0) { /* c set to 0 in config file */
       x = a;  /* taken path */
} else {
       y = b;  /* alternate path */
}
z = d;
if (z) assert();  /* The erroneous behavior */
```

Figure 2: Example to illustrate causality tracking. The assertion only depends on variable z, which itself depends on the value of configuration token d. Configuration token c only affects variables x and y.

ConfAid specifies the taint of each variable as a set of configuration options. For instance, if the taint set of a variable is { FOO, BAR }, ConfAid believes that the value of that variable could change if the user were to modify either the FOO or the BAR token in the configuration file.

Taint is propagated via data flow and control flow dependencies. When a monitored process executes an instruction that modifies a memory address, register, or CPU flag, the taint set of each modified location is set to the union of the taint sets of the values read by the instruction. For example, consider the instruction x = y + z where the taint set of x becomes the union of taint sets of y and z. Intuitively, the value of x might change if a configuration token were to cause y or z to change prior to the execution of this instruction.

In traditional taint tracking for security purposes, control flow dependencies are often ignored to improve performance because they are harder than data flow dependencies for an attacker to exploit. With ConfAid, however, we have found that tracking control flow dependencies is essential since they propagate the majority of configuration-derived taint. A naive approach to tracking control flow is to union the taint set of a branch conditional with a running control flow dependency for the program. However, without mechanisms to remove control flow taint, the taint grows without limit. This causes too many false positives in ConfAid's root cause list.

A more precise approach takes into account the basic block structure of a program. Consider the example in Figure 2. Assume a, b, c, and d were read from a configuration file and have taint sets assigned to them. The value of c does not affect whether the last two statements are executed, since they execute in all possible paths (and therefore for all values of c). Thus, the taint of c should be removed from the control flow taint before executing z = d. When the program asserts, the control flow taint should only include the taint of d to correctly indicate that changing the value of d might fix the problem.

ConfAid also tracks implicit control flow dependencies. In Figure 2, the values of x and y depend on c when the program asserts, since the occurrence of their assignments to a and b depend on whether or not the branch is taken. Note that y is still dependent on c even though the else path is not taken by the execution, since the value of y might change if a configuration token is modified such that the condition evaluates differently.

When the program executes a branch with a tainted condition, ConfAid first determines the merge point (the point where the branch paths converge) by consulting the control flow graph. Prior to dynamic analysis, ConfAid obtains the graph by using IDA Pro [2] to statically analyze the executable and any libraries it uses (e.g., libc and libssl). For each tainted branch, ConfAid next explores each alternate path that leads to the merge point. We define an alternate path to be any path not taken by the actual program execution that starts at a conditional branch instruction for which the branch condition is tainted by one or more configuration values. ConfAid uses alternate path exploration to learn which variables would have been assigned had the condition evaluated differently due to a modified configuration value.

To evaluate an alternate path, ConfAid switches the condition outcome and forces the program to execute the alternate path. ConfAid uses copy-on-write logging to checkpoint and roll back application state. When a memory address is first altered along an alternate path, ConfAid saves the previous value in an undo log. At the end of the execution, application state is replaced with the previous values from the log. Many branches need not be explored since their conditions are not tainted by any configuration token. After exploring the alternate paths, ConfAid performs a similar analysis for the path actually taken by the program. This is the actual execution, so no undo log is needed.

ConfAid also uses alternate path exploration to learn which paths avoid erroneous application behavior. An alternate path is considered to avoid the erroneous behavior if the path leads to a successful termination of the program or if the merge point of the branch occurs after the occurrence of the erroneous behavior in the program (as determined by the static control flow graph). ConfAid unions the taint sets of all conditions that led to such alternate paths to derive its final result. This result is the set of all configuration tokens which, if altered, could cause the program to avoid the erroneous behavior.

Figure 3 shows four examples that illustrate how ConfAid detects alternate paths that avoid the erroneous behavior. In case (a), the error occurs after the merge point of the conditional branch. ConfAid determines that the branch does not contribute to the error, because both paths lead to the same erroneous behavior. In case (b), the alternate path avoids the erroneous behavior because the merge point occurs after the error, and the alternate path itself does not exhibit any other error. In this case, ConfAid considers tokens in the taint set of the branch condition as possible root causes of the error, since if the application had taken the alternate path, it could have avoided the error. In case (c), the alternate path leads to a different error (an assertion). Therefore, ConfAid does not consider the taint of the branch as a possible root cause, because the alternate path would not lead to a successful termination. In case (d), there are two alternate paths, one of which leads to an assertion and one that reaches the merge point. In this case, since there exists an alternate path that avoids the erroneous behavior, configuration tokens in the taint set of the branch condition are possible root causes.

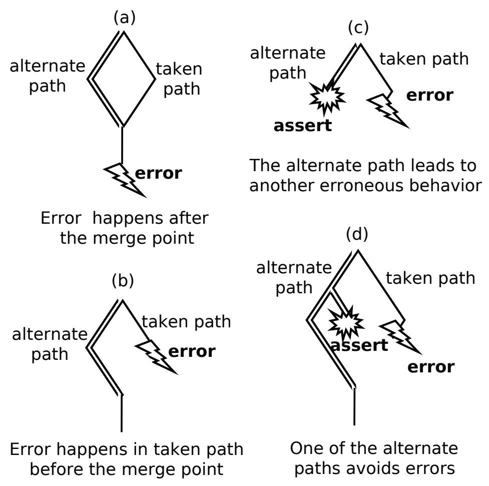

(a)
alternate path
taken path
error
Error happens after the merge point
(b)
alternate path
taken path
error
Error happens in taken path before the merge point
(c)
alternate path
taken path
assert
error
The alternate path leads to another erroneous behavior
(d)
alternate path
taken path
assert
error
One of the alternate paths avoids errors

Figure 3: Examples illustrating ConfAid path analysis

Heuristics for Performance

ConfAid uses two heuristics to simplify control flow analysis. The first heuristic is the bounded horizon heuristic. ConfAid only executes each alternate path for a fixed number of instructions. By default, ConfAid uses a limit of 80 instructions. All addresses and registers modified within the limit are used to calculate information flow dependencies after the merge point. Locations modified after the limit do not affect dependencies introduced at the merge point.

The second heuristic simplifies control flow analysis by assuming that the configuration file contains only a single error—we refer to this as the single mistake heuristic. This heuristic reduces the amount of taint in the application and the number of alternative paths that need to be explored by restricting the number of configuration values that can change. The single mistake heuristic may lead to false negatives. Potentially, if ConfAid cannot find a root cause, we can relax the single-mistake assumption by allowing ConfAid to assume that two or more tokens are erroneous. In our experiments to date, this heuristic has yet to trigger a false negative.

Heuristics for Reducing False Positives

In our design as described so far, two configuration tokens are considered equal taint sources even if one has a direct causal relationship to a location (e.g., the value in memory was read directly from the configuration file) and another has a nebulous relationship (e.g., the taint was propagated along a long chain of conditional assignments deep along alternate paths). Another problem we noticed was that loops could cause a location to become a global source and sink for taint. For instance, Apache reads its configuration values into a linked list structure and then traverses the list in a loop to find the value of a particular configuration token. During the traversal, the program control flow picks up taint from many configuration options, and these taints are sometimes transferred to the configuration variable that is the target of the search.

We realized that both of these problems were caused by our implicit assumption that all information flow relationships should be treated equally. Based on this observation, we decided to modify our design to instead track taint as a floating-point weight ranging in value between zero and one. When the error happens, ConfAid can rank the possible root causes based on their weights with the option with the highest weight being ranked first.

Our weights are based on two heuristics. First, data flow dependencies are assumed to be more likely to lead to the root cause than control flow dependencies. Second, control flow dependencies are assumed to be more likely to lead to the root cause if they occur later in the execution (i.e., closer to the erroneous behavior). Specifically, we assign taints introduced by control flow dependencies only half the weight of taints introduced by data flow dependencies. Further, each nested conditional branch reduces the weight of dependencies introduced by prior branches in the nest by one half. We chose a weight of 0.5 for speed: it can be implemented efficiently with a vector bit shift.

Multi-Process Causality Tracking

The most difficult configuration errors to troubleshoot involve multiple interacting processes. Such processes may be on a single computer, or on multiple computers connected by a network. To troubleshoot such cases, ConfAid instruments multiple processes at the same time and propagates taint information at per-byte granularity along with the data sent when the processes communicate. ConfAid supports processes that communicate using UNIX sockets, pipes, and TCP and UDP sockets and files. Since these operations are performed by Pin instrumentation, the taint propagation is hidden from the application and no operating system modifications are needed.

Evaluation

Our evaluation answers two questions:

> How effective is ConfAid in identifying the root cause of configuration problems?
>
> How long does ConfAid take to find the root cause?

Experimental Setup

We evaluated ConfAid on three applications: the OpenSSH server version 5.1, the Apache HTTP server version 2.2.14, and the Postfix mail transfer agent version 2.7. All of our experiments were run on a Dell OptiPlex 980 desktop computer with an Intel Core i5 Dual Core processor and 4GB of memory. The machine runs Linux kernel version 2.6.21. For Apache, ConfAid instruments a single process; for OpenSSH, up to two processes; and for Postfix, up to six processes.

To evaluate ConfAid, we manually injected errors into correct configuration files. Then we ran a test case that caused the error we injected to be exhibited. We used ConfAid to instrument the process (or processes) for that application and obtained the ordered list of root causes found by ConfAid. We use two metrics to evaluate ConfAid's effectiveness: the ranking of the actual root cause, i.e., the injected mistake, in the list returned by ConfAid and the time to execute the instrumented application.

We used two different methods to generate configuration errors. First, we injected 18 real-world configuration errors that were reported in online forums, FAQ pages, and application documentation. Second, we used the ConfErr tool [4] to inject random errors into the configuration files of the three applications. ConfErr uses human error models rooted in psychology and linguistics to generate realistic configuration mistakes. We used ConfErr to produce 20 errors for each application.

Results

Application	*Root causes ranked first*	*Root causes ranked first with one tie*	*Root causes ranked second*	*Root causes ranked second with one tie*	*Avg. time to run*
OpenSSH (7 bugs)	2	2	2	1	52s
Apache (6 bugs)	3	1	0	2	2m 48s
Postfix (5 bugs)	5	0	0	0	57s

Table 1: Results for real-world configuration bugs

Table 1 summarizes our results for real-world misconfigurations. ConfAid ranks the actual root cause first in 13 cases and second in the other 5. Sometimes, when the actual root cause is ranked second, the token ranked first provides a valuable clue to help troubleshoot the problem. For instance, in Apache the actual error usually occurs nested inside a section or directive command in the config file. For the two Apache errors where the root cause is ranked second, the top-ranked option is the section or directive containing the error.

ConfAid's average execution time of 1:32 minutes is much faster and far less frustrating than manual troubleshooting. For instance, one of the Apache misconfigurations is taken from a thread in linuxforums.org [5]. After trying to fix the misconfiguration for quite a while, the user went to the trouble of posting the question in the forum and waited two days for an answer. ConfAid identified the root cause in less than three minutes.

Application	*Root causes ranked first*	*Root causes ranked first with one tie*	*Root causes ranked second*	*Root causes ranked second with one tie*	*Avg. time to run*	*Avg. time to run*
OpenSSH	17 (85%)	1 (5%)	1 (5%)	0	1 (5%)	7s
Apache	17 (85%)	1 (5%)	0	1 (5%)	1 (5%)	24s
Postfix	15 (75%)	0	2 (10%)	0	3 (15%)	38s

Table 2: Results for randomly generated bugs

Table 2 summarizes the results for randomly generated configuration errors. For OpenSSH, ConfAid ranked the root cause first or second for 95% of the bugs. For the last bug, ConfAid could not run to completion due to unsupported system calls used in the code path. We could remedy this by supporting more calls. ConfAid also successfully diagnosed 95% of the Apache errors. For the remaining bug, the correct root cause was ranked 9th due to our weighting heuristic. For Postfix, ConfAid diagnosed 85% of the errors effectively. The remaining three errors were due to missing configuration options. Currently, ConfAid only considers all tokens present in the configuration file as possible sources of the root cause. If a default value can be overridden by a token not actually in the file, then ConfAid will not detect the missing token as a possible root cause. We plan to extend ConfAid to also diagnose misconfigurations that are due to missing configuration tokens.

Conclusions and Future Work

Configuration errors are costly, time-consuming, and frustrating to troubleshoot. ConfAid makes troubleshooting easier by pinpointing the specific token in a configuration file that led to an erroneous behavior. Compared to prior approaches, ConfAid distinguishes itself by analyzing causality within processes as they execute without the need for application source code. It propagates causal dependencies among multiple processes and outputs a ranked list of probable root causes. Our results show that ConfAid usually lists the actual root cause as the first or second entry in this list. Thus, ConfAid can substantially reduce total time to recovery and perhaps make configuration problems a little less frustrating.

There are several possible directions for future work. First, ConfAid currently only troubleshoots configuration problems that lead to crashes, assertion failures, and incorrect output; it does not yet help diagnose misconfigurations that cause poor performance. One approach to tackling performance problems that we are investigating is to first use statistical sampling to associate use of a bottleneck resource such as disk or CPU with specific points in the program execution. Then,ConfAid-style analysis can determine which configuration tokens most directly affect the frequency of execution of those points.

Second, ConfAid currently assumes that the configuration file contains only one erroneous token. If fixing a particular error requires changing two tokens, then ConfAid's alternate path analysis may not identify both tokens. We therefore plan to allow ConfAid to track sets of two or more misconfigured tokens and measure the resulting performance overhead. Potentially, we can use an expanding search technique in which ConfAid initially performs an analysis assuming only a single mistake, and then performs a lengthier analysis allowing multiple mistakes if the first analysis does not yield satisfactory results.

Finally, we believe that ConfAid can be best improved if it is used and tested by many people. Therefore, we plan to release an open source version of ConfAid to the public. This will require us to make ConfAid more robust in diverse computing environments, and we will also need to use an open source static analysis tool to generate a control flow graph.

Acknowledgments

This research was supported by NSF award CNS-1017148. The views and conclusions contained in this document are those of the authors and should not

be interpreted as representing the official policies, either expressed or implied, of NSF, the University of Michigan, or the U.S. government.

References

[1] M. Attariyan and J. Flinn, "Automating Configuration Troubleshooting with Dynamic Information Flow Analysis," *Proceedings of the 9th Symposium on Operating Systems Design and Implementation*, October 2010.

[2] IDA Pro disassembler: http://www.hex-rays.com/idapro.

[3] A. Kapoor, "Web-to-Host: Reducing Total Cost of Ownership," Technical Report 200503, The Tolly Group, May 2000.

[4] L. Keller, P. Upadhyaya, and G. Candea, "ConfErr: A Tool for Assessing Resilience to Human Configuration Errors," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2008, pp. 157–166.

[5] http://www.linuxforums.org/forum/servers/125833-solved-apache-wont-follow-symlinks.html.

[6] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2005, pp. 190–200.

[7] J. Newsome and D. Song, "Dynamic Taint Analysis: Automatic Detection, Analysis, and Signature Generation of Exploit Attacks on Commodity Software," *Proceedings of the 12th Network and Distributed Systems Security Symposium*, February 2005.

[8] Y.-Y. Su, M. Attariyan, and J. Flinn, "AutoBash: Improving Configuration Management with Operating System Causality Analysis," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, October 2007, pp. 237–250.

Making System Administration Easier by Letting the Machines Do the Hard Work, Or, Becoming an Agile Sysadmin

JOSHUA FISKE

Joshua Fiske is the Manager of User Services at Clarkson University. In this role, he tends the University's fleet of Linux and VMware servers, handles day-to-day operations of the network, and manages a team of professionals who provide direct end-user support. He is passionate about the use of open-source software in higher education.
jfiske@clarkson.edu

One of the challenges faced in the information technology field is the need to be responsive to the needs of our customers. To a system administrator, this typically manifests itself in one of two ways: responding to reports of broken stuff and responding to requests for new stuff. In this article we will focus mainly on doing the latter by taking advantage of virtualization, automation, and patch management.

In the days of yore, responding to a request for new stuff was a costly and time-consuming process that went something like this: Quote and order new hardware. Wait 30 days for hardware to arrive. Rack-mount hardware and install operating system. Configure operating system and services the same way "all the others" are. And don't make a mistake lest we end up with a configuration that is different from everything else in our environment. Heaven help you if you need to make a change to a configuration or if you have a team of sysadmins who each have their "own way" to do a task. Two months later, you might have the new service up and running.

And where are users while you're doing all this? They're working on their own to find a way to meet their need *right now*. By the time you've worked your way through this process, they've found another tool or their needs have changed. All because you weren't able to be agile in responding to their request. But what's the alternative? Imagine being able to receive a request from a user and have a system standing and ready to service their needs in the same day (or 15 minutes later)! Here's how I've been able to do just that.

The Solution

To solve this problem, we take a three-pronged approach: virtualization, automation, and patch management.

By now, most sysadmins are familiar with the advantages of virtualization. Combining systems to run on common hardware ensures efficient use of resources, improves disaster recovery posture, etc. And in this case, it also allows us to be more agile in responding to requests for new systems or services. Because we have a virtualized infrastructure, we generally have some "extra" capacity to stand up new systems without needing to order hardware.

Once we have a virtualized system, we must install an operating system. This process of installing the operating system and configuring services is the largest opportunity for human error to affect the quality of our output, because each sysadmin has a slightly different way of building servers and configuring services. We can fix this by leveraging tools like kickstart and Puppet to automate the process of building systems and configuring services.

And then once you have deployed a large fleet of systems, it can be very helpful to know which systems are in need of patching. A tool like Spacewalk can provide a centralized reporting interface to track this information and to push out updates.

The Practical Solution

Knowing the theoretical solution is good, but I always find a practical example to be much more helpful. So, I'll walk you through the process of provisioning a new server.

We have a pretty standard virtualization environment. In our shop we use VMware ESX (and in some cases, ESXi) with a shared iSCSI storage environment. To provision a new server, we just log in to a VM host with excess capacity and create a new guest. We have a standardized disk size of 20GB for our Linux guests. If we need more storage, we can attach another virtual drive and, through the magic of LVM, we can make that space available to the OS. Once the system is defined, we attach an ISO image and set it as the primary boot device. But don't power on the system just yet.

This is where the exciting stuff begins. We start by creating a kickstart file to define the properties of the new system. Since we've done this before, we can just copy an existing kickstart file and make a few tweaks to it, setting things like the IP address and hostname. Then, we boot the system and respond with the following at the Linux boot prompt:

```
boot:  linux ks=http://shep.clarkson.edu/provisioning/newhost.ks
ip=128.153.5.60 netmask=255.255.255.0 gateway=128.153.5.1 dns=128.153.5.254,
128.153.0.254
```

A full copy of our standard kickstart file is available linked from www.usenix.org/login/2011-02. The most interesting bits of that file are in our postscript, which is where we begin to automate the customizations that we generally perform at our site. Of course, you'll want to customize this to suit your environment. This is an opportunity to be a creative sysadmin; think about the tasks that you generally perform as part of your build process and think about scripting them here:

```
%post
#
# Disable some services
/sbin/chkconfig --level 123456 cups off
/sbin/chkconfig --level 123456 yum-updatesd off
/sbin/chkconfig --level 123456 apmd off
# Install Spacewalk
/bin/rpm -Uvh http://spacewalk.redhat.com/yum/1.0/RHEL/5/i386/spacewalk-
client-repo-1.0-2.el5.noarch.rpm
/usr/bin/yum -y install rhn-setup yum-rhn-plugin
/usr/sbin/rhnreg_ks --serverUrl=http://spacewalk.clarkson.edu/XMLRPC
--activationkey=<activation key here> --force
/bin/sed -i '/enabled=0/ d' /etc/yum.repos.d/CentOS-Base.repo
/bin/sed -i 's/\(gpgcheck=.*\)/\1\nenabled=0/g' /etc/yum.repos.d/CentOS-
Base.repo
```

```
/bin/rpm --import http://mirror.clarkson.edu/rpmforge/RPM-GPG-KEY.dag.txt
/bin/rpm --import http://mirror.clarkson.edu/epel/RPM-GPG-KEY-EPEL
#
# Install Puppet
/bin/rpm -Uvh http://shep.clarkson.edu/provisioning/software/puppet/
facter-1.3.7-1.el5.rf.i386.rpm
/bin/rpm -Uvh http://shep.clarkson.edu/provisioning/software/puppet/
puppet-0.22.4-1.el5.rf.i386.rpm
/bin/sed -i 's/#PUPPET_SERVER=.*/PUPPET_SERVER=shep.clarkson.edu/g' /etc/
sysconfig/puppet
/bin/sed -i 's/#PUPPET_LOG/PUPPET_LOG/g' /etc/sysconfig/puppet
/sbin/chkconfig --level 35 puppet on
```

Several things happen as part of our post-installation process. Reading through this code, we see that it configures some system services, joins the system to our Spacewalk server, and installs Puppet (and facter, which is a prerequisite for Puppet). Then the machine reboots and greets us with a login prompt.

As part of the Puppet setup, the client has generated a set of SSL keys and submitted them to the Puppetmaster server for signing. To allow the Puppet client to retrieve configurations and files from the Puppetmaster server, we need to sign that request. This is done on the Puppetmaster server with these commands:

```
/usr/sbin/puppetca --list
/usr/sbin/puppetca --sign <fqdn>
```

Now that our Puppet client is approved, it looks to the Puppetmaster to see what it should do. Our Puppetmaster directs the client to perform a number of actions, including creating users, sudoers configuration, installation of Apache, and some other settings.

Creating Users

We create a predefined set of administrative users on each host, ensure that these users are members of the wheel group for sudo, and set an RSA key for SSH:

```
class users {
      user { "jfiske" :
        ensure => present,
        uid => 500,
        gid => "wheel",
        managehome => true,
      }
      ssh_authorized_key { "jfiske-key" :
        require => User['jfiske'],
        ensure => present,
        key => "<rsa_key_here>",
        type => "ssh-rsa",
        user => "jfiske",
      }
}
```

Sudoers Configuration

We can also push an entire configuration file to each host. In this case, we have Puppet push the /etc/sudoers file to each system to ensure that the user we created above also has sudo abilities. This is particularly useful if you have a team of administrators who each need to have the same level of administrative access to each system (without needing to distribute a system's root password):

```
class sudo {
        file { "/etc/sudoers":
           owner  => "root",
           group  => "root",
           mode   => 440,
        source => "puppet://shep.clarkson.edu/configuration/sudoers",
        }
}
```

Installation of Apache

We can also install a software package and ensure that it is running. In this case, we install a Web server:

```
class apache {
        package { httpd: ensure => installed }

        service { "httpd":
           ensure => running,
           require  => Package["httpd"],
        }
}
```

Other Settings We Push

We use Puppet to install packages, distribute configuration files, and ensure that services are running. We can even combine these three functions with dependencies to perform more complex tasks. In this example, we tell Puppet which packages should be installed for SNMP, push the relevant configuration file (/etc/snmp/snmpd.conf) and a binary file (/usr/local/sbin/ntp_check), and then ensure that the SNMP service is running.

```
class snmpd {
        case $operatingsystem {
        CentOS: { $snmpd_packages = ["net-snmp", "net-snmp-libs"] }
        }
        package { $snmpd_packages: ensure => installed }
        file { "/etc/snmp/snmpd.conf":
           owner  => "root",
           group  => "root",
           mode   => 644,
        source => "puppet://shep.clarkson.edu/configuration/snmpd.conf",
        require => Package["net-snmp"],
        }
        file { "/usr/local/sbin/ntp_check":
           owner  => "root",
           group  => "root",
                mode   => 755,
                source => "puppet://shep.clarkson.edu/configuration/ntp_check",
                require => Package["net-snmp"],
            }
            service { snmpd:
            name => snmpd,
            enable => true,
            ensure => running
            }
    }
```

Once we have our Puppetmaster configured to perform all of our post-installation configuration steps, we can take the total system build time down to something around 10 minutes. If we are just adding a new application server to an existing pool, then we are done (because, of course, we've also Puppet'ed the install and configuration of the app server software). If we are deploying a new application, then we have a platform on which to begin developing/installing. Reducing system build time allows system administrators to begin to pull themselves up out of the daily minutiae of building systems and instead focus on the more important and interesting aspects of keeping their users happy.

Footnote: Improved Management Abilities

Also, consider the benefits that are yet to be reaped with respect to patch and configuration management. Because we've joined the system to our Spacewalk instance, we will receive notifications when there are package updates that need to be installed. And because we've pushed configurations using Puppet, when we need to make global changes we can update one file and have it propagate to all of our servers. We should also consider storing our Puppetmaster's configuration files in a revision control system, so that we have a solid history of what changes were made when and by whom.

Resources

Installing Spacewalk on CentOS: http://wiki.centos.org/HowTos/PackageManagement/Spacewalk.

James Turnbull, *Pulling Strings with Puppet*: http://apress.com/book/view/1590599780.

GNU Parallel: The Command-Line Power Tool

OLE TANGE

Ole Tange works in bioinformatics in Copenhagen. He is active in the free software community and is best known for his "patented Web shop" that shows the dangers of software patents (http://ole.tange.dk/swpat). He will be happy to go to your conference to give a talk about GNU Parallel.

ole@tange.dk

For daily sysadmin work you often need to do the same specific task to a lot of files/users/hosts/tables. Very often it doesn't matter which one is done first and it would be fine to do them in parallel—but you do not want to run them *all* in parallel, as that will slow your computer to a crawl. This is what GNU Parallel can help you do.

In this article you will see examples of uses of GNU Parallel. The examples are kept simple to make it easy to understand the idea, but there's nothing that prevents you from using GNU Parallel for more complex tasks. More complex examples can be found at http://www.gnu.org/software/parallel/man.html.

History of GNU Parallel

GNU Parallel started out as two separate programs: xxargs and parallel. The purpose of xxargs was to work like xargs but deal nicely with space and quotes. The purpose of parallel was simply to run jobs in parallel. Both programs originated in 2001. In 2005 the two programs were merged, since xxargs was very often used with parallel, and thus the name xxargs was abandoned.

GNU Parallel therefore has two main objectives: replace xargs and run commands in parallel.

In 2010 Parallel was adopted as an official GNU tool and the name was changed to GNU Parallel. For a short video showing simple usage of the tool go to http://nd.gd/0s.

Your First Parallel Job

GNU Parallel is available as a package for most UNIX distributions. See http://www.gnu.org/s/parallel if it is not obvious how to install it on your system. After installation find a bunch of files on your computer and gzip them in parallel:

```
parallel gzip ::: *
```

Here your shell expands * to the files, ::: tells GNU Parallel to read arguments from the command line, and gzip is the command to run. The jobs are then run in parallel. After you have gziped the files, you can recompress them with bzip2:

```
parallel "zcat {} | bzip2 >{.}.bz2" ::: *
```

Here {} is being replaced with the file name. The output from zcat is piped to bzip2, which then compresses the output. The {.} is the file name with its extension stripped (e.g., foo.gz becomes foo), so the output from file.gz is stored in file.bz2.

GNU Parallel tries to be very liberal in quoting, so the above could also be written:

```
parallel zcat {} "|" bzip2 ">"{.}.bz2 ::: *
```

Only the chars that have special meaning in shell need to be quoted.

Reading Input

As we have seen, input can be given on the command line. Input can also be piped into GNU Parallel:

```
find . -type f | parallel gzip
```

GNU Parallel uses newline as a record separator and deals correctly with file names containing a word space or a dot. If you have normal users on your system, you will have experienced file names like these. If your users are really mean and write file names containing newlines, you can use NULL as a record separator:

```
find . -type f -print0 | parallel -0 gzip
```

You can read from a file using -a:

```
parallel -a filelist gzip
```

If you use more than one -a, a line from each input file will be available as {#}:

```
parallel -a sourcelist -a destlist gzip {1} ">"{2}
```

The same goes if you read a specific number of arguments at a time using -N:

```
cat filelist | parallel -N 3 diff {1} {2} ">" {3}
```

If your input is in columns you can split the columns using --colsep:

```
cat filelist.tsv | parallel --colsep '\t' diff {1} {2} ">" {3}
```

--colsep is a regexp, so you can match more advanced column separators.

Building the Command to Run

Just like xargs, GNU Parallel can take multiple input lines and put those on the same line. Compare these:

```
ls *.gz | parallel mv {} archive
ls *.gz | parallel -X mv {} archive
```

The first will run mv for every .gz file, whereas the second will fit as many files into {} as possible before running.

The {} can be put anywhere in the command, but if it is part of a word, that word will be repeated when using -X:

```
(echo 1; echo 2) | parallel -X echo foo bar{}baz quux
```

will repeat bar-baz and print:

```
foo bar1baz bar2baz quux
```

If you do not give a command to run, GNU Parallel will assume the input lines are command lines and run those in parallel:

```
(echo ls; echo grep root /etc/passwd) | parallel
```

Controlling the Output

One of the problems with running jobs in parallel is making sure the output of the running commands do not get mixed up. traceroute is a good example of this as it prints out slowly and parallel traceroutes will get mixed up. Try:

```
traceroute foss.org.my & traceroute debian.org & traceroute freenetproject.
org & wait
```

and compare the output to:

```
parallel traceroute ::: foss.org.my debian.org freenetproject.org
```

As you can see, GNU Parallel only prints out when a job is done—thereby making sure the output is never mixed with other jobs. If you insist, GNU Parallel can give you the output immediately with -u, but output from different jobs may mix.

For some input, you want the output to come in the same order as the input. -k does that for you:

```
parallel -k echo {}';' sleep {} ::: 3 2 1 4
```

This will run the four commands in parallel, but postpone the output of the two middle commands until the first is finished.

Execution of the Jobs

GNU Parallel defaults to run one job per CPU core in parallel. You can change this with -j. You can put an integer as the number of jobs (e.g., -j 4 for four jobs in parallel) or you can put a percentage of the number of CPU cores (e.g., -j 200% to run two jobs per CPU core):

```
parallel -j200% gzip ::: *
```

If you pass -j a file name, the parameter will be read from that file:

```
parallel -j /tmp/number_of_jobs_to_run gzip ::: *
```

The file will be read again after each job finishes. This way you can change the number of jobs running during a parallel run. This is particularly useful if it is a very long run and you need to prioritize other tasks on the computer.

To list the currently running jobs you need to send GNU Parallel SIGUSR1:

```
killall -USR1 parallel
```

GNU Parallel will then print the currently running jobs on STDERR.

If you regret starting a lot of jobs, you can simply break GNU Parallel, but if you want to make sure you do not have half-completed jobs, you should send the signal SIGTERM to GNU Parallel:

```
killall -TERM parallel
```

This will tell GNU Parallel not to start any new jobs but to wait until the currently running jobs are finished before exiting.

When monitoring the progress on screen it is often nice to have the output prepended with the command that was run. -v will do that for you:

```
parallel -v gzip ::: *
```

If you want to monitor the progress as it goes you can also use --progress and --eta:

```
parallel --progress gzip ::: *
parallel --eta gzip ::: *
```

This is especially useful for debugging when jobs run on remote computers.

Remote Computers

GNU Parallel can use the CPU power of remote computers to help do computations. As an example, we will recompress .gz files into .bz2 files, but you can just as easily do other compute-intensive jobs such as video encoding or image transformation.

You will need to be able to log in to the remote host using ssh without entering a password (ssh-agent may be handy for that). To transfer files, rsync needs to be installed, and to help GNU Parallel figure out how many CPU cores each computer has, GNU Parallel should also be installed on the remote computers.

Try this simple example to see that your setup is working:

```
parallel --sshlogin yourserver.example.com hostname';' echo {} ::: 1 2 3
```

This should print out the hostname of your server three times, each followed by the numbers 1 2 3. --sshlogin can be shortened to -S. To use more than one server, do:

```
parallel -S yourserver.example.com,server2.example.net hostname';' echo {} :::
1 2 3
```

If you have a different login name, just prepend login@ to the server name—just as you would with ssh. You can also give more than one -S instead of using a comma:

```
parallel -S yourserver.example.com -S mylogin@server2.example.net hostname';'
echo {} ::: 1 2 3
```

The special sshlogin ':' is your local computer:

```
parallel -S yourserver.example.com -S mylogin@server2.example.net -S :
hostname';' echo {} ::: 1 2 3
```

In this case you may see that GNU Parallel runs all three jobs on your local computer ,because the jobs are so fast to run.

If you have a file containing a list of the sshlogins to use, you can tell GNU Parallel to use that file:

```
parallel --sshloginfile mylistofsshlogins hostname';' echo {} ::: 1 2 3
```

The special sshlogin .. will read the sshloginfile ~/.parallel/sshloginfile:

```
parallel -S .. hostname';' echo {} ::: 1 2 3
```

Transferring Files

If your servers are not sharing storage (using NFS or something similar), you often need to transfer the files to be processed to the remote computers and the results back to the local computer.

To transfer a file to a remote computer, you will use --transfer:

```
parallel -S .. --transfer gzip '< {} | wc -c' ::: *.txt
```

Here we transfer each of the .txt files to the remote servers, compress them, and count how many bytes they now take up.

After a transfer you often will want to remove the transferred file from the remote computers. --cleanup does that for you:

```
parallel -S .. --transfer --cleanup gzip '< {} | wc -c' ::: *.txt
```

When processing files the result is often a file that you want copied back, after which the transferred and the result file should be removed from the remote computers:

```
parallel -S .. --transfer --return {.}.bz2 --cleanup zcat '< {} | bzip2 >{.}.bz2'
::: *.gz
```

Here the .gz files will be transferred and then recompressed using zcat and bzip2. The resulting .bz2 file is transferred back, and the .gz and the .bz2 files are removed from the remote computers. The combination --transfer --cleanup --return foo is used so often that it has its own abbreviation: --trc foo.

You can specify multiple --trc if your command generates multiple result files.

GNU Parallel will try to detect the number of cores on remote computers and run one job per CPU core even if the computers have different number of CPU cores:

```
parallel -S .. --trc {.}.bz2 zcat '< {} | bzip2 >{.}.bz2' ::: *.gz
```

GNU Parallel as Part of a Script

The more you practice using GNU Parallel, the more places you will see it can be useful. Every time you write a for loop or a while-read loop, consider if this could be done in parallel. Often the for loop can completely be replaced with a single line using GNU Parallel; if the jobs you want to run in parallel are very complex, you may want to make a script and have GNU Parallel call that script. Occasionally your for loop is so complex that neither of these is an option.

This is where parallel --semaphore can help you out. sem is the short alias for parallel --semaphore.

```
for i in `ls *.log` ; do
      [... a lot of complex lines here ...]
      sem -j4 --id my_id do_work $i
done
sem --wait --id my_id
```

This will run do_work in the background until four jobs are running. Then sem will wait until one of the four jobs has finished before starting another job. The last line (sem --wait) pauses the script until all the jobs started by sem have finished. my_id is a unique string used by sem to identify this script, since you may have other sems running at the same time. If you only run sem from one script at a time, --id my_id can be left out.

A special case is sem -j1, which works like a mutex and is useful if you only want one program running at a time.

GNU Parallel as a Job Queue Manager

With a few lines of code, GNU Parallel can work as a job queue manager:

```
echo >jobqueue; tail -f jobqueue | parallel
```

To submit your jobs to the queue, do:

```
echo my_command my_arg >> jobqueue
```

You can, of course, use -S to distribute the jobs to remote computers:

```
echo >jobqueue; tail -f jobqueue | parallel -S ..
```

If you have a dir into which users drop files that need to be processed, you can do this on GNU/Linux:

```
inotifywait -q -m -r -e CLOSE_WRITE --format %w%f my_dir | parallel -u echo
```

This will run the command echo on each file put into my_dir or subdirs of my_dir. Here again you can use -S to distribute the jobs to remote computers:

```
inotifywait -q -m -r -e CLOSE_WRITE --format %w%f my_dir | parallel -S ..  -u echo
```

See You on the Mailing List

I hope you have thought of situations where GNU Parallel can be of benefit to you. If you like GNU Parallel please let others know about it through email lists, forums, blogs, and social networks. If GNU Parallel saves you money, please donate to the FSF https://my.fsf.org/donate. If you have questions about GNU Parallel, join the mailing list at http://lists.gnu.org/mailman/listinfo/parallel.

Practical Perl Tools: Hither and Yon

DAVID N. BLANK-EDELMAN

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

Today's column will be all about hither and yon. Actually, this column will be more about moving files from hither to yon or perhaps yon to hither (not entirely sure which; pity William Safire didn't write more Perl modules while he was alive). If you've ever had to move data around, this will be the column for you.

Built-In Functions and Core Modules for File Copying/Moving

Let's start close to home with the function and modules that ship with Perl. Perl has a rename() function should you want to change the name of a file, perhaps moving it in the process to a different place in the current file system. To copy data, there's always code like this:

```
open my $OLDFILE, '<', 'oldfilename' or
   die "Can't open oldfilename for reading:$!\n";
open my $NEWCOPY, '>', 'newfilename' or
   die "Can't open newfilename for writing:$!\n";
# binmode() makes sure no end-of-line translation is
# done on the OSes that make a distinction
# between binary and non-binary files
binmode $OLDFILE;
binmode $NEWFILE;
while (<$OLDFILE>) { print {$NEWCOPY} $_; }
close $NEWCOPY;
```

but that is so gauche. Better would be to use the File::Copy module, helpfully shipped with Perl since 5.002. Barring one twist, you'd be hard-pressed to find a more straightforward syntax:

```
use File::Copy;
copy('oldfilename','newfilename') or die "Copy failed: $!";
```

What's the twist? Sometimes you really want to write this instead:

```
use File::Copy 'cp';
cp('oldfilename','newfilename') or die "Copy failed: $!";
```

In the first sample, copy() creates a copy of the file and gives it the default permissions (e.g., from the active umask); in the second, cp() attempts to preserve the source file's permissions when creating the copy.

The File::Copy module also has a move() function that attempts to move the file in the most efficient way possible. If it can just rename() the file it will; otherwise it will attempt to copy the file and delete the source file.

One quick caveat about this and other similar modules: by default, it won't copy attributes that are specific to a particular file system/OS. For example, if you are trying to copy a file on NTFS, File::Copy won't copy the ACLs on the file by default. The module provides a syscopy() function that can address this concern under certain conditions when the right external modules are present. See the documentation for more information if you think you will need to care about this sort of thing.

Extending File Copy Functionality

Once we have basic file copies down, we can start extending this idea using some other non-core modules. Copying a single file is useful, but perhaps being able to copy an entire tree of files/directories is even better. File::Copy::Recursive is made to do just that. It offers a bit more control than you probably thought you needed, by providing separate functions for recursively copying either all of the files in a directory structure, all of the directories, or both. The syntax is equally easy to use:

```
use File::Copy::Recursive qw(fcopy dircopy rcopy fmove dirmove rmove);

fcopy($orig,$new) or die $!;   # copy files
dircopy($orig,$new) or die $!; # copy directories
rcopy($orig,$new) or die $!;   # copy either

fmove($orig,$new) or die $!;
dirmove($orig,$new) or die $!;
rmove($orig,$new) or die $!;
```

There are a number of option variables you can set to modify the behavior of these functions; be sure to check out the documentation.

Perhaps a more interesting module is File::Copy::Vigilant, which describes itself as "A module for when your files absolutely, positively have to get there." File::Copy::Vigilant code looks awfully similar:

```
use File::Copy::Vigilant;

copy_vigilant( $source, $destination );
move_vigilant( $source, $destination );
```

but what it does under the hood is even cooler. Before a copy or move takes place, the module computes the MD5 digest for the source file. It similarly computes it for the destination file after the operation takes place. If these two digests don't match up, File::Copy::Vigilant will retry the operation a configurable number of times. If you think an MD5 digest comparison is problematic for some reason (e.g., too computationally intensive), you can change it so that it tests file sizes or even does a byte-for-byte comparison of both files.

Bring on the Network Protocols: FTP

So far we've only discussed moving files around in cases where the source and the destination were both directly accessible as a file system from the local machine. But in this space age of plastics and Intergoogles, clearly that isn't always going to be the case. We need a way to automate the transfer of files between machines that are more loosely connected.

The first protocol for doing this that comes to mind is FTP. After all, it is the File Transfer Protocol. There are a number of modules to work with FTP, but we're not going to spend much time on the subject, largely because my sense is that FTP usage has waned tremendously over the years. I'm not going to get all hyperbolic and suggest it is a dying protocol, but the number of times I've fired up an FTP client in the last year can be counted on one hand, give or take a margin of error of a finger or two. Let's look at the basics and then I'll mention a few modules that could come in handy if for some reason you find yourself having to do more FTP transfers than you expect.

The seminal FTP-based module is Net::FTP. Net::FTP is shipped with Perl as part of the equally seminal libnet package. Here's the example from the documentation:

```
use Net::FTP;

$ftp = Net::FTP->new("some.host.name", Debug => 0)
  or die "Cannot connect to some.host.name: $@";

$ftp->login("anonymous",'-anonymous@')
  or die "Cannot login ", $ftp->message;

$ftp->cwd("/pub")
  or die "Cannot change working directory ", $ftp->message;

$ftp->get("that.file")
  or die "get failed ", $ftp->message;

$ftp->quit;
```

The code creates a new $ftp object using the FTP server name as an argument. With this object, we can log in to the server (in this case anonymously), change the working directory, get a file (and put one, though that's not demonstrated above) and then log out.

If we wanted to build on this basic functionality, we could use modules like:

Net::FTP::Recursive—to put and get entire directory structures
Net::FTP::AutoReconnect—to reconnect on failure
Net::FTP::Throttle—to limit the bandwidth used during transfers
Net::FTP::Find—to walk directory trees on a remote FTP server

But as I mentioned, FTP is a bit passé. All the cool kids are most likely using something else. Let's look at two of the alternatives.

Secure Shell (SSH/SCP/SFTP)

Moving files around in a secure fashion these days is commonly done either via an SSH-based method or a SSL/TLS-based method. The latter will be discussed in a future column, so let's focus for the moment on SSH-based methods. The two SSH-based methods that are almost always used are SCP and SFTP. If I had my druthers I'd just show you how they are used from Perl without even bothering to mention anything about the details of the SSH implementations they are based on. But, alas, in order to understand which ones are appropriate under which circumstances, we're going to have to have a little chat about the types of Perl SSH modules.

The trickiest thing about using SSH-based modules in Perl is deciding which one of the several branches to use. Let's meet the three contestants:

1. Perl-ish implementations—The best example of this type is Net::SSH::Perl, which aims to provide a complete SSH1 and SSH2 implementation in Perl. I say "Perl-ish" because in order to handle some of the more computationally intensive work, Net::SSH::Perl depends on other modules that either call external libraries or are partially implemented in C.
2. Thin veneer around the libssh2 (www.libssh2.org) library—libssh2 describes itself as a "client-side C library implementing the SSH2 protocol." The module Net::SSH2 lets you use all of its power from Perl.
3. Wrappers around existing SSH binaries—Modules like Net::OpenSSH provide pleasant Perl interfaces to what essentially amounts to "shelling-out" to call the appropriate SSH client binary. (This isn't necessarily as bad as it initially sounds, as we'll see in a moment.)

So, how do you choose between them? It really depends on what is most important to you. The first option is good if you don't want to be concerned about installing SSH client binaries or the libssh2 module. The minus of that option is it has quite a few dependencies, some of which used to be a bear to get built and installed. I haven't tried installing it in a while, though, so perhaps this has improved some over time.

The second option is good if you like the notion that there is a single library that does all of the heavy lifting connected to relatively straightforward Perl code to use it. Its drawback is that you have to trust that the library is solid, well tested, and performant enough for your comfort level. The libssh2 team has made excellent progress over the past year or so, so perhaps this is less of a concern.

Finally, the last option is good if you approve of the idea that the Perl code will be depending on (e.g., in the case of Net::OpenSSH) binaries that have undergone a tremendous amount of scrutiny for security issues and are in active use by countless numbers of people every day as a crucial part of their work. The other plus of these modules is that they reuse any existing SSH-related configuration (like config files and SSH keys) you already have in place. As I alluded to above, its minus relates to having to spin up a separate secondary process each time work needs to be done. Net::OpenSSH tries to mitigate that performance hit to a certain extent by making use of OpenSSH-specific multiplexing. This lets you run multiple ssh commands over the same session without having to spin up a new session for each. That can be a big efficiency win in certain cases.

Now that we've discussed the plumbing underneath the SSH file transfer modules, let's talk about what's available in that space. If you want to use SCP, the first file transfer protocol available with SSH, there are two options which both come out of category #3 above: Net::OpenSSH and Net::SCP. Here's an example of using the former to transfer a file:

```
use Net::OpenSSH;

my $ssh = Net::OpenSSH->new($host);
$ssh->error and
  die "Couldn't establish SSH connection: ". $ssh->error;

$ssh->scp_put("localfile", "/remotedir")
  or die "scp failed: " . $ssh->error;
```

With the introduction of SSH version 2 came SFTP, an additional file transfer protocol that mimicked FTP in its interface. There are a greater number of

Perl SFTP options than there are SCP. These include Net::SFTP (built on top of Net::SSH::Perl), Net::OpenSSH (yup, it does both), Net::SSH2::SFTP (built on Net::SSH2), and Net::SFTP::Foreign (which calls the already installed ssh binaries). At the moment, my module of choice in this space is Net::SFTP::Foreign, because it is the most comprehensive of the bunch and relies on binaries I already trust. Here's some code that performs the same task as the previous sample:

```
use Net::SFTP::Foreign;
my $sftp = Net::SFTP::Foreign->new($host);
$sftp->error
  and die "Unable to establish SFTP connection: " . $sftp->error;

$sftp->put("localfile", "/remotedir")
  or die "put failed: " . $sftp->error;
```

Net::SFTP::Foreign also has cool methods like find() to search the remote file system, mget()/mput() to transfer multiple files based on wildcards, and readline() to read a line at a time from a remote file.

As an aside before we move on, there are a few spiffy SSH-based modules we won't be able to cover here, like SSH::Batch and SSH::OpenSSH::Parallel, both for running commands on multiple hosts in parallel. Check them out on CPAN if you are interested.

Rsync

I've often said that I think one of Andrew Tridgell's greatest gifts to sysadmins (and to others) is rsync, both the tool and the algorithm. (Note: I'm cheating a bit by including this in a list of protocols, but bear with me.) At LISA '10, I asked my classes if anyone in the room hadn't yet heard of rsync. I was pleased to see no hands raised. If you happened to be hanging out with Plato in his cave and haven't heard of rsync, the one-sentence summary might be something like "rsync is a very efficient file transfer utility based on an algorithm that lets it send just differences between the source and the destination over the wire." It can use other transports (e.g., SSH) for security if desired. More info can be found at rsync.samba.org.

The types of Perl rsync modules pretty closely mirror those we talked about for SSH. The one difference is there really is no good option for #2 above (i.e., using a Perl wrapper around a library) because librsync itself (sourceforge.net/projects/librsync/) is no longer actively maintained and hasn't seen a new release in over four years. What we do have is an all-Perl implementation called File::RsyncP and a wrapper around the standard rsync client called File::Rsync. I think you'd have to have a good and probably specialized reason to use the pure Perl version (e.g., it was originally written to be used as part of BackupPC), so here's an example from the File::Rsync docs:

```
use File::Rsync;
$obj = File::Rsync->new( { 'archive'    => 1,
                           'compress'   => 1,
                           'rsh'        => '/usr/local/bin/ssh',
                           'rsync-path' => '/usr/local/bin/rsync' } );

$obj->exec( { src => 'localdir', dest => 'rhost:remdir' } )
  or warn "rsync failed\n";
```

If it looks to you like we're basically just writing out the standard options to the rsync command using Perl syntax, that's exactly right.

What About the Rest?

Oh dearie me. We're about to run out of space and there are still tons of things we haven't talked about. For example, you probably heard of this newfangled protocol for moving data around called HTTP. I predict it will be pretty big by the time you read this. There's probably going to be a whole other column in our future on just that protocol.

I don't know if this will mollify you until then, but let me end this column by handing you the keys to your very own nuclear device. If you want to have some fun, in a Dr. Strangelove kind of way, you might want to check out Net::CascadeCopy. In this column we've talked about moving files from one place to another, but largely in a "onesies-twosies" approach. What if you want to move data to a kerjillion servers? Net::CascadeCopy is designed to propagate files as fast as possible. Instead of a hub and spoke arrangement (a server copies the files out to all of its clients), this module turns every client into a server. As the doc says, "Once the file(s) are [*sic*] been copied to a remote server, that server will be promoted to be used as source server for copying to remaining servers. Thus, the rate of transfer increases exponentially rather than linearly." Important: Be careful with this thing. If you melt down your entire network using this module, the Secretary and I will disavow any knowledge of your actions.

Take care, and I'll see you next time.

Pete's All Things Sun: Comparing Solaris to RedHat Enterprise and AIX—Virtualization Features

PETER BAER GALVIN

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.

In our last episode, there was a battle royal comparing the general features of Solaris, RHEL, and AIX. In my view, there was no clear winner. Each had pros and cons, but they were all quite usable and feature-rich in the end. However, the battle does not end there. One area, perhaps the most important differentiator, was left for this, the final round.

Virtualization is extremely important to datacenter managers, providing a valuable tool to increase system use, reduce the number of systems, reduce costs, and increase manageability. It's also a complex topic, with technologies and features varying greatly, and few clear "best" approaches. Further, software vendors are increasing the importance of virtualization, in many cases limiting the licenses needed for an application to only the CPUs within a virtual machine in which the application runs. The Oracle Database, for example, running in a capped Solaris Container, needs fewer licenses than if the container were uncapped, potentially saving the datacenter quite a bit of money. Given its variations, complexity, and importance, it makes sense to give the virtualization battle a column of its own.

Comparison

For a full description of why these three operating systems are included and others are not, please see my December 2010 *;login:* column, which also provides an overview of each operating system and potential reasons to run each. Fundamentally, AIX 7.1, Solaris 10 9/2010, and Red Hat Enterprise Linux 5.5 are going to continue to be important operating systems, each having a potentially bright future (even if for different reasons). The purpose of this two-part column is to help readers sort through these operating system choices and provide a basis for comparison and consideration for use.

Virtualization is certainly many things to many people, but included here is hardware and operating system–provided virtualization. Certainly, Solaris and RHEL can also be virtualized by other technologies, such as VMware ESX and VirtualBox, and those are valid choices. However, those are not features of the selected operating systems and therefore are outside the scope of this column. Within the scope, though, is hardware virtualization, and even if Solaris only has that feature available on SPARC hardware, RHEL has no such feature, and AIX has that feature on all implementations (as it runs only on the Power CPU and Power has hardware virtualization). Given those differences, why is hardware virtualization "allowed" in this comparison? Fundamentally, most Solaris sites run Solaris on SPARC hardware, and many of those use the hardware virtualization features.

Likewise, AIX shops make extensive use of the Power hardware virtualization features. Hardware virtualization is an important tool available to administrators of those systems, and therefore must be considered when looking at the complete virtualization picture.

Virtualization

The gamut of virtualization technologies to consider is therefore hardware, multiple-OS software, and single-OS software. Those three are further explored in the next three sections.

Oracle/Sun Hardware Virtualization

In the Sun world, this is "Dynamic Domains" and "LDOMs" (now renamed by Oracle to "Oracle VM Server for SPARC," but called LDOMs here for brevity). This feature is implemented at the hypervisor level. By default, all hardware is seen in one chunk (for lack of a better word). The hypervisor allows the hardware, including CPUs, memory, I/O controllers, and system disk drives, to be divvied up into two or more chunks. Each chunk gets its own Solaris install and can run different OS releases and patch levels, different applications, and so on. Only Solaris 10 and Solaris 11 Express are supported on the M and T servers, however, so operating system choice is somewhat limited. For almost all intents, the given server acts like multiple servers that just share the same sheet metal.

LDOMs existing only on Sun "T" servers, and sharing the same motherboard, can have single points of failure. Dynamic Domains existing on M servers do not generally share single points of failure. Dynamic Domains and LDOMs are both somewhat dynamic, in that some resources can be moved between the chunks without downtime. The number of allowed Dynamic Domains varies based on the M-server model, from zero on the M3000 through two on the M4000 (a minimum of two CPUs per domain) to 24 on the M9000. The number of LDOMS is limited by the number of threads in the T-server box. A T3-1 has one socket containing 16 cores and 8 threads per core, for a total of 128 threads. LDOMs can be as small as one thread, for a possible total of 128 in the T3-1. Practically, one LDOM per core is more reasonable and common than one LDOM per thread.

Each Dynamic Domain has its own I/O controllers and is independent of other Dynamic Domains to perform its I/O. LDOMs need to have one or two "I/O Domains" which include all of the I/O controllers and have all I/O routed through them. If there are two I/O Domains, then they can be configured to provide redundant paths to the devices for high availability.

IBM Power Hardware Virtualization

IBM Power servers have LPARs (Logical PARtitions). While Dynamic Domains and LDOMs are included with the cost of the Oracle server, IBM frequently has an additional cost for features such as virtualization. LPARs are more flexible than LDOMs in that they can be configured to the single core increment, up to a maximum of the number of cores in the system, as well as in micro-partitions. Most Power servers have a "micro-partitions" feature in which an LPAR can be split into up to 10 sub-chunks. There are some fixed limits: for example, currently the IBM Power 790 can only have 256 partitions (a combined total of LPARs and micro-partitions). That is still a large number, though. And even the micro-partitions are full hardware instances in that they have their own copy of AIX installed.

AIX versions 5.2 and beyond are supported on most Power servers, as are RHEL and SUSE, so a wide variety of operating systems can be running within the same system.

LPARs can act like Dynamic Domains and have dedicated I/O, or have a "Virtual I/O Server" partition which manages I/O and which other LPARs and micro-partitions talk to to get I/O performed. It can also mix these two options. There does not seem to be a way to make the Virtual I/O Server highly available, but a Virtual I/O Server is running AIX and can be configured for multi-pathing within the LPAR. As with Dynamic Domains, resources may be moved between LPARs. LPARs with that feature enabled are known as Dynamic LPARs or DLPARs.

There are some features provided by IBM Power hardware virtualization that are not found within Oracle's Sun products. The most impressive and useful of these is "Partition Mobility," which can move an LPAR or micro-partition between Power servers without disrupting service, taking a few seconds. Partition Mobility works for all operating systems, including Linux. There is also an "inactive" version of Partition Mobility that makes it easy to move a halted partition between servers. Of course, all storage used by the partition (the boot disk as well as data disks) must be able to be mounted on the target server, via FC SAN or iSCSI connectivity. Also, all I/O must occur via the Virtual I/O Server.

Active Memory Sharing is another interesting feature, in which a pool of memory is configured and that pool is made available to multiple partitions. This can be simpler and more flexible than moving memory between partitions.

PowerVM Lx86 provides a supported solution for running Linux x86 binaries within a Power partition. This solution involves a binary translator that converts and then caches the x86 instructions into Power instructions. No modification of the original Linux x86 binary is needed. Also translated are system calls from the x86 versions to ones compatible with the native Linux operating system running on the Power CPU within the partition. There is obviously a performance penalty incurred by translation, but the fast Power CPU probably helps decrease the impact of that translation on the performance of the x86 Linux application.

RHEL Hardware Virtualization

This section might be surprised to find itself here, but at least a bit needs to be said about the possibility of using VMWare ESXi to "hardware virtualize" RHEL. This solution to having hypervisor support running virtualized RHEL (and many other OSes) is legitimate and functional. In fact, it provides the "vMotion" feature, which enables the transfer of a running virtual machine from one VMware server to another without service interruption. ESXi itself is free, but there is a maintenance cost if desired, and features such as vMotion are only available for a price. Perhaps the biggest difference between the ESXi solution and the others discussed above is that ESXi comes from a third party and adds support complexity because a problem resolution may require calls to two support desks rather than just one. So, rather than crossing "hardware virtualization" off of the RHEL feature list, consider this option.

Of course, this is one of many third-party virtualization options, which include software solutions such as VMware ESX, Xen, and VirtualBox, and even hosting solutions such as Amazon EC2 and Joyent's Cloud. Those are well worth understanding and evaluating as part of performing virtualization planning due diligence.

Solaris Software Virtualization

Solaris includes "Containers," also called "Zones," as a core virtualization feature. Given that Containers have been discussed at length elsewhere and in previous issues of this column, only the basics are discussed here. Containers provide a secure environment in which applications run. Those applications have no knowledge of other applications in other Containers. Applications in one Container can only communicate with other Containers over network protocols. Most importantly, Containers do not run their own kernel. Rather, one kernel is used to create all Containers and all applications. There are Solaris 8 and Solaris 9 Containers, allowing a Solaris 8 or Solaris 9 system to be converted from a physical instance to a virtual instance (P to V) and run within a Container (at added cost). Containers may not reside on NFS, but can reside on block storage such as DAS, SAN, and iSCSI. Finally, Containers can be detached, and then attached to another server that sees the same storage, and can be duplicated, but they cannot be moved without service interruption.

AIX Software Virtualization

AIX has Workload Partitions, or "WPARs," which are very, very similar to Containers. Again, there are no separate kernels, applications are protected from each other, resources can be managed, and so forth. It is an AIX feature, unlike LPARs, which are a Power CPU feature. Unlike Containers, WPARs can reside on NFS storage. Also unlike Containers, WPARs can be moved without service interruption via the AIX Live Application Mobility feature. Such a move is much like VMware's vMotion.

RHEL Software Virtualization

Unlike AIX WPARs and Solaris Containers, RHEL KVM is very new, having been released in 2009, and seemingly not very heavily used in production environments. KVM replaces Xen, which was the chosen virtualization technology for RHEL 5. KVM has some impressive features, including supporting 64 virtual CPUs per guest with low overhead, as well as live migration, but the only supported guests are RHEL versions 3 through 6 and Windows 2003, 2008, XP, and 7.

KVM is included in RHEL, and also in a separate product from Red Hat named "Enterprise Virtualization." RHEV is a virtualization platform, in some ways similar to VMware ESX. It supports RHEL and Windows guests and provides virtualization features such as live migration, high availability, power management, and system scheduler (for scheduling activities). Oddly, the RHEV manager only runs on Windows. Apparently, RHEL and RHEV are in a transition period and only time will tell how enterprise-worthy these components are. As is the case with Linux in many areas, and one of its strongest benefits, there are many free choices surrounding virtualization and virtualization management. Determining which, if any, of them are production-ready can be a challenge, though.

Conclusions

If just the native, included-with-the-OS virtualization features are considered, then Solaris has a distinct advantage over the other contenders via its Domains, LDOMs, and Containers. However, expanding the possibilities to features available for purchase from the vendors, IBM puts on an impressive show with features that match all of Sun's, and exceeds them with the ability to move partitions between servers without service interruption. RHEL can only match these features when adding third-party solutions such as VMware ESXi for hardware virtualization. That is a valid choice, as long as the challenge of needing to involve another entity in support issues is balanced against the benefits of the added feature set.

Finally, RHEL 6.0 was just announced and promises to add new features to the already sound offering. For more details on what is included see http://press.redhat.com/2010/11/10/red-hat-enterprise-linux-6-a-technical-look-at-red-hats-defining-new-operating-platform/.

Tidbits

Solaris 11 Express has finally been born, after a long pregnancy and even more prolonged labor (of love?). Certainly the waiting was the hardest part, and was possibly extended by the purchase of Sun by Oracle. S11 Express has many of the features of OpenSolaris, except that there is no source code available for it. It does have advanced features such as ZFS encryption and a vastly better network functionality code named "crossbow." According to the FAQ (http://www.oracle.com/technetwork/server-storage/solaris11/overview/faqs-oraclesolaris11express-185609.pdf), S11E can be used for non-production use for free, but use in production requires a support contract. It is available for download from http://www.oracle.com/us/products/servers-storage/solaris/solaris-11-express-185123.html.

References

AIX 7: ftp://public.dhe.ibm.com/common/ssi/ecm/en/pod03054usen/POD03054USEN.PDF.

IBM Power server features: https://www-304.ibm.com/businesscenter/fileserve?contentid=205264.

AIX virtualization: http://www.ibm.com/developerworks/wikis/display/virtualization/Virtual+IO+Server.

IBM Live Partition Mobility: http://www.redbooks.ibm.com/redbooks/pdfs/sg247460.pdf.

RHEV overview: http://www.redhat.com/rhev/.

RHEL KVM: http://www.redhat.com/f/pdf/rhev/DOC-KVM.pdf.

iVoyeur: More Ganglia on the Brain

DAVE JOSEPHSEN

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

Welcome to the second installment in my Ganglia series. In the last issue I promised (threatened?) to walk you through building a Ganglia data collection module in C, and that's exactly what we're going to do.

Ganglia, as you'll recall, is composed primarily of two daemons: gmond, which runs on the client side, collecting metrics, and gmetad, which runs server-side and polls the clients, pushing their data into RRDTool databases. Gmond, out of the box, can collect a litany of interesting metrics from the system on which it is installed and can be extended to collect user-defined metrics in three different ways.

The first method to add new metrics to gmond is to collect the data yourself using a shell script, or similar program, piping the resultant data to the Gexec daemon. This method is simple and widely used, but adds overhead to a system carefully designed to minimize it. The second and third methods are to write plug-ins for gmond in Python or C. Writing gmond plug-ins in Python requires that Python be installed and that gmond be compiled against the Python libs on every box where you might want to use your module.

C modules interest me for a number of reasons. Ganglia is such a nifty design that it "feels" wrong to me to burden the system with a bunch of /bin/sh instantiations, to say nothing of the mess implied in the idea of maintaining and scheduling my own shell scripts everywhere. Further, we take great care to minimize the package count in our production environments, and I have no need for Python otherwise. C modules on the other hand are lightweight, elegant, and more likely to make their way into the corporate Git repository so that their versions can be tracked. Finally, a well-written C module can be contributed back to the Ganglia maintainers, and possibly merged into the project, which will benefit everyone who uses it in the future.

How do we write one of these things? In terms of coding strategy, we can organize data collection modules into two types: those I'll call "static" and those I'll call "dynamic." Static modules collect a known series of data metrics. Consider a memory data collector module. We know up front that we're going to be collecting a known series of metrics: free memory, used memory, free and used swap, etc.

Dynamic modules, on the other hand, are going to collect a user-defined series of metrics. The dynamic module I'm going to share with you, for example, is a process-counter module. It simply counts the number of named processes (like httpd) running on the client system. I'm referring to it as "dynamic" because the module doesn't know which specific processes you want to count beforehand.

Creating a gmond module mostly involves creating a struct of type "mmodule." This struct is made up of three function pointers and another struct containing metric metadata. Because Ganglia relies on the Apache Portable Runtime Library, many of the data structures involved are APR types, and digging more than a couple of layers deep will land you squarely in the APR headers, which you may find a bit vexing (I sure did). The module writers were sensitive to this, and have provided a few functions to insulate you from needing to know too much about the underlying APR data structures. The problem is, to use these functions you need to statically define the metric metadata up front. So the primary difference between writing modules that I call "dynamic" and those I call "static" is how much you'll need to interact with APR. You'll see what I mean as we continue.

Let's take a look at the memory module that comes with the Ganglia distribution, the full source of which is available in the gmond/modules/memory directory of the Ganglia tarball or at http://www.usenix.org/publications/login/2011-02/mod_mem.c. First, take a look at the very bottom of the file:

```
mmodule mem_module =
{
        STD_MMODULE_STUFF,
        mem_metric_init,
        mem_metric_cleanup,
        mem_metric_info,
        mem_metric_handler,
};
```

As you can see, aside from the global STD_MMODULE_STUFF, which is the same for every module, the struct is made up of a pointer to a struct called mem_metric_info, and three pointers to functions, called mem_metric_init, mem_metric_cleanup, and mem_metric_handler. Your module needs to define all three functions, but, as you'll see below, the metric_info struct can be dynamically created later if you're writing a "dynamic" module. This happens to be a static module, so the metric_info struct is statically defined:

```
static Ganglia_25metric mem_metric_info[] =
{       {0, "mem_total", 1200, GANGLIA_VALUE_FLOAT, "KB", "zero", "%.0f",
UDP_HEADER_SIZE+8, "Total amount of memory displayed in KBs"},
        {0, "mem_free", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "Amount of available memory"},
        {0, "mem_shared", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f",
UDP_HEADER_SIZE+8, "Amount of shared memory"},
        {0, "mem_buffers", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f",
UDP_HEADER_SIZE+8, "Amount of buffered memory"},
        {0, "mem_cached", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f",
UDP_HEADER_SIZE+8, "Amount of cached memory"},
        {0, "swap_free", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "Amount of available swap memory"},
        {0, "swap_total", 1200, GANGLIA_VALUE_FLOAT, "KB", "zero", "%.0f",
UDP_HEADER_SIZE+8, "Total amount of swap space displayed in KBs"},
#if HPUX
        {0, "mem_arm", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "mem_arm"},
        {0, "mem_rm", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "mem_rm"},
        {0, "mem_avm", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "mem_avm"},
        {0, "mem_vm", 180, GANGLIA_VALUE_FLOAT, "KB", "both", "%.0f", UDP_
HEADER_SIZE+8, "mem_vm"},
#endif
        {0, NULL} };
```

Ganglia_25metric is defined in lib/gm_protocol.h. The fields from left to right are:

int key	I'm not sure what this is for, but setting it to zero seems safe.
*char *name*	the name of the metric, for the RRD
int tmax	the maximium time in seconds between metric collection calls
Ganglia_value_types type	used by APR to create dynamic storage for the metric, this can be one of: string, uint, float, double or a Ganglia Global, like the ones above
*char *units*	unit of your metric for the RRD
*char *slope*	one of zero, positive, negative, or both
*char *fmt*	A printf style format string for your metric which MUST correspond to the type field
int msg_size	UDP_HEADER_SIZE+8 is a sane default
*char *desc*	A text description of the metric

Gmond will read the mmodule struct at the bottom of the file and then call the init function contained therein. The init function in turn calls two other important functions: MMETRIC_INIT_METADATA and MMETRIC_ADD_METADATA for each element in the metric_info struct I pasted above. A few APR-related things have been done for us, based on the fact that we defined that metric_info struct, but we don't need to worry about that now, because INIT_METADATA and ADD_METADATA will interact with APR for us. So for each metric listed in our metric_info struct, INIT_METADATA initializes a row of APR storage for our metric and ADD_METADATA calls our handler, which causes data to be collected and stored for our metric.

Examining the handler function, we find that it's a simple switch-case, which is using the index number from the metric_info struct we defined to decide what data to go gather. If you look at any of these data-collection functions (e.g., mem_total_func();), you'll find that they simply read the required information out of the proc file system on Linux.

That's pretty simple. The mmodule struct at the bottom calls init, which calls MMETRIC_INIT_METADATA and MMETRIC_ADD_METADATA once for each element defined in metric_info. MMETRIC_ADD_METADATA in turn calls our handler function, which executes a different function depending on the current index number of the metric_info array. That works for pretty much any physical metric that one can imagine, such as memory, CPU, temperature, or network metrics. All of these metrics are understood up front and may be defined as such.

Things get a little more complicated when we can't predict what the metric_info struct will look like. In the case of my process counter (full source available at http://www.usenix.org/publications/login/2011-02/mod_count_procs.c), an end user is providing us with the names of one or more processes to be counted, so we won't know until runtime what the metadata in the metric_info struct will look like. Thus, we need to programmatically (or dynamically) generate the metric_info struct at runtime. My mmodule struct definition looks like this:

```
mmodule cp_module =
{
        STD_MMODULE_STUFF,
        cp_metric_init,
        cp_metric_cleanup,
        NULL, /* Dynamically defined in cp_metric_init() */
        cp_metric_handler,
};
```

Since we didn't name the metric_info struct, we need to create it manually in our init function. This requires a bit more interaction with APR. First, I globally declare metric_info at the top of the file:

```
static apr_array_header_t *metric_info = NULL;
```

I also need a Ganglia_25metric pointer inside the scope of the init function, so that I can modify the contents of individual array elements inside metric_info:

```
Ganglia_25metric *gmi;
```

Then I can create the struct using the apr_array_make function:

```
metric_info = apr_array_make(p, 1, sizeof(Ganglia_25metric));
```

p is the name of an APR memory pool that we've been passed from gmond. The second argument is the initial size of the array, and the third argument tells APR what kind of elements this data structure is going to store. It's important that the third element be defined clearly in this manner.

We set gmi to the next available slot in the metric_info stack by calling apr_array _push:

```
gmi = apr_array_push(metric_info);
```

So the general strategy is to read a list of process names from user input and then iterate across them, calling apr_array_push, and manually populating gmi with the metric details we need.

Gmond.conf, the config file for gmond, allows the user to pass parameters to modules with configuration entries such as this one:

```
module {
        name = "cp_module"
        path = "modcprocs.so"
        Param ProcessNames {
           Value = "httpd bash"
        }
}
```

My init function reads these user-supplied parameters using gmond-supplied functions:

```
if (list_params) {
        params = (mmparam*) list_params->elts;
```

```
                for(i=0; i< list_params->nelts; i++) {
                    if (!strcasecmp(params[i].name, "ProcessNames")) {
                        processNames = params[i].value;
                    }
                }
        }
```

I use the infamous strtok() function to parse out the space-separated list of process names and iterate through them, first getting a new spot on the metric_info stack and then modifying its contents:

```
        for(i=0; processName != NULL; i++) {
                gmi = apr_array_push(metric_info);
                gmi->name = apr_pstrdup (p, processName);
                gmi->tmax = 512;
                gmi->type = GANGLIA_VALUE_UNSIGNED_INT;
                gmi->units = apr_pstrdup(p, "count");
                gmi->slope = apr_pstrdup(p, "both");
                gmi->fmt = apr_pstrdup(p, "%u");
                gmi->msg_size = UDP_HEADER_SIZE+8;
                gmi->desc = apr_pstrdup(p, "process count");
```

Now that the element has been dynamically created, it can have INIT_METADATA, and ADD_METADATA called against it, just like its static brethren:

```
        MMETRIC_INIT_METADATA(gmi,p);
        MMETRIC_ADD_METADATA(gmi,MGROUP,"cprocs");
```

Once we've iterated across every process the user wants us to count, we put an empty terminator on the metric_info array with one last call to apr_array_push, and then we manually set our metric_info array to be the array used by the mmodule struct down at the bottom of the file (the one we'd initially declared as void) with this line:

```
        cp_module.metrics_info = (Ganglia_25metric *)metric_info->elts;
```

Unlike the handler function in the memory module, which calls a different function for each metric in metric_info, my handler always does the same thing. That is to say, it always counts how many of the named process are running. To do the actual counting I've added a function called count_procs, which uses the procps library and is beyond the scope of this article. I do want to point out that my handler function is still using the same metric_index number that the memory module uses, but instead of using it in a switchcase with a function per index number, it's simply using it to de-reference the process name from the metric_info array:

```
        Ganglia_25metric *gmi = &(cp_module.metrics_info[metric_index]);
        count_procs(gmi->name);
```

That about covers the workings of both types of modules. Once written, I compiled my module on a Linux box from within the Ganglia tarball gmond/modules/cprocs directory:

```
        cc mod_count_procs.c -shared -I/usr/include/proc -I/usr/include/apr/
            -I../../../include/ -I../../../lib -lproc-3.2.7 -o modcprocs.so
```

I included /usr/include/proc for the count_procs() function, which uses procps.

In addition to the configuration I've already mentioned in gmond.conf which tells gmond to execute the module and optionally passes parameters to it, gmond also needs to be told to schedule polling for the metrics by adding metric parameters to a collection group:

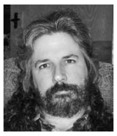
```
collection_group {
        collect_every = 80
        time_threshold = 950
        metric {
           name = "httpd"
           value_threshold = "80"
           title = "HTTPD Processes"
        }
        metric {
           name = "bash"
           value_threshold = "100"
           title = "BASH Processes"
        }
}
```

Once the module has been compiled and copied into the modules directory (/usr/lib/ganglia/ by default) and the configuration file updated to use it, gmond will report the new metric, and gmetad will create an RRD for it and display it on the front-end. Speaking of front-ends, as I write this, the Ganglia devs are currently going through a user interface rewrite which will update the UI with many of the features I wished for in my last article, including the ability to modify graphs by passing RRDTool options in a URL. The more I use Ganglia, the more I like it, and I heartily recommend checking it out if you haven't yet had the opportunity.

Take it easy.

/dev/random

ROBERT G. FERRELL



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

Wading through all the ballyhoo about the Stuxnet worm being a "game-changer" (what game would that be, exactly?) by dint of its very precise targeting, sophisticated coding, and really rad fashion sense, my thoughts began to wander (the way they always do) down the well-beaten path of baseless speculation. Apparently/reportedly/allegedly Stuxnet was created to, among other things, mess with the rotational speed of the specific centrifuges in use by the targeted nuclear fuel processing facility in an unnamed country that rhymes with Tea Lawn. Leaving aside the glaring question of why anyone would want to make centrifuges network-accessible—and, for that matter, how they would go about it—I want to turn instead to a fascinating (and I use that adjective because even in this enlightened day and age we're not allowed to print the more satisfying one) exploration of other applications of this, um, technology.

Before I proceed further, I would like to snatch yet another moment of your precious time to discuss the whole concept of computer worms, or, rather, the nomenclature thereto appertaining. Why a *worm*? Worms, or anyway the worms I've encountered in my backyard with spade in hand and crabgrass demise in mind, can wriggle and flop with the best of them but intimidation or propulsion-wise they're not on top of the game. I would think computer *eel* or computer *adder* or even computer *spirochete* would be more descriptive and menacing than just plain ol' *worm*. I wasn't there when that train left the station, though; I guess the fella what calls the turn names the baby.

If some unidentified malware incubator can spit out a worm to take over centrifuges in a uranium-enrichment plant, that same coding acumen could be applied to something more useful to the average Joe such as, say, getting better seats at the ball game or lowering the price on that expensive flat-screen HD. What if you could gain control over the local amusement park infrastructure long enough to change the duration or max speed on your favorite roller coaster? SCADA systems are just about ubiquitous these days, probably far more so than most people understand. All sorts of big, important, dangerous doo-dads are connected to the Interwebs now, including flood control gates, electrical grid routing substations, distillation/cracking systems at refineries, chemical manufacturing equipment—just about anything with a flow rate, liquid level, temperature, voltage, rotational velocity, or other parameter that can be measured and controlled via feedback loops. Not to mention your garage light, blender, microwave, and dog flap.

Throughout the history of American pop culture (it doesn't officially become *pop* culture until your pop no longer understands it), or at least the history of American pop culture from the early 20th century onward (and prior to that I don't think we had much, since pop culture requires widespread media access to propagate), the litmus test for things that we as men feel powerless to deal with has been the superhero (women resort to shopping). Like canaries in a coal mine, the appearance and sudden popularity of a Superman or Batman or Mighty Mouse (a nod to Steinbeck; you figure it out) has heralded a conscious or unconscious collective conclusion by the creative/imaginative segment of the populace that the Powers That Be ain't cuttin' the mustard from a problem-solving perspective. Captain America slew the Nazis for us, for example. Superman took care of all those pesky urban criminal gangs and served as a sort of science fiction bellwether. Batman explored our profound sense of moral ambiguity concerning violence perpetrated in the name of justice. Plus, he wore really cool black outfits and had a fabulous black car with big tail fins.

Now that I've segued so neatly from SCADA to Batman, I think it's time that a certain sidekick finally came into his own. I must say I never understood the point of Robin. What the heck kind of hero gets named after a migratory thrush, anyway? (The North American robin is not related to the European one, FYI.) I mean, why would a robin hang around with a bat in the first place? Robins aren't nocturnal; they're birds, not mammals; they navigate by sight, not sound. What's up with that bizarre pairing, then?

I think I've uncovered the answer: Kane and Finger were prescient. That stuff they put out at the time about Robin being named for Robin Hood (hence the pseudo-Medieval Hollywood costume) was just a cover story. Robin really *was* named for the bird because...and this is where genius rears its glittery head...they anticipated the need one day for a superhero *who eats worms*.

That's right, folks: remember, you heard it here first. Robin is the obvious superhero for the age of self-propelled malware. While Batman dukes it out with insane circus clowns and fugitive animated store window manikins from "Polyester Billy's Short 'n' Waddly Men's Wear," the Boy Wonder takes on the latest polymorphic encrypted code monster. Even zero-days are no match for the (Es)caped Crusader and his new protégé, *Early Bird*. Thrill as they dodge razor-sharp distributed denial of service packets. Gasp as our heroes narrowly avoid being ensnared by the heinous *Spearfisher*. Perch on the edge of your chair, wring your hands, and postpone going to the bathroom until your eyes cross repeatedly while the insidious *Identity Thief* works his oily wiles on the unsuspecting citizens of Webtropolis. Don't forget to visit our snack bar and bring your automobile title.

The pundits say that increasingly sophisticated worms are an inevitable consequence of our interconnected world. A hero will rise to struggle against these vile abominations in the name of goodness and low-calorie sweetener and that hero will be clad in crimson vest, green tights, and antique gold satin tablecloth.

All hail the stalwart savior of SCADA, the worm's nemesis: *Robin the Avianator*.

Book Reviews

ELIZABETH ZWICKY, WITH RIK FARROW AND SAM STOVER

Lately, I have actually been asked the same questions repeatedly, so it seems like a good moment to provide a few answers; you could call this a "FAQ" list if the term hadn't been co-opted by most Web sites to mean "questions we were hoping you might ask frequently, or at least would read the answers to if we put them near a title implying you would definitely be interested in them."

How do I read so many books? I'm afraid the answers are boring and not very useful; first, I read faster than most people, and I always have, and second, I don't own a television.

Why an iPad? Sadly, this answer is probably just as useless for most people, because it isn't a decision based on research on eBook readers in general. I like the size, and the user interface is familiar to me. I don't expect it to be a whole computer, but I like having more than just books available to me (when sick, playing silly games is almost as soothing as reading fiction, which explains my score at Heroes of Kalevala). My child is passionately fond of it, which is both an advantage and a downside. It will also read pretty near every eBook format ever, definitely an advantage.

Does this eBook stuff really work? It works for me. I'm certainly helped out by the big screen, which means that books in PDF format show up almost exactly as they would on paper, but it's still striking how fast you get used to paging electronically. You don't get the same cues about where you are in the book, which makes it harder to find things again. Different readers and different formats will have very different effects, of course.

I tried one of this month's books both in PDF and in ePub. There are two big differences between the formats. PDF is a familiar problem for publishers, and almost guaranteed to be very close to the printed book, probably being produced in the regular publishing flow. ePub is really only used for eBooks, and the production process will be a separate one, which often results in new errors (for instance, the book I read was named "epub" in ePub format, but was correctly titled in both the PDF readers I use). But ePub is reformattable, which means that if you have a small screen, the text can reflow, so that it's a reasonable print size and still fits on the screen. PDF doesn't do that; it's pictures of pages. For books that are all text, this is perfect. It gets a little trickier with pictures inserted; unless the book is actually redone for ePub, they're going to end up where they got put in the now irrelevant page layout. It's livable, and aside from the title, the book I read appeared to be accurate.

Being Geek

Michael Lopp

O'Reilly, 2010. 309 pp.

ISBN 978-0-596-15540-7

This is the best book on career skills for high-tech people that I've come across. Its prototypical reader would be a developer on about his second job, but there's plenty here for people earlier and later in their careers, and it doesn't require much translation for people in support and/or operations. (And yes, I did say "his" on purpose, but it's actually reasonably unbiased; the author is a man writing about a male-dominated world, but he doesn't omit women or treat them as a different species.) It covers a wide range of topics, from interviewing to deciding when to leave a job, and it includes managing both up and down, and giving presentations.

This is a lot of area to cover. As a result, it's a speedy and opinionated ride. It's in short pieces, not closely tied to each other and easy to read separately. Most technical people will find it both enjoyable and informative; some of the advice won't suit you, but that's true of all advice books.

Glitch

Jeff Papows

Pearson Education, 2010. 198 pp.

ISBN 978-0-13-216063-6

This is a book for IT managers. It has a lot of good stories, some good insight into trends, and some good advice, surprisingly loosely linked together, and is totally centered on what IT management can do to improve life. I find some of the takes on things odd. I believe the assertion that there is a lot of COBOL code still in production, but ever fewer people able to maintain it. I am dubious that the problem to be solved here is a shortage of COBOL programmers rather than an oversupply of COBOL code, and even more skeptical about the prospects for bribing people to learn COBOL by also letting them learn something new and flashy.

If there's an IT manager in your life who needs to be scared straight, this is the book to do it, full of moral tales about past failures, suggestions about upcoming disasters, and virtuous advice. It doesn't have a lot of interest for a more general audience.

Data Analysis with Open Source Tools

Philipp K. Janert

O'Reilly, 2010. 498 pp.
ISBN 978-0-596-80235-6

I love this book a lot. Practical data analysis is not at all the same thing as theoretical data analysis, and there are almost no resources to help you with it. Even people who can do it effectively are rare, most people being either unfamiliar with the tools or far too enamored of sophisticated methods when knife-fighting is called for. And this is a whole book that devotes itself to truly practical data analysis, the sort of book that will tell you how to figure out that it's not worth optimizing a process, and then turn around and point out to you that often nobody wants to hear that.

It includes a short introduction to various tools, including R, and a number of Python packages for scientific programming. It assumes that you're capable of programming in any old programming language (and provides appropriate help with R, which is not any old programming language). It also has a lovely annotated bibliography at the end of each chapter with suggestions of other resources, one of my favorite features.

My first warning is that the introduction says you need to be "not math-phobic." In general, when books say this, they mean they have numbers in them. In this case, it means that calculus comes up every few pages, and there is an excursion into eigenvalues (which you can skip). There is an appendix which explains calculus, but an appendix is not going to help that much if you didn't have any calculus to begin with, which, I have to admit, I don't. Nonetheless, I managed to get a great deal of use out of this book.

My second warning is that if you're not already doing data analysis, you may be unable to figure out why I'm so enthusiastic. First, you've got to get the data into an analyzable state, which it doesn't help much with, and then you have to figure out what questions to ask. There's some help with that, but most of it is aimed at particular sorts of situations; apparently the data analysis I mostly do (which can generally be summarized as "something is wrong, here's a pile of data") is relatively rare, because it's not much like the situations most discussed. I don't blame the book for these omissions, which are big complicated areas in their own right, but nonetheless, there are going to be plenty of people who are still bewildered.

Building the Perfect PC, 3d Edition

Robert Bruce Thompson and Barbara Fritchman Thompson

O'Reilly, 2010. 342 pp.
ISBN 978-1-449-38824-9

I like to build my own systems, so when O'Reilly announced this book, I asked for a copy immediately.

Confession first: unlike Elizabeth, I don't read that quickly and so I only read the first two chapters (up to page 87), and then skimmed later chapters. Having written that, I found the Thompsons' book the best I've read when it comes to building PCs. They cover components in detail, offer recommendations (that I agree with no less) for best vendors, and explain many things that I often wondered about. For example, how big a power supply do you really need? I've generally gone for the smallest power supply that seemed to cover the requirements of my components, but they explain that a power supply that is rated at 400 watts will often have a maximum efficiency at 200 watts. I had thought I would be wasting power with an "oversized" power supply, but I was doing just the opposite: I was wasting power by operating my power supply outside of its efficiency envelope. They also point out that running a power supply near its maximum will result in it failing sooner—sometimes much sooner.

The authors offer great advice about picking CPUs, with explanations of different classes of CPUs, why paying a premium for 10% performance will not be satisfactory for most builders, recommendations for sockets with an upgrade path, and matching CPUs to the tasks you expect from them. They cover all components in detail, and although they didn't get into the nitty-gritty details of memory (just one mention of CAS), they don't really need to, as your only option is to buy memory that is supported by your motherboard. I've been bitten by this, also by buying a CPU that would fit the motherboard socket but not be supported by the motherboard's BIOS (grrrr), so this is important advice.

The Thompsons provides specs for building six types of systems: budget PC, mainstream system, extreme system, media center, appliance/nettop, and home server. They include rationales for all of the component decisions, and all the information you might need to assemble these systems properly. On top of all of this, they use Linux (Ubuntu), so if your plan is to install Linux instead of Windows 7, you know you are working with people who have the same notion that you do about operating systems. This includes recommendations for Linux media center software.

I plan to continue reading this book, as I build a system almost every year. Last year it was an Atom appliance (that I could buy a motherboard with a processor, video, sound, and SATA support for $64 just astounded me), and this year it will likely be a better desktop system. Robert's bio says he has built or repaired hundreds of PCs, and I believe him. The details make this book worth the price of avoiding the many problems you can run into when building your own PC. Oh, and building your own PC will not cost you any more, and you will usually wind up with better components than if you bought a vendor-built system.

—Rik Farrow

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

Deviant Ollam

Syngress, 2010. 236 pp.

ISBN: 978-1-59749-611-7

Finally. No, I mean *finally*, a great lockpicking book. I had high expectations when I ordered it, and it overdelivered. If you want to learn how locks work, how to pick locks, and how to actually get good at it, this is the book for you. Even if you don't have any interest or need to learn about physical penetration testing, picking locks is a challenging hobby.

For the longest time, the resources available to aspiring lockpickers were somewhat limited. There are a couple of older books, and some trade schools offer distance-learning locksmith courses, but a lot of the "juicy" info just wasn't available to the general public. In fact, in some states, possession of a lockpick set can be a crime, if you aren't a licensed locksmith. That said, owning this book isn't going to get you into trouble, but practicing on locks that aren't yours certainly could, which is why I was glad to see an opening section dealing with "Ethical Considerations."

Chapter 1 discusses the how pin tumbler and wafer locks work, and Chapter 2 explains the weaknesses inherent in the locks which allow for picking. You have to love Chapter 3, which is titled "Beginner training: How to get very good, very fast" and delivers on that promise. The most common type of lock is the pin tumbler, and most (decent) locks have four or more stacks of pins. This chapter shows you how to set up a practice lock with only one stack to start, then progress to four (and beyond), with suggestions on how to make the picking easier or harder as you practice, practice, practice.

Chapter 4 goes one step further into advanced training and more complex locks and tools. Chapter 5 discusses "Quick-entry tricks" like shimming and bumping—and even shows you how to make your own shims. Chapter 6 takes a look at some other types of locks, such as tubular (the kind you see in a soda machine), as well as two locks not seen much in North America: Cruciform and Dimple locks.

The Appendix contains 20 pages of pictures and descriptions of tools and tool sets. There is even a DVD with 1.5G of animations, figures, and videos. All of the figures in the book are given in color, and the videos are from various presentations and examples (e.g., shmoocon). All in all, a fantastic resource. The book is fairly short, as the important topics are presented efficiently and with a dash of humor. As mentioned before, the only real way to get better is to practice, and there is a lot of attention given to providing ideas and methods for everything from getting started all the way to designing almost impossible scenarios. Unlike the "white and orange" Syngress books of the past, the new "black and yellow" design seems to indicate a new trend in quality. This book was extremely well written and edited. 10 out of 10.

—Sam Stover

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

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

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

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

Discounts on registration fees for all USENIX conferences.

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

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

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

USENIX Board of Directors

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

Have You Seen the Latest USENIX Short Topics in Sysadmin Books?

Check out the latest titles:

Best-seller! #21: *Foundation for Cloud Computing with VMware vSphere 4,* by John Arrasjid, Duncan Epping, and Steve Kaplan

Virtualization is recognized as a foundation for cloud computing. This book is an overview of the VMware technologies and how they can support the various services and management pieces required for cloud architecture. Without diving overly deeply into specific design patterns, it provides insight into the tools to fit your design criteria.

New! #22: *Job Descriptions for System Administrators, 3d Edition,* by Tina Darmohray

In 1993, SAGE, as the professional organization for system administrators, created a set of appropriate job descriptions for system administrators. Recently brought up to date, this third edition of *Job Descriptions for System Administrators* includes a set of manaagement-level descriptions. It includes everything needed to write your job description or hire a sysadmin: core job descriptions rated by level, sets of check-off items, and descriptions of alternative and ancillary titles.

New! #23: *A Sysadmin's Guide to Navigating the Business World,* by Mark Burgess and Carolyn Rowland

Achieving business alignment should be the desired end state both for system administrators and for the business for which they work. Getting there is a challenge for sysadmins, because they are often viewed as technicians, not strategic advisers. They need to think like business leaders, communicate their value and impact in a way that business leaders will understand, and establish a consistent value-driven model for IT within the organization. This is a book about how system administrators can better support the strategic goals of the workplace. Aligning IT processes and infrastructure with "business" is a topic that has been largely ignored in the past. The authors have set out to correct that, offering lots of examples, principles, and promises to help you navigate the waters of business.

Coming this month! #24: *Cloud Computing with VMware vCloud Director,* by John Arrasjid, Ben Lin, Raman Veeramraju, Steve Kaplan, Duncan Epping, and Michael Haines

This book augments book #21 and can be considered volume 2 on VMware cloud solutions. It covers the terms used in cloud computing, the financial considerations, the use cases, and the VMware technologies that are part of the VMware vCloud. As with book #21, this book provides insight into the tools you need. In some areas, such as vShield, additional depth is provided to help understand implementation details and use cases.

The full list of Short Topic Books is available at http://www.sage.org/pubs/short_topics.html.

SAGE members, don't forget: Not only can you view all of the books online in PDF form, but during each year of your membership you may also choose any one book to be sent to you for free. In addition, you may order additional print books for the low member price of $12 per book. Nonmembers may order any book for only $20 per copy, or join SAGE today and save!

—Anne Dickison

CONFERENCES

Conference Reports

In this issue:

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

Vancouver, BC, Canada
October 4–6, 2010

Kernels: Past, Present, and Future

Summarized by Brendan Cully (brendan@cs.ubc.ca)

An Analysis of Linux Scalability to Many Cores

Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich, MIT CSAIL

At the previous OSDI, Silas Boyd-Wickizer presented Corey, a new many-core operating system motivated by the premise that traditional OS design, as represented by Linux, faced fundamental limitations to scalability. This talk, also given by Boyd-Wickizer, refutes that premise. It does so by selecting several real applications with good parallel implementations that demonstrate kernel-based scalability problems and exploring the root causes of the bottlenecks.

Boyd-Wickizer et al. did their examination on an off-the-shelf 48-core Linux x86 server, running the applications on a RAM disk to avoid disk bottlenecks and stress the kernel. They found the bottlenecks, fixed them, and kept repeating this until either scalability was linear or the bottleneck was not in the kernel (i.e., it was in hardware or the application itself). They were able to fix most of the kernel bottlenecks with 16 patches and 3000 lines of code, demonstrating that Linux can scale well up to at least 48 cores. Most of the remainder of the talk consisted of walkthroughs of the specific bottlenecks found and the fixes applied. All of the fixes were straightforward applications of classic parallel techniques, typically to reduce cache coherence and memory bus contention by manually maintaining cache-local copies of shared data structures.

Eric Van Hensbergen from IBM Research asked whether the authors were attempting to integrate their changes into upstream Linux. Boyd-Wickizer responded that they hadn't even attempted it, because the engineering effort required was better spent on other research. Most other questions concerned the tension between this paper and the one on Corey from the previous year. Margo Seltzer (Harvard) and Eddie Kohler (UCLA) both asked variants of the question, "If shared state is bad for scalability, is it better to remove it at the points where it causes problems, or to avoid it generally, using it only at points where it is known not to cause problems?" Hank Levy (UW) put it most succinctly, getting a big laugh, when he asked, "Should we be designing new OS architectures or not? In other words, could you have written these papers in the opposite order?" Boyd-Wickizer carefully replied that Linux did not appear to have scalability problems up to 48 cores, but the possibility existed that it might after that.

Trust and Protection in the Illinois Browser OS

Shuo Tang, Haohui Mai, and Samuel T. King, University of Illinois at Urbana-Champaign

Shuo Tang presented the Illinois Browser OS (IBOS), an OS built from scratch to run Web browsers. It was motivated by two ideas: first, that vulnerabilities become more serious as they get deeper into the application stack, from individual Web application into the browser and finally the OS; and second, that it is easier to secure a system if the security layer has more semantic knowledge of what it is protecting.

Previous efforts like Google Chrome have used OS process isolation to provide some protection between compromised Web applications and the rest of the system, but IBOS takes the next logical step: it runs each browser in its own virtual machine with hardware-enforced isolation from the rest of a user's system. Its system call interfaces map to browser semantics much better than an ordinary operating system's would. For instance, instead of providing disk and network interfaces, it offers system calls for fetching URLs and storing cookies. This allows security policy to specify more precise invariants. By customizing the set of OS services to a Web browser, IBOS is also able to dramatically reduce the total TCB compared to an ordinary browser running on top of Linux. Tang concluded the presentation by revealing that his slides were served by a Web browser running on top of IBOS.

Shriram Rajagopalan of UBC asked if IBOS allowed the full set of services provided by modern browsers. Tang stated that IBOS used WebKit and supported most current browser features, including HTML 5. Etienne Le Sueur (NICTA) asked whether IBOS prevented cross-site scripting attacks and, if so, whether that broke beneficial uses such as Google Web Toolkit applications. Tang maintained that IBOS provided an enforcement mechanism, but using it was dependent on the user's security policy. The most fundamental question was from James Mickens (MSR), who noted that more and more Web browser research seemed to be retasking older core OS mechanisms, and wondered whether Tang believed that there was anything fundamentally different about Web browsers that made this reuse interesting. Tang simply replied that since Web browsers were running more general-purpose applications, we should be borrowing more operating system techniques.

FlexSC: Flexible System Call Scheduling with Exception-Less System Calls

Livio Soares and Michael Stumm, University of Toronto

System calls are expensive, both because of the time required to switch into kernel mode and because of cache contention while bouncing back and forth between user and kernel code. The traditional interface to the kernel is synchronous system calls, which make it impossible to amortize these costs. Livio Soares proposed an alternative kernel interface, FlexSC, which makes system calls asynchronous. This allows calls to be batched and, as a side effect, makes it possible to run kernel code on different cores from the user code it services. He also proposed a wrapper for FlexSC, called FlexSC-Threads, that hides the new system call style behind a traditional synchronous, thread-based API to support legacy code without major changes.

Soares demonstrated the degree to which system calls can reduce application performance, by modifying an application that makes very few system calls (Xalan from the SpecCPU benchmark) so that it made a large number of null system calls. In theory, the time excluding kernel processing to complete the benchmark should be about the same, but in fact the null system calls could halve performance, due to cache and TLB evictions. His solution was to use shared memory to queue system call requests. This not only allows batching of system calls, but can avoid context switches entirely if a separate core is dedicated to kernel request processing. Comparisons between MySQL and Apache using FlexSC showed up to 115% improvement in the Apache benchmark on four cores.

Van Hensbergen again asked whether Soares would attempt to integrate his interface into Linux. He received the same response given by Boyd-Wickizer at the first talk: it would be a lot of work and Soares would prefer to spend the energy on other things. Many others asked detailed questions about overhead measurement. For example, Michael Vrable wondered if the authors had measured cache contention due to userspace rescheduling when threads blocked. Soares had not, and he believes that this could indeed be another source of lost performance. Sasha Federova (SFU) noted that for FlexSC-Threads to show benefits, you would require more threads than cores; she wondered whether this mode might cause degradation for some workloads. Soares agreed that the benefits would mostly be for server workloads and that scientific applications that were 100% CPU-bound wouldn't need this.

Inside the Data Center, 1

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

Finding a Needle in Haystack: Facebook's Photo Storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel, Facebook Inc.

Peter Vajgel explained that Haystack is a photo storage system that addresses Facebook's significant scalability needs. Since April 2009, Facebook's photo storage has grown from 15 billion to 64 billion images. The update rate has increased similarly, from 220 million uploads per week to one billion currently. Prior to Haystack, pictures were served from a Content Distribution Network (CDN) that caches photos from an NFS server. Eventually, Facebook's working set grew to become so large that the cache hit rate through the CDN's had declined to 80% and the system bottlenecked on metadata access. At this larger scale, each photo access required 10 IOPS, due to deep directory lookups and file fragmentation. Through caching and reducing the directory sizes, they reduced this overhead to 2.5 IOPS. Haystack was developed to further lower the overheads of accessing a file to a single I/O operation, while maintaining a simple system based on commodity hardware.

Logically, the system is built from Haystacks, which are append-only object stores, and Needles, which are the photos and their associated metadata. Haystacks are generally 100GB in size and are stored on 10TB XFS volumes in a RAID6 configuration. The Haystack photo server builds a compact index of these images. The indexes for 10TB of images can be stored in 5GB, making the index less than 2% of the size of inodes. The system also supports load balancing and caching, and it operates over HTTP. In closing, Vajgel reiterated the goal of serving each request in a single operation and provided some hints at future work. The team plans to consider other RAID configurations and to investigate the use of flash-based storage.

Jason Flinn from the University of Michigan asked if prefetching was possible. Vajgel replied that users often select individual pictures from a page of thumbnails organized by some tag. This looks entirely random to Haystack. Christopher Colohan from Google asked if social networking could provide some clues to what photos would be accessed. Vajgel said that this might be possible, but that it would be best done at the application level, well above Haystack. Vipul Mathur from Network Appliance asked if migrating between Haystacks would be useful for load balancing, but Vajgel thought that RAID6's poor support for mixed workloads would limit such an approach, and he noted that the existing load balancer seemed sufficient.

Availability in Globally Distributed Storage Systems

Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan, Google, Inc.

Murray Stokely began by posing questions of data availability, the causes of unavailability, and methods for tuning to the observed environment. To find answers, the authors gathered a year's data from tens of Google's clusters. They found that most unavailability periods are transient with a median length of 15 minutes. However, unavailability durations differ significantly by cause. The median time to restart a node's storage software is 1 minute, but it is 10 minutes for a planned reboot, and a few minutes longer for an unplanned reboot. Interestingly, 37% of failures observed were correlated in time to other failures. Larger-scale outages of this type regularly share common causes: for example, software bugs, shared hardware components, or supply chain issues.

Florentina Popovici then explained how these results informed the development of analytical models to study the availability of large distributed storage systems. This approach led to several insights. For small to medium bursts of failures and large encodings, when data is striped across many nodes, rack-aware placement can increase the MTTF by a factor of 3. Using a Markov model for analysis, another insight was that improving data availability below the node layer did little to help, although reducing node failure rates has a significant effect on availability. She concluded by stressing that a highly available distributed storage system requires exploring beyond disk failure rates, and that correlated failures are important in understanding large-scale storage availability.

Garth Gibson from CMU commented on graphs that appeared to capture unavailability due to rolling updates. He saw potential in making active decisions about how these events occur. Stokely agreed, pointing to the possibility of making additional replicas before a planned reboot. Mehul Shah from HP Labs asked if Reed-Solomon encoding doesn't also offer cost savings and asked why that wasn't used initially. Popovici explained that there may be performance impacts, such as parallelism in batch-oriented processing, recovery bandwidth concerns, and the need for more servers and CPUs. Philip Howard from PSU asked how expected increases in CPU core density would affect correlated failure rates. Stokely confirmed that this is a motivating question for the work and that the cost per core per watt per year is the determining factor. He also noted that the same questions can be analyzed about the number of disks per server and other parameters with this model.

Nectar: Automatic Management of Data and Computation in Datacenters

Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang, Microsoft Research Silicon Valley

Li Zhuang observed that a large fraction (20–40%) of computation in their datacenters is redundant and that most storage is occupied by datasets that are never accessed. The two inefficiencies share a common problem—intermediate data created as part of larger processing operations. Nectar attempts to increase datacenter efficiency by tracking and reusing these intermediate results, while also allowing them to be deleted safely when the need arises.

Nectar is designed for datacenter applications written in the dataflow language LINQ, which may involve multiple stages of computation being stored persistently. As these intermediate files are created, Nectar records the task that created them and its associated parameters. Since the environment is deterministic, future attempts to run the same task are replaced with much more efficient accesses of the cached data. Under storage pressure, a mark and sweep garbage collector identifies intermediate files that can be deleted and regenerated on demand. Since usage information is tracked, datasets that are old and infrequently accessed can be prioritized for deletion. Zhuang concluded by describing Nectar as a promising approach that automates and unifies the management of data and computation in the datacenter.

Zhuang fielded several questions about Nectar's design and the DryadLINQ environment. In addition, David Schultz of MIT observed that storage capacity sizes relative to throughput seem to suggest that memory and I/O bandwidth are more important than capacity. Zhuang stressed that over time, it's important that capacity and computation aren't wasted as well. Micah Brodsky of MIT asked if the higher-level semantics and policies we have been adding to these dataflow environments are leading us towards reinvention of distributed databases, or if we are inventing something new. Zhuang replied that while they learned from distributed databases, there are significant differences that influence design. Cited examples included the scale of operation and the design of materialized views.

Security Technologies

Summarized by Nathan Taylor (tnathan@cs.ubc.ca)

Intrusion Recovery Using Selective Re-execution

Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek, MIT CSAIL

Software bugs, misconfigured security policies, and weak passwords all routinely lead to compromised systems. When a break-in is discovered, system administrators are caught between simply rolling back to the latest backup and losing users' recent work, and spending an inordinate amount of time on a manual recovery, with the chance of missing a change caused by the adversary perpetually looming. To this end, the authors present a tool for system recovery, dubbed Retro, that can tease apart exploit-related changes from benign changes made by legitimate users.

Zeldovich began by contrasting their work to two straw-men arguments: the first involved a naive taint tracking scheme, where all tainted files are restored from an earlier backup. This exhibits the usual explosion problem common to taint tracking. The second, in-VM replay, is too slow, and external state changes, such as crypto keys, mean that benign replay may go off the rails. Retro's approach involves selective execution, wherein execution data such as function calls and syscall arguments are logged in a so-called action history graph. This graph is used to roll time back to the compromise of the system and replay all non-attack activity, while skipping replay of operations the system has concluded were not affected by the attacker. If the attack involved communication with the outside world, this activity cannot be resolved by the system; the recovery will pause and prompt the sysadmin for guidance.

Ten real-world and synthetic challenge attacks were used to evaluate the tool; six were able to roll back automatically, while the remaining four needed some form of user input to handle legitimate network traffic or skip over the attacker's SSH login attempts. The amount of time needed to restore a system was found to be a function of the number of objects to track, and not the length of the log. The runtime overhead varied depending on workload: a HotCRP server averaged 4GB/day with a 35% slowdown, while a continuous kernel compilation averaged 150GB/day with a 127% slowdown. However, pushing the logging to a spare core significantly improved performance.

The presentation drew a lively Q&A session. Josh Triplett of Portland State asked about recording fewer user actions to pay a lower CPU/storage overhead. Zeldovich replied that this is possible, at the expense of reconstructing more of the dependency graph at repair time. Margo Seltzer of Harvard noted the similarities to providence and asked about what constitutes "acceptable" nondeterministic execution at repair time. Zeldovich argued that anything that the execution could have originally created was fair game. An audience member wondered about trying to explode the repair space by creating many false positives, to which Zeldovich pointed out that activity that taints everything could itself be a symptom of an attack. Lorenzo Cavallaro (Vrije Universiteit) asked about porting to Windows. Zeldovich replied that the ideas were applicable to Windows, but several more managers would have to be written. They use SELinux to protect Retro, and something would be needed to replace that as well.

Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications

Adam Chlipala, Impredicative LLC

Enforcing sensible security policies is a challenge if a vulnerability relies on semantic knowledge of the application, such as a particularly crafted URL or a magic value within a cookie. Developers need to find attack vectors and audit them, but one can always be blindsided by a vector that one had not considered. Chlipala argued that it's far better to isolate a resource and have a policy stating how said resource may be accessed. He went on to present UrFlow, a SQL-like policy language that exploits a common database query–like idiom to define what parts of a program have access to a sensitive resource. Chlipala argued that UrFlow has the best of both static and dynamic checking: no source-level annotations are required, and all codepaths can be checked without incurring a runtime overhead.

UrFlow translates its policies into simple first-order logic for modeling what the system should know to be true when matching against policies; for instance, after reading a cookie, UrFlow knows that the system will know a password. Program paths are checked against its policies statically. Each execution path is executed symbolically, and as each progresses, the new logical clauses it has generated are sent to the theorem prover.

UrFlow was tested against applications ranging from ~100 to ~500 lines of code with ~50 policies each. Static checks are usually performed in well under a second. There were some significant outliers in runtime, but Chlipala hoped that applying traditional optimization tricks would smooth that out. Chlipala finished his talk by highlighting a disconnect of PLT researchers and programming practitioners: it's hard to sway developers away from imperative languages, but most Webapps use SQL, a perfectly reasonable declarative language anyway!

Most of the Q&A session was concerned with writing policies in practice. James Pendergrass (APL) asked if there is a way to ensure that rules are consistent with each other; Chlipala admitted that this is a tricky problem. Along the same lines, Pendergrass asked whether UrFlow is amenable to growing policies over time. Chlipala replied that it would be, since you won't be able to accidentally undo any previous policy. Joshua Triplett asked whether the theorem prover could be improved to provide hints about how the code could be changed to guarantee security. Chlipala admitted that it's tough to determine which fact in the rule base caused the prover to error out, but that there might be ways to heuristically prune it.

Accountable Virtual Machines

Andreas Haeberlen, University of Pennsylvania; Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel, Max Planck Institute for Software Systems (MPI-SWS)

Haeberlen presented a running scenario of a multiplayer game where players would like to guarantee that no player is using a cheat, and in this way they addressed the more serious problem of knowing whether programs on a third-party machine are executing as intended. A user should be able to detect when a remote machine is "faulty" and provide evidence of the fault to a third party without knowing the internals of their software. The authors' solution to this uses so-called accountable virtual machines. AVMs use a log of network traffic, signed with other players' keys, as a secure certificate of correct execution. Players may replay each others' logs in their own VM. If a player is running the game with a cheat, such as the "infinite ammo" hack, control flow will diverge from the log when the unhacked reference VM's game runs out of bullets. In so doing, strong accountability for arbitrary binaries is offered, without needing to trust other players or the AVM itself.

MPI-SWS built a prototype AVM atop VMware Workstation and tested it on Counterstrike 1.6. Performance is reasonable; on native hardware, the game averages 158FPS, and the authors observed a 13% slowdown with the AVM, mainly due to the overhead of the execution logging. The log size grows at 8MB/min, but because each action is replayed, replaying takes about as long as the game itself. A sample of 26 real-world CS cheats were all detected. But Haeberlen wonders whether cheaters could adapt their cheats. Re-engineering hacks to evade AVM detection, cheaters would have to fill in missing interrupts with the right branch counter value, which, although very hard, is theoretically possible.

Neal Walfield (Johns Hopkins) wondered whether the AVM itself could be hacked to fake the log. Haeberlen responded that the authenticators sign the AVMs and invited Walfield to discuss this offline. The subsequent questions involved seeking clarification on what class of cheats the AVM can detect. Haeberlen reiterated that the AVM is guaranteed to find the attack unless the cheat is "plausible" input; for instance, a better-aim cheat might not be detected. On the other hand, because there's no input that could drive the program into such a state, the "infinite ammo" cheat is detectable. Brad Chen was skeptical of the applicability of the AVM to domains outside gaming, since the act of replaying requires sharing a great deal of information. Haeberlen admitted that the auditor has to be able to see everything, but he pointed out that in a cloud computing scenario a cloud owner ought to be able to audit his own VM.

Concurrency Bugs

Summarized by Kaushik Veeraraghavan (kaushikv@umich.edu)

Bypassing Races in Live Applications with Execution Filters

Jingyue Wu, Heming Cui, and Junfeng Yang, Columbia University

Jingyue Wu described a use-after-free data race in Mozilla's JavaScript engine with a simple root cause that nonetheless took developers a month to fix. Wu pointed out that a data race bug in a critical Web service that remains unresolved for weeks or months could be exploited by an attacker. As a solution, Wu proposed LOOM, a tool that can work around known race conditions in live programs before the developer releases a fix. LOOM accomplishes this by using execution filters, which are simple declarative statements that explicitly synchronize regions of application code.

LOOM combines static preparation with live updates. Specifically, a LOOM compiler plugin bundles the LOOM update engine into an application binary. A user wishing to install an execution filter on a live application invokes the LOOM controller. The LOOM controller is responsible for translating the filter's declarative statements into specific operations (e.g., mutual exclusion is translated into lock/unlock, ordering requirements are translated into semaphores, etc.). The controller passes the translated application code to the LOOM update engine responsible for patching the live application. To ensure that filters can be safely installed, Wu described an algorithm termed evacuation that uses static analysis to identify dangerous regions of code and ensures that (1) threads wishing to enter a dangerous region are paused at a safe location before the region, and (2) threads already executing in the dangerous region exit the region before the update is installed.

Wu described LOOM's evaluation on nine data races in six applications—in all cases, LOOM applied the live update in a timely manner and bypassed the race. On average, LOOM adds less than 5% performance overhead. Wu also described an alternate implementation using the Pin dynamic instrumentation tool, which introduced a 10x overhead. Wu demonstrated that LOOM is also scalable—MySQL running with 32 threads experiences only a 12% performance overhead.

Richard Draves (Microsoft Research) was curious if the evacuation algorithm makes the application susceptible to deadlock. Wu responded that while an application could deadlock, they did not come across any in their evaluation of LOOM. Further, LOOM implements a mechanism to uninstall the fix if a deadlock manifests. The next questioner asked if Wu could use information collected at runtime to fix the bug in the source code. Wu responded that users can examine the application source code and can also write execution filters targeted at specific file names/line numbers to fix the bug.

Effective Data-Race Detection for the Kernel

John Erickson, Madanlal Musuvathi, Sebastian Burckhardt, and Kirk Olynyk, Microsoft Research

John Erickson, a test lead on Microsoft's Windows 7 team, described the first data race in the Windows kernel he had discovered using DataCollider: a developer had assumed that certain bitwise operations were atomic without realizing that the operations were actually read/write byte instructions. A rare write-write interleaving would overwrite a callback flag resulting in a system hang. As this race was timing dependent, it was not caught prior to shipping Windows 7. This example illustrates that data races are hard to detect and debug.

The primary challenge in using existing happens-before and lockset algorithms to detect data races in the kernel is that both approaches need to be aware of all locking semantics. To address this challenge, Erickson proposed DataCollider, a data race detection tool for the kernel that is based on two insights: (1) instead of inferring a data race, cause a data race to actually manifest so there are no false positives; (2) the overhead of instrumenting all memory accesses can be significantly reduced by sampling a random subset of all memory accesses. DataCollider works as follows: a code or data breakpoint is set on a memory location being sampled. If a thread accesses the sampled memory location, it is blocked for a short time window. If a second thread tries to access the same memory location within that window without grabbing the appropriate lock, the breakpoint causes the second thread to block too. By pausing both threads at the instance the race occurs, DataCollider has access to both thread stacks, the full register and global state of which can be logged to report the race to the developer.

DataCollider allows users to control execution overhead by explicitly selecting the sampling rate—a sampling rate of 0% allows the code to execute at native speed. An evaluation of DataCollider on Windows 7 revealed 113 races on booting to the desktop. While the majority are benign, 25 are confirmed bugs in Windows 7 that are fixed or will be fixed soon.

George Candea (EPFL) wondered about the difficulty in categorizing races in real products as benign or malign. Erickson responded that much of the bucketization was manual and required source code analysis. Rik Farrow questioned whether breakpoints were truly randomly selected or whether they were applied at carefully selected locations. Erickson responded that all the memory accesses were enumerated and breakpoints were set on a random sampling without any knowledge of the actual memory accesses. The only tuning factor is that users can specify how many breakpoints they wish to execute per second. Michael Brodsky (MIT) asked how dependent DataCollider is on having debug symbols, since third-party driver manufacturers might not provide these with their binaries. Erickson responded that DataCollider requires debugging symbols, so breakpoints are applied to actual memory locations. Peter Druschel (MPI-SWS) wondered how the sampling time and detection rate were related to the total number of races. Erickson answered that while they do want to ensure that DataCollider is detecting a uniform sampling of races and not just the easiest 20, they cannot evaluate if this is the case, since no other data race detectors operate within the kernel.

Ad Hoc Synchronization Considered Harmful

Weiwei Xiong, University of Illinois at Urbana-Champaign; Soyeon Park, Jiaqi Zhang, and Yuanyuan Zhou, University of California, San Diego; Zhiqiang Ma, Intel

Concurrent programs increasingly rely on synchronization to guarantee correct execution. While many popular applications such as MySQL use standard synchronization libraries (e.g., pthreads), others such as LDAP rely on ad hoc synchronization, which is often harder to understand and debug.

The primary contribution of Xiong's work is quantitative evidence that ad hoc synchronization should be considered harmful. Xiong and his co-authors spent three months documenting every instance of ad hoc synchronization in 12 concurrent programs, including MySQL, Apache, and Mozilla. While all of them implemented some form of ad hoc synchronization, Xiong found that MySQL employed the most—83 instances. Next, Xiong analyzed bug patches uploaded to the Bugzilla database and discovered that almost half of all ad hoc synchronizations resulted in a concurrency bug. Unfortunately, existing static analysis tools that detect deadlocks, data races, and other concurrency bugs are rendered useless for multi-threaded programs that use ad hoc synchronization, as these tools do not understand the custom-locking semantics employed in ad hoc synchronization.

The second contribution of Xiong's work is SyncFinder, a tool that automatically detects and annotates ad hoc synchronization. The insight in SyncFinder is that every ad hoc synchronization executes in a loop with an exit condition that is controlled by a dependent variable. After applying several pruning algorithms, SyncFinder identifies shared dependent variables that control ad hoc sync loops and annotates all instances where the variables are read or written to. An evaluation of SyncFinder revealed that, on average, it detects 96% of all ad hoc synchronizations with a 6% false-positive rate.

The first questioner wondered how many bugs normal synchronization introduced in comparison to ad hoc synchronization. Xiong responded that the percentage of bugs is much higher in ad hoc synchronization. Bryan Ford (Yale) wondered what the current synchronization primitives lack that caused developers to turn to ad hoc synchronization. Xiong responded that, from an analysis of program comments, it appeared that developers just wanted a flexible and simple synchronization primitive. Ford wondered what such a synchronization primitive should offer, to which Xiong responded that such a primitive already existed: conditional waits. Dan Peek (Facebook) suspected that programmers implementing ad hoc synchronization wished to avoid a synchronous write on the system bus which hurts performance, and wondered whether such synchronization was always harmful. Xiong responded that while not always harmful, ad hoc synchronization might not scale with the underlying architecture, and also degenerates code maintainability. Dan Peek followed up by asking if lock-free data structures should also be considered harmful. Rather than commenting on lock-free data structures, Xiong offered that developers should use well-known existing synchronization primitives. Chris Hawblitzel (Microsoft Research) wondered whether developers declaring synchronization variables as volatile would help SyncFinder. Xiong responded that if C/C++ adopted volatile as a standard it would help SyncFinder. Josh Triplett (Portland State University) wondered how fast SyncFinder runs. Xiong responded that the largest codebase SyncFinder was executed on was MySQL, which has over 1 million lines of code and took about 2.5 hours. On the other hand, OpenLDAP, much smaller but containing a lot of ad hoc synchronization with significant data and control dependency, also took 2.5 hours.

Monster Poster Session

First set of posters summarized by John McCullough (jmccullo@cs.ucsd.edu)

VSSIM: Virtual SSD Simulator

Joohyun Kim, Haesung Kim, Seongjin Lee, and Youjip Won, Hanyang University, Seoul, Korea

VSSIM allows researchers to explore different configurations of SSD hardware. The simulator emulates both page and hybrid translation layers. The researchers have validated the expected behavior of higher block usage overhead of the hybrid translation layer as well as the effects of TRIM. Contact: james@ece.hanyang.ac.kr.

Energy Consumption Behavior of Modern SSD and Its Architectural Implication

Balgeun Yoo and Youjip Won, Hanyang University, Seoul, Korea

Yoo and Won evaluate the energy use of an X-25M SSD. Using writes of varying size, they find that energy use is proportional to the number of channels that are activated as well as the number of ways—chips within a single package—that are active. Changes in power draw are bounded between consumption for 8kB writes and 512kB writes. Contact: starthunter@ece.hanyang.ac.kr.

ABACUS: A Configurable Profiling Engine for Multicore Processors

Sergey Blagodurov, Eric Matthews, Sergey Zhuravlev, Lesley Shannon, and Alexandra Fedorova, Simon Fraser University

There are a number of hardware performance metrics that are useful for multicore scheduling that are not commonly available. ABACUS is a system that implements counters such as instruction mix and pipelines stalls with no processor time overhead on an FPGA-based processor core. Contact: sba70@cs.sfu.ca.

DoublePlay: Parallelizing Sequential Logging and Replay

Kaushik Veeraraghavan, Dongyoon Lee, Benjamin Wester, Peter M. Chen, Jason Flinn, and Satish Narayanasamy, University of Michigan

DoublePlay employs multiple executions to detect data races. The system uses a novel technique of dividing a parallel execution into epochs, recording the high-level thread execution schedule along with memory checkpoints. By re-executing the parallel schedule in a serial manner, the memory states can be compared to detect data races. Contact: kaushikv@umich.edu.

Using Program Analysis to Diagnose Configuration Problems

Ariel Rabkin and Randy Katz, University of California, Berkeley

Java has a wide variety of configuration parameters. Deciphering which parameter might be responsible for a stack trace can be challenging. This work uses static analysis to tie error points in Java to the relevant configuration parameters. Contact: asrabkin@eecs.berkeley.edu.

DCR: Replay Debugging for the Data Center

Gautam Altekar and Ion Stoica, University of California, Berkeley

Deterministic replay is very helpful in debugging distributed systems but can have high overhead. Most distributed system bugs arise from control plane errors. By logging only the control traffic, the total volume can be reduced by 99%. Thus, the control traffic can be used to replay the overall behavior and semantically equivalent data can be constructed using STP techniques. Contact: galtekar@cs.berkeley.edu.

Reconfigurable Virtual Platform for Real Time Kernel

Dilip K. Prasad, Nanyang Technological University, Singapore; Krishna Prasath, Coventry University, UK

Evaluating real-time applications across multiple platforms can be very challenging. Prasad and Prasath have created a platform that is easily reconfigurable for different operating systems and that cleanly integrates into an IDE. Contact: dilipprasad@pmail.ntu.edu.sg.

ErdOS: An Energy-Aware Social Operating System for Mobile Handsets

Narseo Vallina-Rodriguez and Jon Crowcroft, University of Cambridge

ErdOS leverages social interactions to improve energy use in mobile devices. For instance, if a power-hungry application is redundant to a localized area, a single device can take the measurement and gossip the results to those allowed by social graph access controls. Contact: nv240@cam.ac.uk; Web: http://www.cl.cam.ac.uk/~nv240/erdos.html.

Leviathan: Taming the #ifdef Beast in Linux et al.

Wanja Hofer, Christoph Elsner, Frank Blendinger, Wolfgang Schrîder-Preikschat, and Daniel Lohmann, Friedrich-Alexander University Erlangen-Nuremberg

Understanding ifdef-laden code can be very challenging in many situations. The authors have created a pre-processing FUSE plug-in that allows developers to operate on the pre-processed code. Edits to either the view or the code are automatically transferred back to the appropriate part of the original. Contact: hofer@cs.fau.de.

Joan: Shepherd Application Privacy with Virtualized Special Purpose Memory

Mingyuan Xia, Miao Yu, Zhengwei Qi, and Haibing Guan, Shanghai Jiao Tong University

Multiple network applications executing on the same machine are not entirely safe from one another. The authors explore a technique of per-application hypervisor protected memory that can be selectively shared in either modifiable or read-only form.

Configuration Bugs in Linux: The 10000 Feature Challenge

Reinhard Tartler, Julio Sincero, Wolfgang Schrîder-Preikschat, and Daniel Lohmann, Friedrich-Alexander University Erlangen-Nuremberg

The configuration specified in the Linux config file may not be correctly embodied by the relevant ifdefs in the code. Misspellings and contradictions from the combination of ifdef conjunctions and feature dependencies can lead to dead or misbehaving code. The authors have used code analysis to identify many points of contradictory and dead code, culminating in 123 patches with 64 acknowledged by kernel authors. Contact: tartler@informatik.uni-erlangen.de.

Backup Metadata as Data: DPC-tolerance to Commodity File System

Young Jin Yu, Dong In Shin, Hyeong Seog Kim, Hyeonsang Eom, and Heon Young Yeom, Seoul National University

File system metadata is critical to the integrity of the actual data stored on the file system, but it is ignored by typical backup techniques. The authors extract the filesystem-level pointers for use in a filesystem recovery mechanism that is faster than traditional file system scans.

Second set of posters summarized by Robert Soule (soule@cs.nyu.edu)

Using Mobile Phones to Set Up Secure VNC Sessions on Kiosks

Wooram Park, Sejin Park, Baegjae Sung, and Chanik Park, Pohang University of Science and Technology

Many people use publicly available computers called kiosks to access remote desktops. In order to prevent the leakage of private data, this work proposes using a mobile phone to establish a VNC session with the remote desktop. The kiosk attests its identity to the remote desktop, using keys issued by a remote key server, which are stored on the mobile phone.

Aggressive VM Consolidation with Post-Copy-based Live Migration

Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi, National Institute of Advanced Industrial Science and Technology

There are two techniques for migrating live virtual machines, pre-copy and post-copy. This work implements the post-copy live virtual machines migration of KVM, which performs significantly faster than previous pre-copy implementations.

Mnemosyne: Lightweight Persistent Memory

Haris Volos and Michael Swift, University of Wisconsin—Madison; Andres Jaan Tack, Skype Limited

Storage class memory provides low-latency, persistent storage. This work seeks to provide programmers with direct access to storage class memory. Mnemosyne provides two abstractions: one for allocating memory, and a second, called durable memory transactions, for atomically modifying persistently stored data structures.

Beacon: Guiding Data Placement with Application Knowledge in Multi-Tiered Enterprise Storage System

Hyojun Kim, Georgia Institute of Technology; Sangeetha Seshadri, IBM Almaden Research Center; Yi Xue, IBM Toronto; Lawrence Chiu, IBM Almaden Research Center; Umakishore Ramachandran, Georgia Institute of Technology

Enterprise storage systems contain both cheap but slow HDD and fast but expensive SSD. This work tackles the problem of deciding what data is stored in which storage system. Current approaches use a reactive approach, which places data after monitoring usage over time. In contrast, this work modifies applications to provide hints about their spatial (what files), temporal (duration), and priority requirements, allowing for predictive placement of data.

Guest Transparent Dynamic Memory Balancing in Virtual Machines

Changwoo Min, Inhyuk Kim, Taehyoung Kim, and Young Ik Eom, Sungkyunkwan University

Multiple guest operating systems on a single host machine compete for memory. This work seeks to solve those conflicts by estimating the memory requirements of each guest operating system and pre-allocating that memory.

Parallel Operating System Services in fos

David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Harshad Kasture, Lamia Youseff, Jason Miller, and Anant Agarwal, Massachusetts Institute of Technology

fos is an operating system for multicore systems that treats operating system services like distributed Internet servers. Each service is implemented as a set of spatially separated server processes called a fleet. The servers' processes in a fleet collaborate to provide both high-level services and as low-level services such as page allocation, scheduling, and memory management.

S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems

Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

S2E provides an automated path explorer and modular path analyzers and is used for various tasks, including performance profiling, reverse engineering software, and bug finding in both kernel-mode and user-mode binaries. S2E scales to larger real systems, such as a full Windows stack, better than prior work.

Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory

Shivaram Venkataraman, University of Illinois; Niraj Tolia, HP Labs; Roy H. Campbell, University of Illinois

Hardware manufacturers have developed new nonvolatile memory devices. This work explores what are the best data structures for storing data on these new devices. Rather than using the traditional write-ahead logging approach, this work keeps multiple copies of the data structures (for example, a B-tree) and atomically moves between subsequent versions.

Tracking and Exploiting Renewable Energy in Grid-Tied Datacenters

Nan Deng and Christopher Stewart, Ohio State University

Datacenters use a device called a grid-tie to combine energy from renewable sources and energy supplied by a grid. When placing these grid ties in the datacenter, engineers must make a choice between placing a small number of high-capacity grid-ties (which themselves consume energy), or a large number of low-capacity grid-ties. This work seeks to find the optimal placement strategy through simulation.

Making Tracks with Malware: Control Flow Visualization and Analysis

Jennifer Baldwin, University of Victoria

This work presents a new tool for visualizing the control flow of assembly programs. The tool provides better visualizations than existing systems, and demonstrates the clarity of its presentation by showing the control flow of a malware application.

dBug: Systematic Evaluation of Distributed Systems

Randy Bryan, Garth Gibson, and Jiri Simsa, Carnegie Mellon University

dBug is a tool for finding bugs in distributed systems. It introduces a thin interposition layer between the application and the operating system, which hijacks system calls for synchronization and shared memory. An arbiter scheduler then systematically explores different execution paths to discover bugs.

Dynamic Forwarding Table Management for High-speed GPU-based Software Routers

Joongi Kim, Keon Jang, Sangjin Han, KyoungSoo Park, and Sue Moon, KAIST

In prior work, the author implemented a software router in a GPU. After using the system in practice, it became clear that there was a need for dynamic updates to the forwarding tables. This work explores how to support these dynamic table updates in the presence of bursty network traffic.

Preventing Memory-Allocation Failures Through Anticipatory Memory Allocation

Swaminathan Sundararaman, Yupu Zhang, Sriram Subramanian, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Recovery code for memory allocation failures in file systems is buggy. This work seeks to avoid executing that recovery code at all. A utility performs static analysis on the file system code to determine how much memory the software will request. The system then pre-allocates the memory and proxies all subsequent requests for memory, servicing the requests from the pre-allocated pool.

Third set of posters summarized by Michael Roitzsch (mroi@os.inf.tu-dresden.de)

Coerced Cache Eviction: Dealing with Misbehaving Disks through Discreet-Mode Journalling

Abhishek Rajimwale, Vijay Chidambaram, Deepak Ramamurthi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

What if the controller of a magnetic hard drive or an SSD does not fully honor cache flush requests for its internal cache? If you run a workload that tries to enforce consistency requirements by explicitly flushing the cache, a system crash may lead to unexpected inconsistencies. The poster authors suggest forcefully evicting everything from the cache by sending a cache flush workload to the disk. They address the problem of balancing the resulting overhead with the probability of achieving a full flush.

Testing Device Drivers without Hardware

Matthew J. Renzelmann, Asim Kadav, and Michael M. Swift, University of Wisconsin—Madison

The authors want to simplify driver maintenance for kernel developers without access to the device in question. This situation occurs when making changes to kernel interfaces that lead to subsequent changes in other drivers. The proposed solution consists of a symbolic execution engine for the driver code, combined with a symbolic hardware representation. This symbolic hardware is generic per device-class. KLEE is used for the symbolic execution. To avoid symbolic execution of startup code like kernel boot and device initialization, the system uses device traces to fast-forward to the interesting parts of the execution.

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Justin Ma, Michael J. Franklin, Scott Shenker, and Ion Stoica, University of California, Berkeley

A new programming model for applications processing large amounts of data in a distributed environment was presented. Whereas MapReduce and Dryad are well fitted for acyclic data-flow algorithms, this model focuses on algorithms that reuse a working set across operations. One target application is interactive data mining. The programming paradigm is constructed around the concept of a Resilient Distributed Dataset (RDD). An RDD provides an abstraction for objects that handles distribution and persistence. To achieve the fault-tolerance of MapReduce, an RDD can be rebuilt if it is lost due to failure.

The CONSCIOUS Virtual Machine Model: Transparently Exploiting Probability Processors

Jonathan Appavoo, Dan Schatzberg, and Mark Reynolds, Boston University; Amos Waterland, Harvard University

When collecting instruction pointer traces of a virtual machine, the authors noticed recurring patterns in the trace. They proposed the idea of collecting a repository of those patterns and the results of the operations they represent. If such a known pattern is recognized in the trace of a running VM, the execution can fast-forward to the end of the pattern and apply the canned result. This can save time and energy by not repeating recurring operations. As a side effect, the VM traces can be converted to audio files so you can actually hear the patterns!

Tracking and Exploiting Renewable Energy in Grid-Tied Datacenters

Nan Deng and Christopher Stewart, Ohio State University

Datacenters can be equipped with on-site renewable energy generators such as wind turbines or solar collectors. However, the energy provided by those does not suffice to run the datacenter, so energy from the traditional power grid needs to be mixed in. This task is performed by a grid-tie component which itself consumes energy and can fail, so the number and placement of these components poses an interesting research challenge.

Dynamic Runtime Optimizations inside BT-based VMMs

Mehul Chadha and Sorav Bansal, Indian Institute of Technology Delhi

Traditional runtime optimization of an application is done by tracing and just-in-time compilation of the application. The presenter, Sorav Bansal (sbansal@cse.iitd.ernet.in), proposes extending this idea to the entire system. Combining a virtual machine monitor with binary translation and a just-in-time compiler, runtime optimizations can be performed throughout the system, even crossing protection boundaries. The system can be enhanced with peephole and trace-based optimizations in future work.

Can You Keep a Secret?

David Cock, NICTA and University of New South Wales

Covert timing channels are an increasingly relevant problem for crypto systems, as the recent padding oracle attack on ASP.NET-based Web sites has shown. The poster presenter, David Cock (david.cock@nicta.com.au) proposed using a real-time scheduler on the seL4 microkernel to control component response times. The cryptographic code of an application, like the OpenSSL library, would be separated into an isolated component with individual scheduling. Hooking into seL4's communication endpoints, the scheduler can delay responses of a component and thus decouple the component's actual response time and the behavior observable from outside. Reducing the variation on observed response time, the scheduler can reduce the bandwidth of the timing channel.

The Private Peer Sampling Service: The Ground for Your Secret Society

Valerio Schiavoni, Etienne Rivière, and Pascal Felber, University of Neuchâtel, Switzerland

The subject of this poster is group membership within a larger peer-to-peer overlay network. The network is cryptographically protected by the exchange of public keys. If you want to form groups within this network, you want to keep the group membership information private. The work presented on the poster employs onion routing and attacks the problems of firewalls, NATs, and the resulting challenges regarding the visibility of nodes.

Gang scheduling isn't worth it . . . yet.

Simon Peter, Andrew Baumann, and Timothy Roscoe, ETH Zurich

Andrew Baumann from the Barrelfish group (http://www.barrelfish.org/) presented this poster. Looking at opportunities for gang scheduling in the Barrelfish multikernel, the team studied the merits of gang scheduling for different workloads. Today's commodity operating systems typically do not employ gang scheduling. But this usually does not hurt, because workloads on these systems are dynamic, bursty, and interactive. Multiple parallel applications with fine-grained synchronization would be needed for the lack of gang scheduling to become a problem. One situation where it would help are stop-the-world garbage collectors.

Multi-pipes

Eric Van Hensbergen, IBM Research; Noah Evans, Alcatel Lucent Bell Labs; Pravin Shinde, ETH Zurich

This poster presented one aspect of a larger project on operating systems and services for high performance computing that scale to millions of cores. Specifically, it presented a data flow abstraction developed for BlueGene supercomputers. Multi-pipes are a generalization and extension of traditional UNIX pipes. Multiple readers and writers are handled deterministically, including fan-in and fan-out scenarios. Operations like broadcasts or reductions on the data traveling through the multi-pipe are supported. As a bonus, shell integration for multi-pipes is available.

Scaling Middleboxes through Network-Wide Flow Partitioning

Norbert Egi, Lancaster University; Lucian Popa, University of California, Berkeley; Laurent Mathy, Lancaster University; Sylvia Ratnasamy, Intel Labs, Berkeley

Large network installations contain many routers and each one contains a general-purpose processor. Turning those routers into middleboxes lets them perform duties like policing traffic, caching, or traffic encryption. However, this functionality is not now in one central location like a traditional firewall, but distributed throughout the network. Traffic must be suitably partitioned to scale this distributed middlebox system.

STEAMEngine: Driving the Provisioning of MapReduce in the Cloud

Michael Cardosa, Piyush Narang, and Abhishek Chandra, University of Minnesota; Himabindu Pucha and Aameek Singh, IBM Research—Almaden

STEAMEngine is a runtime for assigning VMs to execute MapReduce workloads. Given a set of VMs distributed over multiple machines, pending MapReduce tasks are mapped to a subset of those VMs. Individual jobs are profiled at runtime to model their resource needs. The spatio-temporal assignment of MapReduce jobs to VMs can then be tuned to meet timeliness or energy goals.

KÁRMA: A Distributed Operating System for MicroUAV Swarms

Peter Bailis, Karthik Dantu, Bryan Kate, Jason Waterman, and Matt Welsh, Harvard University

Studies show the US bee population decreased by 30%. But agriculture needs bees to pollinate the crops. The resulting gap is closed with the Robobees: minimal robotic bees that include sensors and compute resources. The particular aspect presented in this poster is the development of a distributed system that exhibits swarm intelligence. Combining a suitable swarm programming model, the necessary hardware and artificial intelligence, Robobees can one day actually fly and perform the pollination tasks of a real bee.

Fourth set of posters summarized by Justin Seyster (jseyster@cs.stonybrook.edu)

Diagnosing Performance Changes by Comparing System Behaviours

Raja R. Sambasivan, Carnegie Mellon University; Alice X. Zheng, Microsoft Research; Elie Krevat, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R. Ganger, Carnegie Mellon University

Sambasivan presented Spectroscope, a project that uses end-to-end traces to diagnose distributed system performance changes. Each trace represents the entire path of a request through the system. Comparing traces from before and after a performance change can help find the cause of the change. A comparison can find structural changes, such as a request being routed to a remote datacenter, and response-time changes in individual components. Both kinds of change are ranked by their contribution to the overall performance change so that the developer can localize the source of the problem. The authors used Spectroscope to diagnose several real-world performance bottlenecks in Ursa Minor, a prototype distributed storage system.

The Anzere Personal Storage System

Oriana Riva, Qin Yin, Dejan Juric, Ercan Ucan, Robert Grandl, and Timothy Roscoe, ETH Zurich

Oriana Riva described Anzere, which manages synchronization of media, contacts, and other personal data across multiple personal devices and virtual cloud resources. Anzere emphasizes the expressiveness of replication policies, which are specified with a logic constraint language. For example, the user can require that one-day-old music is accessible to a cell phone with no more than 100ms delay. As another example, the user can also set a limit on cloud storage usage in terms of a maximum monthly fee. Although potentially complex, these policies remain tractable.

Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis

Yagz Onat Yazr and Chris Matthews, University of Victoria; Roozbeh Farahbod, Defense R&D Canada Valcartier; Stephen Neville, University of Victoria; Adel Guitouni, Defense R&D Canada Valcartier; Sudhakar Ganti, Yvonne Coady, and Burak Martonalt, University of Victoria

Yagiz Onat Yazir presented this poster about moving to a model of provisioning virtual machines with loosely defined resources instead of asking customers to specify a priori CPU, memory, and bandwidth requirements. This would make cloud computing services more like the public utilities that we are used to. In the case of the cloud, loosely defined resource requirements map to "enough" CPU, memory, and bandwidth to achieve a set response time for requests under the current load.

In this kind of model, virtual machine resource limits vary dynamically, which means it is neither efficient nor realistic to statically allocate virtual machines to physical nodes. VMs must migrate between nodes when a node becomes over- or underutilized because VM resource constraints changed.

Centralized VM allocation algorithms can produce a near-optimal configuration, but they require far too many migrations to be practical. The Multiple Criteria Decision Analysis technique (or PROMETHEE method) presented in this poster deals individually with problem nodes in order to limit migrations.

The allocater migrates VMs whenever an anomaly—an over- or underutilized node—becomes apparent. To deal with overutilization, it migrates VMs away from the node until it is correctly provisioned, and to deal with underutilization, it evacuates the node so it can shut down. The resulting VM allocations do not distribute resources as effectively as centralized provisioning, but they allow resource constraints to vary without an impractical amount of VM migrations.

Remote Desktops Using Streaming VM Record/Replay

Nitesh Mor, Shikhar Agarwal, Sorav Bansal, and Huzur Saran, Indian Institute of Technology Delhi

In traditional remote desktop systems, a lot of bandwidth is dedicated to sending images of the remote system's desktop back to the client.

The system presented in this poster instead transfers virtual machine replay information, which often requires substantially less bandwidth than desktop graphics. The remote machine runs on virtual machine software that supports virtual record and replay. All the recorded replay information (such as interrupt timings) is sent over the network to a virtual machine running on the client. The client virtual machine can replay the remote machine's execution in real time, as it receives the replay data.

The most dramatic speed improvement comes from the bandwidth reduction when playing a DVD, which over a traditional remote desktop connection requires streaming the DVD video itself. Streaming video is not necessary with VM record/replay, but initial state of both VMs must be synchronized, meaning that the DVD's contents are already available at the client side. Because of the synchronization requirement, which involves an initial disk transfer of the VM, the technique is most suitable for a client making frequent remote desktop sessions to the same remote VM, or to multiple VMs which share similar disk images.

Porting File System Structures to Nameless Writes

Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Commercial flash drives keep a mapping table from logical block addresses to physical block addresses, which the drive can use to dynamically remap blocks for the sake of wear leveling. Yiying Zhang explained that these tables become more wasteful as drives get bigger: a 2TB SSD would need 30GB of mapping tables for a page-level mapping using a standard flash page size. Although mapping strategies exist that can keep smaller mapping tables, they have a higher I/O cost.

Zhang and co-authors show how device support for "nameless writes" makes it possible to modify file systems to operate efficiently without device remapping and without modifying on-disk structures. A nameless write gives the disk the task of choosing a free block to write to. Because the disk chooses which blocks to write, it no longer needs to remap blocks to effectively spread out writes. The nameless write operation returns the physical block index to the file system.

To handle blocks that must be relocated after wearing out, the disk will also need to support a device callback, notifying the file system that it has to update pointers to the relocated block.

DiCE: Predicting Faults in Heterogeneous, Federated Distributed Systems

Vojin Jovanović, Marco Canini, Gautam Kumar, Boris Spasojević, Olivier Crameri and Dejan Kostić, EPFL, Switzerland

Marco Canini and Dejan Kostić presented DiCE, designed to analyze heterogeneous, federated distributed systems such as BGP, the Internet's routing protocol. DiCE can take a complete snapshot of all the nodes in the analyzed system and then explore possible system behaviors from that snapshot using "concolic" execution, a hybrid of symbolic and concrete execution. In BGP, this approach can find configuration errors that might allow prefix hijacking or policy conflicts. Fuzz testing makes it practical to search for valid messages that have the potential to harm the system.

Fifth set of posters summarized by Edmund L. Wong (elwong@cs.utexas.edu)

A Replay-based Approach to Performance Analysis

Anton Burtsev, Eric Eide, and John Regehr, University of Utah

Anton Burtsev presented a novel way of approaching performance analysis of complex software systems with a full-system deterministic replay. Burtsev leveraged Xen VMM to record the entire execution history of a system with a constant overhead of several percent. An analysis algorithm is invoked on a copy of the original execution, which is recreated by means of replay mechanisms offline. Burtsev argued that such an approach turns performance, a dynamic property of a particular execution, into a static property that can be analyzed separately from an actual instance of a running system. To provide general support for replay-based performance analyses, Burtsev suggested a general analysis framework which combines traditional replay mechanisms with a realistic performance model, and a semantic interface to the behavior and the runtime state of the system.

Mitigating Risk in Green Cloud Computing

Sakshi Porwal, Muki Haklay, John Mitchell, Venus Shum, and Kyle Jamieson, University College London

Sakshi Porwal and co-authors studied how power consumption, an important consideration for cloud computing providers, can be reduced. Porwal showed two models of computing: the waterfall model, in which tasks are assigned to an idle machine only after all other non-idle machines were fully utilized, and the distributed model, in which tasks are load-balanced across multiple machines. She showed that the waterfall model reduces power consumption and produces less CO2 as a result. Porwal also explored distributing tasks to sites which are located in geographically colder areas and thus require less power for cooling mechanisms, which represent approximately a third of power consumption at datacenters.

It Wasn't My Fault: Understanding OS Fault Propagation Via Delta Execution

Cristiano Giuffrida, Lorenzo Cavallaro, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Cristiano Giuffrida proposed characterizing faults in OSes by introducing faults into the system, isolating the faulty execution, and comparing it to a fault-free execution to see how the execution, state, and environment differed. By creating a catalog of faults and their effects, Giuffrida envisioned that an OS can be developed that can recover from failures and continue execution. This catalog could be used to remove traces of faulty behavior without affecting correct parts of the execution.

Fine-grained OS Behavior Characterization

Lorenzo Cavallaro, Cristiano Giuffrida, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Lorenzo Cavallaro proposed building a microkernel-based OS with extremely focused and small components. The behavior of these components could be automatically profiled and characterized by using an IPC-based monitoring scheme. These profiles could then be later used to detect anomalous behavior and bugs in OS components. Cavallaro argued that the overhead of such a system could be ameliorated by giving users the option to adjust the granularity and accuracy of the monitoring infrastructure.

Resurrecting Static Analysis: Exploiting Virtualization to Address Packing and Encrypted Malware

Christopher Benninger, Niko Rebenich, and Stephen W. Neville, University of Victoria; Rick McGeer, HP Labs; Yvonne Coady, University of Victoria

Christopher Benninger proposed a new technique for detecting malware that may be "packed" (encrypted or compressed). Instead of relying on static techniques alone, which are difficult and often fail when malware is packed, or dynamic techniques, which are not as successful as static techniques before the advent of packing, Benninger argued that malware can be more accurately detected by exposing malware to static analysis after it has unpacked itself and right before it attempts to execute. Toward this goal, he developed an event-driven platform for identifying packed malware running on a VM, an introspection tool for accessing an offending process's memory space from a separate VM, and an analysis tool for identifying and flagging malware variants.

Block-IO Scheduling for Guaranteed Scalable Storage Performance

Bader Al Ahmad, Sriram Murali, and Sathish Gopalakrishnan, University of British Columbia

Sriram Murali observed that many applications require real-time interactivity, yet many of them run on cloud environments consisting of virtual machines handling multiple clients. Murali argued that disk I/O is the major barrier to achieving guaranteed QoS for virtual machines; thus, efficient disk utilization is critical to providing real-time guarantees. Towards this goal, Murali implemented a scheduler based on weighted fair-queuing in the block device driver of the Xen hypervisor which offers hard and soft real-time guarantees to the different virtual machines accessing the disk. This scheduler is scalable from commodity servers to enterprise cloud-based solutions.

SlapChop: Automatic Microkernels

Sara Dadizadeh, Jean-Sébastien Légaré, and Andrew Warfield, University of British Columbia

Modern software is complex and monolithic. Sara Dadizadeh presented SlapChop, a system for decomposing a large system into small, single-purpose source code libraries that perform specialized tasks. SlapChop dynamically analyzes a running system, collects traces, finds the parts that are relevant to the task that is to be isolated, maps those instructions back to the original source, and generates specialized libraries consisting of this source code. She showed an example of SlapChop being performed on an OS kernel.

Benchmarking Online Storage Providers—Does Bandwidth Matter?

Andi Bergen, University of Victoria; Rick McGeer, HP Labs; Justin Cappos, University of Washington; Yvonne Coady, University of Victoria

Andi Bergen argued for the importance of benchmarking online storage providers, which are becoming increasingly popular. Because users are often bandwidth-limited in their own connections, Bergen argued that such measurement should be done in a distributed fashion, by having multiple sites perform many operations on the storage providers. His hope is to develop a tool that allows for customizable benchmarking depending on what metrics users are interested in.

Measurements of Personally Identifiable Information Exposure on the Web

Xiao Sophia Wang, University of Washington; Sam Burnett, Georgia Tech; Ben Greenstein, Intel Labs Seattle; David Wetherall, University of Washington

Personally identifiable information (PII), such as names, addresses, and credit card numbers, are being used online as a part of the Web, yet very little is known about how prevalent PII exposures are in practice. Xiao (Sophia) Wang showed that, although users would expect that PII is only sent to the Web sites they are visiting, this information is often sent to third-parties or easily gleaned by eavesdropping or through cookies. Studying the top 100 Web sites in the US, Wang, surprisingly, found that 35% send passwords in the clear, 26% send some form of PII to third parties, and 54% store some form of PII in cookies.

Depot: Cloud Storage with Minimal Trust

Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish, University of Texas at Austin

Sangmin Lee and Srinath Setty presented Depot, a cloud storage system that minimizes trust for safety and for liveness. Depot achieves this through a new consistency model, fork-join-causal consistency, which guarantees causal consistency in the absence of faults and handles forks by reducing equivocation to concurrency. Despite its minimal reliance on trust, Depot still provides useful properties: safety properties include eventual consistency, fork-join-causal consistency, bounded staleness, integrity, recoverability, and eviction of faulty nodes. Liveness properties include allowing nodes to always write, always exchange updates, and read when correct nodes have the required objects. Lee and Setty argued that Depot's cost is low in terms of latency, and its weighted dollar cost is modest.

Iodine: Interactive Program Partitioning

Nathan Taylor and Andrew Warfield, University of British Columbia

Clearly identifying important parts of modern software is complicated but imperative for understanding whether exploits or bugs may exist in the code. Nathan Taylor described his system, Iodine, as a way to discover these components through techniques inspired by MRI. Iodine outputs a control-flow graph with edges representing control-flow connections and colors representing how frequently a particular component is visited. These colors eventually fade if a component is no longer visited, giving the user an interactive idea regarding what code is being executed over time. The system provides users with an interactive interface, allowing users to poke into the code and explore the effects that unvisited branches have on the graph.

Masking Network Heterogeneity with Shims

Danny Yuxing Huang, Williams College; Eric Kimbrel and Justin Cappos, University of Washington; Jeannie Albrecht, Williams College

Danny Yuxing Huang tackled the problem of dealing with heterogeneous network environments—those which contain NATs, VPNs, firewalls, mobile devices, etc.—by developing a new technique for building applications. A common solution involves using libraries to virtualize network heterogeneity. Huang argued that porting programs with these libraries to deal with specific types of network heterogeneity is tedious and error-prone; due to semantic differences, the programs and/or the libraries must be modified to work together. Instead, Huang proposed the use of formally verified wrappers, or shims, that abstract away the details of providing functionality such as NAT traversal or mobility support from applications. Unlike the above-mentioned libraries, shims use an interface that is semantically identical to the underlying network interface. Thus, shims are completely transparent to the application, require no modification of the original code, and may be composed together dynamically. Huang demonstrated his technique by constructing a NAT traversal shim that enables an unmodified server application to accept incoming connections from behind a NAT device.

Reducing System Call Latency via Dedicated User and Kernel CPUs

Josh Triplett, Philip W. Howard, Eric Wheeler, and Jonathan Walpole, Portland State University

Systems have an increasing number of cores, allowing tasks to often have their own CPUs. Josh Triplett observed, however, that tasks still time-slice between user and kernel modes. Triplett proposed leveraging the increasing number of cores by providing each application with a dedicated syscall thread, which stays in-kernel. This thread has its own CPU and performs the actual system calls, in order, on behalf of application threads. Triplett argued that his approach does not require modification of user code or existing kernel syscalls and does not require a new type of thread or scheduler, while improving the performance of common syscalls in Linux.

Sixth set of posters summarized by William Enck (enck@cse.psu.edu)

Deterministic Concurrency within the OpenMP Framework

Amittai Aviram and Bryan Ford, Yale University

Available techniques to prevent shared-memory concurrency bugs are insufficient; developers infrequently adopt new languages, and deterministic thread schedulers make race conditions repeatable without eliminating them. Amittai Aviram (amittai.aviram@yale.edu) presented Deterministic OpenMP (DOMP), which simplifies the prevention of race conditions by following a programming model resembling document handling in version control systems. When concurrent threads start executing at a fork, DOMP makes a copy of the current shared program state for each thread. Once the threads finish and rejoin their parent, DOMP merges their writes into the parent's state; if two threads have written to the same memory location, DOMP signals the race condition as an error. DOMP will implement most of the core features of standard OpenMP, including the *parallel, for,* and *sections* directives, as well the *reduction* clause, thus making it easy to parallelize existing sequential code in conventional languages deterministically by adding just a few annotations.

Diagnosing Intermittent Faults Using Software Techniques

Layali Rashid, Karthik Pattabiraman, and Sathish Gopalakrishnan, University of British Columbia

Layali Rashid (lylrashid@gmail.com) explained that today's complex hardware chips are prone to intermittent errors. When errors are detected, faulty components should be disabled. Given the existence of an error, this work tracks program dependencies to the instructions where the error originated. Using the properties of the offending instruction, the faulty logic unit is diagnosed and disabled to ensure that future intermittent errors cannot affect program correctness.

NanoXen: Better Systems Through Rigorous Containment and Active Modeling

Chris Matthews, University of Victoria; Justin Cappos, University of Washington; Yvonne Coady, University of Victoria; John H. Hartman, University of Arizona; Jonathan P Jacky, University of Washington; Rick McGeer, HP Labs

Current systems lack robustness and security. This work, presented by Chris Mathews (cmatthew@cs.uvic.ca), proposes a new computational model based on "virtual components." In this model, applications execute as sets of components. Components are primitive computational units with well-defined semantics. The components execute inside of either "native client" (NaCl) containers or virtual machine (VM) containers, depending on isolation requirements. While this work is at a very preliminary state, Matthews and his co-authors aim to use these primitives to design a more robust and secure operating system environment.

Namespace Composition for Virtual Desktops

Dutch Meyer, Mohammad Shamma, Jake Wires, Maria Ivanova, Norman C. Hutchinson, and Andrew Warfield, University of British Columbia

Dutch Meyer (dmeyer@cs.ubc.ca) explained that many enterprises are beginning to deploy PCs as thin clients running virtual machines served from back-end servers. These systems generally operate at the block level, which is semantically poor and difficult to administer. Meyer and his co-authors proposed a model that serves file namespaces instead of raw block devices. The proposed system can create namespaces for new VMs on the fly and can merge namespaces from multiple clients to create new views of the system helpful for collaboration and administration. By focusing on file semantics, redundant storage and scanning can be eliminated, and overall management complexity can be reduced.

Compression of High Resolution Climate Data

Jian Yin and Karen Schuchardt, Pacific Northwest National Lab

Climate simulations consume and produce petabytes of data. Traditional compression algorithms perform poorly on this data and frequently require the entire archive to be decompressed before being used by simulations and analysis tools. Jian Yin (jian.yin@pnl.gov) and his co-authors developed a compression algorithm based on properties of the climate data that allows block-based decompression for use of subsets of the overall dataset. They use heuristics to determine where to break the compression blocks such that the decompression and analysis can be pipelined. Additionally, the decompressed blocks are cached to avoid redundant decompression and speed analysis. Finally, they include prediction algorithms to decompress data blocks so that new blocks are immediately available.

Informed System Design through Exact Measurement

Daniel A. Freedman, Tudor Marian, Jennifer Lee, Ken Birman, Hakim Weatherspoon, and Chris Xu, Cornell University

Daniel A. Freedman (dfreedman@cs.cornell.edu) considered the application of a class of exact network measurements to help inform system design, particularly for architectures that involve the intersection of endpoint systems and network links. He discussed the design of network instrumentation—using physics test equipment, such as oscilloscopes, pattern generators, lasers, etc.—for the exact capture of packets in flight, and they demonstrate its application for a particular deployed 10 Gigabit Ethernet wide-area network (WAN). In fact, on such a WAN, they observe anomalous behavior that contests several common assumptions about the relationship between input and output traffic flows. Finally, the authors connect their observations of emergent packet chains in the network traffic with higher-level system effects—namely, to explain previously observed anomalous packet loss on receiver endpoints of such networks.

Accele Scheduler: Energy Efficient Virtual CPU Scheduling for Modern Multicore CPUs

Tetsuya Yoshida and Hiroshi Yamada, Keio University; Hiroshi Sasaki, University of Tokyo; Kenji Kono, Keio University; Hiroshi Nakamura, University of Tokyo

CPU chips now frequently include multiple processing cores, but not all of the cores always need to run at their highest frequency. Dynamic voltage and frequency scaling (DVFS) improves energy efficiency by scaling down voltage and frequency when possible. On modern multicore chips, all cores must use the same voltage, due to architectural limitations, even if their frequencies are individually scaled. Tetsuya Yoshida (tetsuyay@sslab.ics.keio.ac.jp) and his co-authors have developed a scheduling algorithm for virtual CPUs that distributes the processing load to optimize energy consumption in such an environment. They have already achieved an energy delay product (EDP) of 23.6%, which is better than Xen's existing credit scheduler.

Redflag: Detailed Runtime Analysis of Kernel-Level Concurrency

Justin Seyster, Abhinav Duggal, Prabakar Radhakrishnan, Scott D. Stoller, and Erez Zadok, Stony Brook University

Justin Seyster (jseyster@cs.stonybrook.edu) explained that large code bases, such as the Linux kernel, are difficult to analyze for concurrency bugs. Seyster and his co-authors have developed a runtime analysis system to focus on the locking of any specific data structure. The tool provides lockset and block-based analysis. When working with code as large and complex as the Linux kernel, subtle complications result. For example, occasionally variables contain bit-fields to store system state. Therefore the tool must treat individual bits in such variables as variables themselves. The tool is also sensitive to the order in which locked blocks execute. Using their tool, Redflag, they have identified one race condition in their own file system code and independently discovered two additional known-concurrency issues in Linux's file system code. The authors hope to apply the analysis to other data structures within the Linux kernel.

Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns

Etienne Le Sueur and Gernot Heiser, University of New South Wales

Etienne Le Sueur (etienne.lesueur@nicta.com.au) and Gernot Heiser observed the power consumption of three multicore AMD Opteron systems over a seven-year period to study the effectiveness of dynamic voltage and frequency scaling (DVFS). They found four factors that cause DVFS to become significantly less effective: (1) scaling in semiconductor technology, (2) increased memory performance, (3) improved sleep/idle modes, and (4) multicore processors. They believe that in the future, cache management techniques such as turning of parts of the L3 cache, which is as large as 8MB, will be much more effective than DVFS at conserving energy.

Research Vision Session

Summarized by Shivaram Venkataraman (venkata4@illinois.edu)

The research vision session held at OSDI '10 was similar in spirit to the Wild and Crazy Ideas sessions at ASPLOS. It consisted of four presentations on systems research ideas for the future. Ice cream was served before the session began and the presentations offered interesting ideas, generated lively discussion, and were well attended.

Epoch Parallelism: One Execution Is Not Enough

Jessica Ouyang, Kaushik Veeraraghavan, Dongyoon Lee, Peter M. Chen, Jason Flinn, and Satish Narayanasamy, University of Michigan

Jessica Ouyang presented a new style of parallelism called "Epoch Parallelism," a novel approach to writing multithreaded programs. Since it is hard to write multi-threaded programs that are fast and correct, Ouyang proposed that programmers could write two programs: one which was fast but buggy and another which was slower and correct. Programs are then split into multiple epochs and the output from the first epoch of the faster program can be used to accelerate the execution of the following epochs in parallel. If a checkpoint from the faster program does not match that from the slower execution, the speculatively executed parts of the program are rolled back.

Michael Vrable from UCSD asked how this execution pattern could be used with I/O or network interactions. Ouyang agreed that output can be externalized to the user only when the slower process completed, but that different parts of the application could internally proceed using the speculative results. Emin Gün Sirer from Cornell inquired about applications which could make use of the correctness-speed tradeoff. Ouyang clarified that the faster executions in this model were not entirely buggy and would mostly give the correct answer. Examples for this included avoiding additional runtime assertion checks and optimistic concurrency techniques such as lock elision.

Automated Software Reliability Services: Using Reliability Tools Should Be as Easy as Webmail

George Candea, Stefan Bucur, Vitaly Chipounov, Vova Kuznetsov, and Cristian Zamfir, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

The second research vision proposed the creation of a software certification service and was presented by George Candea (EPFL). Although many tools are being developed to help test, debug, and verify correctness of software, Candea observed that these were not having much of an impact in the real world. To make software reliability more effective, Candea argued that reliable software should have an advantage over competitors in the market.

He then presented a case study of how Underwriter Labs had established a safety standard for electrical appliances early in the twentieth century and how a similar certification service could promote software reliability. This service would use automated techniques to objectively measure the reliability of binaries and would provide ratings similar to crash-test ratings for cars. Candea listed the systems research problems related to building such a service. These included automated proof generation techniques and scalable testing of binaries on massive clusters.

Peter Chen, University of Michigan, pointed out that there were blogs which provided such ratings for antivirus software today and that software was tuned to perform well on these benchmarks. Candea replied that the certification service would be based on concepts like model checking and symbolic execution, which would make tuning the software more difficult when compared to benchmarks. Ivan Beschastnikh, University of Washington, asked if the certification service was applicable to all kinds of software. He noted that comprehensive testing was already used in critical software in embedded systems and that users might not care much about the reliability of services like Facebook. Candea replied that today, users chose software based on functionality and that it could be possible that they would continue to do the same. However, he hoped that presenting data about reliability of software would help them make a better decision.

SIGOPS to SIGARCH: "Now it's our turn to push you around"

Jeffrey C. Mogul, HP Labs

Jeff Mogul from HP Labs presented his thoughts on how operating systems were being ignored by the computer architecture community and what the SIGOPS community could do to change this. Mogul argued that OS designers used commodity hardware to build their systems and had stopped asking the architects for newer features like core-to-core message passing primitives or interfaces for managing thermal issues.

Analyzing the problem in detail, Mogul pointed out that architects had no open-source simulator which could run real operating systems, and the OS community could help articulate and build such a simulator. Also, there was a need for operating system benchmarks which could be run by architects in simulators to evaluate how their changes affect the OS. Finally, Jeff also argued that SIGARCH should be encouraged to accept papers which focus on OS issues in conferences like ASPLOS.

This was followed by a lively question-answer session. Margo Seltzer from Harvard disagreed with the presentation and argued that the best compliment for an OS designer was for the user to not realize that the OS was running. Mogul replied that it would be some time before we had a zero-overhead operating system and that we need better cooperation between the hardware and the OS community for that to happen. David Lillethun from Georgia Tech asked how the inertia about changing instructions sets, especially x86 for desktops, affected some of the problems pointed out earlier. Mogul jokingly remarked that his terminal still runs on an Alpha computer and that he felt managing the memory system was a greater challenge than the instruction set. Rik Farrow argued that CPU architecture features, such as the trap into privileged mode, actually shape how operating systems work, noting that there was a paper in this conference about avoiding the penalties for making that context shift. Farrow applauded the idea that the systems and architectures groups would actually communicate, instead of systems researchers just working around problems caused by architecture. Emin Gün Sirer's comment, that he could imagine a corresponding talk at WWW about how the OS community had not provided the database community with the right interface and had forced them to write to raw disk, drew laughter and applause from the audience.

Embrace Your Inner Virus

Michael F. Nowlan and Bryan Ford, Yale University

The final research vision presentation at OSDI '10 was by Michael Nowlan from Yale University, who proposed a system design in which the OS expects applications to behave like viruses. First, Nowlan noted that viruses were prevalent and transferred easily across machines through the Internet and other media like Bluetooth and USB. Instead of trying to fight viruses, he proposed that the OS should be designed to handle viral applications (vapps) using techniques like code sandboxing and information flow control.

Nowlan presented several case studies of how vapps could be useful. The first case study was about a contact application which spread to all the users in a particular room. This could be useful for users to get in touch with other attendees at a conference. The second case study analyzed a photo uploading vapp that could be used to disseminate photographs to other machines as soon as they are taken. If the original machine is lost, users can recover their photos by being "re-infected" from other machines.

Nowlan also discussed various business models that could be used to build such viral applications. Most interesting among these was the "Quid Pro Quo" model where the user could consume a service for free but had to submit a local resource in exchange. For example, a weather application on a phone could be used for free but the user would need to upload the local temperature from the phone's thermometer. Following the presentation, Nowlan was asked if the bad vapps would not affect the system. He replied that they were proposing greater transparency into applications and the use of techniques like information flow control in OS design which could overcome such problems.

Deterministic Parallelism

Summarized by Alan Dunn (adunn@cs.utexas.edu)

Deterministic Process Groups in dOS

Tom Bergan, Nicholas Hunt, Luis Ceze, and Steven D. Gribble, University of Washington

Tom Bergan presented dOS, a set of modifications to the Linux kernel to allow groups of processes to run deterministically. Nondeterminism in OS execution makes bugs harder to reproduce and causes replicas in state machine replication-based fault-tolerance protocols to become unsynchronized. To avoid these problems, dOS introduces the ability to create a "deterministic box," in which contained processes execute in a deterministic manner. Bergan noted that dOS guarantees a deterministic ordering of internal operations (including inter-thread communication), but does not guarantee deterministic performance.

Deterministic execution is not a new problem, but the dOS project improves over prior work both conceptually and in practice. Bergan explained that dOS incorporates a distinction between "internal" and "external" nondeterminism. The difference is that internal nondeterminism is controllable by the operating system, while external nondeterminism is caused by uncontrolled sources such as user input and network packets. Internal nondeterminism can then be controlled by use of algorithms that ensure deterministic outputs, such as deterministic ordering of IPC operations and process scheduling. Only external nondeterminism needs to be logged and replayed to processes. Bergan pointed to log sizes that were several orders of magnitude smaller than a prior full-system determinism system as evidence for the utility of the selective determinism approach.

The main pieces of dOS's implementation are the DMP-O deterministic scheduler and a "shim" for monitoring external inputs and controlling their delivery. While DMP-O is from prior work, dOS still needed a way to detect communication between threads, which was accomplished by modifications to system calls and page protections for shared memory. Threads can then be run in parallel until communication occurs and communication events are serialized in a deterministic order. Discussion of the shim included two examples: deterministic record and replay, and replication of a multi-threaded server.

Questions for the presentation ranged over topics from technical comments on performance to high-level questions about design motivation. Amittai Aviram (Yale University) and Micah Brodsky (MIT) asked whether performance issues incurred by fine-grained shared memory between processes could be resolved. Bergan commented that previous systems had used specialized hardware or compiler-inserted instrumentation to solve this problem, and that still other approaches might be possible. Madan Musuvathi of Microsoft Research wondered whether nondeterminism was such a bad thing and whether there might be tradeoffs between degrees of nondeterminism and performance. This was a key theme for this session, as the three different systems presented managed this tradeoff in different ways. Bergan said that the notions of internal and external nondeterminism were dOS's way of handling this: in dOS, one can select the processes that need to be deterministic, and thus influence what portions of nondeterminism are internal and external.

Efficient System-Enforced Deterministic Parallelism

Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford, Yale University

Awarded Jay Lepreau Best Paper!

Bryan Ford presented Determinator, a microkernel designed to eliminate data races, which was awarded one of two Jay Lepreau Best Paper awards for OSDI '10. There are currently many systems devoted to helping programmers manage existing data races that occur in conventional operating systems. Ford said that despite all of this, it would be nice if programmers didn't have to worry about data races and were not forced into using specific programming languages to achieve this end. Determinator is an attempt to accomplish this: it is a microkernel with a small set of system calls to manage shared state. Facilities that would be present in modern monolithic UNIX systems, such as the C library, process management, and file system API, are instead implemented in user space and take advantage of these system calls to eliminate data races.

The microkernel system calls are built on the paradigm of "check-out/check-in" of state. When a kernel thread forks, it creates a local copy of its parent's address space. Any reads and writes that this thread performs are then against its local copy. When the thread joins its parent, its differences from the parent are merged back to the parent's state. Ford described how this solves data race issues: read-write races are eliminated, because writes will occur to a different local state than reads, while write-write races are detected upon join and can be trapped and then resolved.

Determinator was evaluated by comparing its speed on benchmark programs to that of Linux. The primary factor influencing benchmark speed was the granularity at which the changes children made to shared state were found and merged back into a parent, with finer granularity leading to worse performance.

There were a number of interesting questions after this presentation. An audience member from Microsoft Research asked about the relationship between Determinator's approach to shared state and transactions. Ford responded that the two have similar isolation properties, but that transactions generally do not provide determinism. Eric Eide of the University of Utah asked about whether it might be better to mandate use of a specific language for Determinator. Ford said that while a single language might help by allowing easier introduction of primitives to manage granularity of shared state, it seemed advantageous to allow application programmers greater language freedom. Jason Flinn of the University of Michigan commented that experience with systems like CODA showed that resolving write-write conflicts can be challenging, and he wondered how difficult this was with Determinator. Ford said that his team did not have enough experience writing applications for Determinator to comment definitively, but that he believed there would be some issues to explore with respect to when changes to state are propagated.

Stable Deterministic Multithreading through Schedule Memoization

Heming Cui, Jingyue Wu, Chia-che Tsai, and Junfeng Yang, Columbia University

Heming Cui presented Tern, a system for improving system determinism via capturing and reusing thread schedules. Tern shares some of the motivation of other papers from the Deterministic Parallelism session in that it also aims to allow for easier testing and debugging through increased determinism. However, Tern also targets a difficulty found in other systems: often a small change in program inputs causes radical changes in the schedules produced, which eliminates some of the benefits of deterministic scheduling. Cui described how Tern avoids this by representing schedules as sequences of synchronization operations, which can potentially be reused for multiple inputs. It calculates sets of constraints that must be satisfied by an input to use a certain schedule and caches schedules with their constraints.

Cui illustrated the use of Tern with example code from PBZip2. To use code with Tern, it is necessary to first annotate the variables that are important to scheduling. The code is then instrumented at compile time to allow Tern to record and control scheduling. For any given program run, constraints on the annotated variables are created based on the branches that the program takes. These constraints are stored with the resultant schedule in a memoizer. The constraints will be evaluated for subsequent program runs, and the stored schedules will be used if their constraints are satisfied.

Tern was evaluated based on its degree of schedule reuse, stability in bug reproduction, and overhead. For several real-world programs (Apache and MySQL), Tern was able to use 100 or fewer schedules to cover significant stretches of execution, corresponding to a 90% schedule reuse rate. Computational overhead was often less than 10%.

Ding Yuan of the University of Illinois at Urbana-Champaign claimed that Tern would not eliminate data races. Cui pointed out that this is true, although this was a tradeoff made to allow schedule reuse, thus increasing performance and stability. An audience member asked about whether constraint size and number could affect performance, and Cui said that they have techniques for removing redundant constraints and eliminating irrelevant constraints to mitigate this issue. Bryan Ford of Yale asked whether Tern memoizes schedules only at whole program granularity, and whether other granularities would be useful. Cui responded that currently only whole program granularity is supported, but that others could be useful.

Systems Management

Summarized by Don Porter (porterde@cs.utexas.edu)

Enabling Configuration-Independent Automation by Non-Expert Users

Nate Kushman and Dina Katabi, Massachusetts Institute of Technology

Nate Kushman presented KarDo, a tool for automating configuration and other administrative tasks. The key problem addressed is that users generally know what they want, but do not know how to do it. There is no easy way to automate fairly common tasks that require GUI manipulation. The KarDo system improves on the state of the art by collecting a few traces (generally two) of experts performing a given task, and then distilling a generalized trace which can be replayed on an end user's computer. KarDo requires no kernel or application modifications. Kushman presented details of how they generalize their traces. KarDo uses a support vector machine to classify GUI actions into three categories: update, commit, and navigate. KarDo uniquely identifies GUI widgets by their text, which is extracted from the accessibility interface. In generalizing a trace, navigation is separated from the rest of the trace; using all traces, the needed navigation steps are generalized for a canonical trace. Kushman also presented algorithms for removing needless actions and handling differences in the GUI.

KarDo was evaluated using 57 tasks taken from the Microsoft Help and eHow Web sites. The authors collected traces from a set of diversely configured virtual machines and then tested them on a different set of virtual machines. Existing automation techniques worked on only 18% of the test virtual machines, whereas KarDo had an 84% success rate. Of the 16% of tests that failed, 4% were navigation errors, 5% had missing steps, and 7% were classifier errors. More traces should lower these error rates.

Adam Chlipala of Harvard asked why they didn't just parse configuration files, and Kushman said that they could go quite a bit farther than just parsing files. Someone asked whether a collected trace could reveal private information. Kushman responded that the traces used for training can include private data, but if any two traces have different values, these are considered private and filtered out of the canonical trace. Josh Triplett from Portland State University asked whether the same tool could be applied to command-line tasks. Kushman responded that a key challenge is text processing, which would require additional insights to produce canonical traces of the same quality.

Automating Configuration Troubleshooting with Dynamic Information Flow Analysis

Mona Attariyan and Jason Flinn, University of Michigan

Mona Attariyan presented ConfAid, a tool that automatically identifies the root cause of a configuration error. The motivating problem is that configuring software is difficult and prone to user error. Currently, if a user can't get a piece of software to work, s/he must ask colleagues, search a manual, or look at the code if it is available. Users need better tools to troubleshoot these errors. The usage model of ConfAid is that a user runs the misconfigured software under ConfAid and it outputs a list of likely root causes.

ConfAid works by tracking dynamic information flow through the application binaries. ConfAid does not require source code or debugging symbols. Attariyan described several technical details of how they use information flow analysis of control and data flows to assess which configuration variables influenced the program control flow that led to the incorrect behavior. In this analysis, there is a fundamental tradeoff between the precision of the analysis and performance; ConfAid uses imprecise analysis and introduces three heuristics to improve performance and lower the false-positive rate.

To evaluate ConfAid, the authors tested 18 real-world errors from OpenSSH, Apache, and PostFix (collected from manuals, support forums, etc.), and 60 randomly generated errors. For the real-world errors, ConfAid ranked the correct root cause first for 72% of the errors and second for the rest. Among the random errors, ConfAid ranked the correct root cause first or second for 55/60 errors. The average execution time of ConfAid was 1 minute, 32 seconds.

Someone asked whether ConfAid required an understanding of configuration file syntax, Attariyan responded no. Alice Zheng of Microsoft Research asked if all bugs the authors dealt with were single-mistake bugs, and whether they expected all bugs in the real world to be single-mistake. The authors only looked at single-mistake bugs. ConfAid would likely suggest both mistakes independently, but cannot output that both configuration variables must be changed in tandem. Regarding real-world configuration errors, it is hard to say, but the test errors they used seemed quite common on the forums.

Lunchtime Award Announcements

Summarized by Rik Farrow (rik@usenix.org)

Two awards were announced during the symposium luncheon. Robert Morris of MIT received the Mark Weiser award for an individual who has demonstrated creativity and innovation in operating systems research. Roger Needham and Michael Schroeder received the SIGOPS Hall of Fame Award for their 1978 paper on authentication over a non-trusted link (Needham-Schroeder protocol). Schroeder was present to accept the award.

Inside the Data Center, 2

Summarized by Don Porter (porterde@cs.utexas.edu)

Large-scale Incremental Processing Using Distributed Transactions and Notifications

Daniel Peng and Frank Dabek, Google, Inc.

Frank Dabek described the new Percolator indexing system used at Google to reduce the delay between Web crawling and re-indexing. Prior to Percolator, there was a delay of days between crawling a new document and its incorporation into the index, as the entire corpus of crawled documents was reindexed. The solution described is incremental reindexing, which introduces a number of new challenges, described in the talk. In order to support incremental indexing, the authors added distributed transactions to BigTable, which provides snapshot isolation semantics, and added notification support.

Among the challenges described was the problem of "Bus Clumping," the problem that the randomized scans for new work tend to clump working on data that is time-consuming to process, reducing parallelism and overloading servers. The solution they adopt is trying to acquire a lightweight scanner lock per row of BigTable. If the lock acquire fails, the scanner jumps to a random point in the table to look for new work—the equivalent of a city bus teleporting ahead in the route. Percolator also stressed certain new errors in the Google cluster, including a set of failing CPUs that randomly failed to XOR bits correctly, and an incorrect resistor value that powered off a certain motherboard. Dabek concluded with advice based on this experience: (1) push performance debugging through all layers of the system and (2) expect weirdness proportional to machine count.

The talk concluded with the assertion that Percolator is an existence proof that distributed transactions can be implemented at Web scale. This was reflected in a question from David Cock of the University of New South Wales regarding the novelty of the work; Dabek answered that the novelty is the scale of the system, which the field had given up on. Mehul Shah of HP Labs asked about the limits of the system and how it handled stale locks left by clients. Dabek responded that the largest problem with concurrency was heavy write conflicts, which were addressed with a backoff heuristic. The space required to store notifications in BigTable is not an issue, and stale locks were cleaned up lazily. Margo Seltzer of Harvard University asked his thoughts on the debate between MapReduce versus databases. Dabek said that if you pretend MapReduce is a database, it is a bad one, but that MapReduce is not dead and is still used heavily within Google.

Reining in the Outliers in Map-Reduce Clusters using Mantri

Ganesh Ananthanarayanan, Microsoft Research and UC Berkeley; Srikanth Kandula and Albert Greenberg, Microsoft Research; Ion Stoica, UC Berkeley; Yi Lu, Microsoft Research; Bikas Saha and Edward Harris, Microsoft Bing

Srikanth Kandula presented the Mantri system, used in production at Microsoft Bing. The goal of Mantri is reducing outliers, which slow down MapReduce jobs; reducing outliers improves productivity, gives cloud service providers more predictable completion times, and better utilizes datacenter resources, saving money.

One key cause of outliers is unavailable inputs at a computation node. To address this, Mantri replicates intermediate data and introduces heuristics to predict what data to replicate where, weighted by the cost of recomputation. A second key cause of outliers is variable network congestion, which Mantri addresses by carefully placing reduce tasks such that traffic on a link out of a rack is proportional to the bandwidth. Although global coordination to load-balance the network links across all jobs is difficult, having each job balance its own traffic is a good approximation of the ideal. A final cause of outliers is workload imbalance, often due to contention at a given machine. There is a long tail (25%) of miscellaneous causes for this.

Based on experience working with Mantri, Kandula recommended three principles for managing MapReduce systems: (1) predict to act early, (2) be aware of resource and opportunity cost of an action, and (3) act based on the cause. The overall result of deployment in the Bing cluster is a median 32% reduction in job completion time and lower utilization.

Someone asked what Kandula would do if they had a non-uniform cluster, say, with some machines that were faster but handled fewer requests. Kandula answered that they have a scheduling system that normalizes machine capabilities to a slot abstraction, which are scheduled rather than entire machines. Emin Gün Sirer from Cornell asked how the authors might change the MapReduce interface to address stragglers, given what they know from experience. Kandula responded that they would have an interface to yield more even partitions than simple hashing.

Transactional Consistency and Automatic Management in an Application Data Cache

Dan R.K. Ports, Austin T. Clements, Irene Zhang, Samuel Madden, and Barbara Liskov, MIT CSAIL

Dan Ports presented a system called TxCache, which provides transactional consistency for an in-memory database cache such as memcached. The key problem this work addresses is that modern Web applications face immense scaling challenges, but existing caching techniques only offer limited help. For instance, personalization on sites like Facebook foils whole-page caching. Similarly, database caches are of limited use, since Web applications require increased post-processing of data in the application itself. Application layer caches, such as memcached, provide a useful alternative. By caching application objects, these caches can separate common and customized content and reduce overall server load. The key challenge to this approach is that current application-level caches do not provide transactional consistency, leaving the application to address transient anomalies.

Ports then described the TxCache system, which provides a simple interface to delineate transactions on cacheable data. TxCache provides bounded staleness for transactions, allowing read-only operations to improve performance by returning slightly stale data where safe. Programmers can also specify cacheable, side effect–free functions, allowing the system to cache their results and avoid needless recomputation. Ports then described several key challenges, including selection of timestamp intervals and maintaining coherence through invalidations.

The system was evaluated using the RUBiS benchmark with a single database server and nine front-end/cache servers. The experiments showed that a larger cache yielded a higher hit rate and better performance. Allowing stale results also improves performance by as much as 3–5x, but the knee of the performance curve was around 20–30 seconds. The authors also measured the overhead of consistent caching by allowing inconsistent reads; performance improved only marginally, indicating that the costs are negligible.

Marcos Aguilera of Microsoft Research asked whether TxCache provided serializability or snapshot isolation. Ports answered that the system provides either, dictated by the guarantees of the underlying database. Ports was also asked whether they got performance wins on a single system, or only across several nodes. He answered that the number of nodes determines the size of the cache and the hit frequency.

Piccolo: Building Fast, Distributed Programs with Partitioned Tables

Russell Power and Jinyang Li, New York University

Russell Power presented Piccolo, a framework for developing high-performance distributed applications. Problems that can fit into memory in a cluster are a key motivation for this work. Power structured the talk around page rank as a representative example. Existing data flow models, such as MapReduce, don't expose global state, and models such as MPI and RPC require explicit communication, making them harder to write. The goal of Piccolo is to provide distributed, in-memory state. The runtime system transparently handles communication when the programmer requests reads and writes.

Power described several technical challenges in developing a page rank example application on Piccolo and challenges in implementing Piccolo on a cluster. Throughout the talk, he refined a pseudocode example that was both reasonably detailed and simple enough to fit onto one slide. For instance, Piccolo must add synchronization primitives for concurrent writes to a variable. Because many writes are actually accumulator functions, these can be used with release consistency to improve concurrency. Power also explained that storing state at nodes increases the complexity of load balancing, as it is harder to start new jobs. Starting jobs at remote nodes gives up locality and harms performance; moving is hard because the old site must still forward updates to the new site.

The system was evaluated on a cluster of 12 nodes totaling 64 cores. Compared to Hadoop, calculating page rank on a 100M page baseline, Piccolo was substantially faster. The iteration time remained nearly flat as more workers were added to match a larger input graph, indicating near-perfect scaling. The Piccolo code is available at piccolo.news.cs.nyu.edu.

Josh Triplett asked what fall-back strategy they used for non-commutative aggregate functions. Power answered that these were relatively rare in their experience and that they could use pairwise locking, but that locking was slow enough to avoid at all costs. Daniel Greenway asked about checkpointing and what they did if nodes fail. Power replied that they roll all nodes back to the last checkpoint.

Cloud Storage

Summarized by Katelin Bailey (katelin@cs.washington.edu)

Depot: Cloud Storage with Minimal Trust

Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish, The University of Texas at Austin

Prince Mahajan presented Depot, an attempt to remove trust from cloud storage. This system was unable to completely remove trust from the equation, but Mahajan argued that it comes very close: for *put* availability, consistency and staleness detection, the system requires no trust, while it minimizes the trust necessary for *get* availability and durability (reliant on one node).

Depot has multiple failover servers and can default to client-to-client communication in the case of errors. As with other work in the area, Depot uses both local state and metadata added to commits to allow clients to check the history and detect forks in the state. The system uses Fork-Join-Causal consistency in the case of unreliable nodes, which allows for taking a forked subset and reconciling it as if it were two concurrent commits from "virtual nodes." It also allows for the eviction of a node that is consistently faulty or malicious. Mahajan then covered implementation details and performance evaluation for the Depot project and the related "teapot" project implemented on Amazon's S3, demonstrating almost unmodified server-side code. He claimed that the performance overhead was modest and the implementation practical for use.

There were a large number of questions following the talk. Dave Koch of NICTA pointed out that CPU overheads on clients were high. Mahajan conceded that one of the graphs showed a 400% CPU overhead for one test, but argued that CPU cycles are cheap enough to allow this overhead to be modest nonetheless: throughput is the concern. David Schultz pondered the correctness of clients during reads: for example, a Byzantine client who had the only copy of some data. Mahajan clarified that individual clients can have replication specifications or filters, such as only reading data that is resident on four nodes. This would, however, reduce availability when only one node is online, as Schultz pointed out.

Comet: An Active Distributed Key-Value Store

Roxana Geambasu, Amit A. Levy, Tadayoshi Kohno, Arvind Krishnamurthy, and Henry M. Levy, University of Washington

Roxana Geambasu presented Comet, a variation on distributed hash tables (DHT), which allows applications to make use of small bits of code inserted in the DHT. Motivated by earlier work (Vanish, Geambasu 2009) which exposed the frustrating nature of working with a one-size-fits-all DHT serving many applications, the project produced a system that is flexible, lightweight, and provides isolation of the included code. The goal was to create an extensible DHT that can offer different capabilities to the different applications.

The implementation of this project focused on a unit called the Active Storage Object (ASO), which consists of data and tiny snippets of code, written in a very restricted form of Lua. The architecture of the system consists of an active runtime over the standard DHT, which ASOs can access via handlers and an ASO API. The ASO is sandboxed and has a very limited interface to the outside world, enhancing the overall security of the system. The sandbox allows the ASO to have some knowledge of the host and the DHT in general, but restricts actions to periodic tasks, interceptions of accesses, and minimal DHT actions (put, get, lookup). Geambasu demonstrated how even these limited ASOs can create powerful capabilities for the application. Her examples include an altered replication policy, a proximity tracker for closest peers, and self-monitoring code. She pointed out that Comet not only allows policies to differ between applications, but opens up new opportunities for tracking and analyzing usage, or even adding debugging or logging possibilities.

Pietros Maniatos of Intel Labs, Berkeley, wondered about security. Geambasu noted that there were global restrictions set on the ASOs such that they could not take over the host, and clarified that the only classes of operations allowed in the modified Lua code were math operations, string operations, and table manipulation. Additionally, there is a hard limit on lines of code allowed. She also addressed the question of replicas sharing data or being treated separately: the replicas are indeed treated as separate copies, but they can locate and even merge with each other, if desired. Dutch Meyer asked about variance in performance, for example, with a garbage collector, and Geambasu suggested they might use another type of language. Ben Wester (U. Michigan, Ann Arbor) asked if code gets run on every get, and Geambasu replied that in a traditional DHT, things may fail, requests get dropped, so some redundancy is a good thing.

SPORC: Group Collaboration using Untrusted Cloud Resources

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten, Princeton University

Ariel Feldman presented a system for building collaborative projects on an untrusted server, motivated by the desire to use cloud services without trusting the provider. The system presented a single server with a number of clients, moving the application state to the client, as well as a copy of state stored client-side and all server storage being encrypted. The system relies on the previously researched ideas of fork* consistency and operational transformation (OT).

Feldman took the audience through a number of common scenarios. He outlined how the fork* consistency is represented by a history hash embedded in each commit, including sequence numbers and corresponding checks when an update is pushed out to the clients. In addition, Feldman demonstrated in detail how the OT transform functions can be used to deal with merging forked groups, as well as handling pending transactions and offline commits. Lastly, Feldman talked about how access control works in SPORC: symmetric keys (for efficiency) keep the commits encrypted in transit and on the server. Access control list (ACL) changes are performed with barriers to prevent conflicting changes; all changes after a barrier are rejected until the change is fully committed. Redistribution of AES keys is done via encryption as well, preserving a chain of encrypted keys for use by members joining later. Feldman argues that at no point would the server have to be trusted. Performance evaluations on a local-area network indicate that SPORC would have usable latency and throughput for moderate-sized groups.

Bryan Ford of Yale proposed a scenario where Alice attempts to evict (malicious) Bob from the access list, but Bob adds (equally malicious) Eve and Fred before he is evicted, and the process cascades such that there are more and more malicious members of the community. Feldman explained that there are three levels of access possible in SPORC: administrators, editors, and readers, providing some assurance, although a rogue administrator could cause substantial problems. Josh Triplett probed the choice to use servers, if they are simply used for ordering and storage. Feldman replied that they allow for a more timely commit process than a decentralized system. He also verified that the project assumed a correct client. However, he pointed out that fork* recovery allows for an undo option, if necessary.

Production Networks

Summarized by Peter Bailis (pbailis@eecs.harvard.edu)

Onix: A Distributed Control Platform for Large-scale Production Networks

Teemu Koponen, Martin Casado, Natasha Gude, and Jeremy Stribling, Nicira Networks; Leon Poutievski, Min Zhu, and Rajiv Ramanathan, Google; Yuichiro Iwata, Hiroaki Inoue, and Takayuki Hama, NEC; Scott Shenker, International Computer Science Institute (ICSI) and UC Berkeley

Jeremy Stribling presented Onix, a software system for controlling large-scale networks. Onix is a cross-institution effort designed for real-world production network deployments and is currently under quality-assurance testing. Jeremy began by describing typical router architecture: a fast-path forwarding plane determines where to send packets, while a control plane inside the device can reprogram the forwarding plane and handle exceptions. Centralizing the control plane as in Software-Defined Networking (SDN) allows greater control over the network, but many of the issues related to scalable rule propagation and device reprogramming are difficult. Onix moves this centralized control logic to a distributed infrastructure while providing a high-level interface, solving state distribution problems and abstracting low-level mechanisms. Onix has several goals, including generality, scalability, reliability, simplicity, and performance.

For generality, developers program against a network graph called the Network Information Base (NIB) in which nodes are physical entities like switches, hosts, and ports. The NIB is the focal point of the system, and Onix takes care of talking to other instances, importing external state changes and exporting local state changes. Because different data has different storage requirements, it can be stored in replicated transactional storage (SQL) or in a one-hop in-memory DHT. For an ARP server, for example, the switch topology would be stored as hard state in the transactional storage, and the IP and MAC soft state would be stored in the DHT. This storage is specified pre-runtime, and at runtime the programmer only interacts with the NIB.

Scalability and reliability concerns amount to distributed state management problems. There are application-specific tradeoffs involved in determining how to partition the network. For example, in some networks Onix need not connect to every switch, which is more scalable than connecting to all of them. Traditional network scaling uses partitioning (VLAN systems) and aggregation, which reduces fidelity. Onix instead uses different instances to control different parts of the network, and network equipment connects only to a subset of instances. Onix can also aggregate data, reducing its fidelity before sharing. For example, when exporting uptime data, Onix can present an average of the information to other nodes instead of providing an exhaustive dataset. Link failures are the application's responsibility, but Onix assumes a reliable management connectivity or uses a multi-pathing protocol. Failures are handled using distributed coordination (via Zookeeper integration). Performance details are available in the paper.

Jeremy stressed that Onix is a real, production program. A prototype of over 150,000 lines of code is currently under testing and expected to be deployed in future products in the next few months or within the year.

Marco Canini from EPFL asked several questions about the application of Onix. Jeremy explained that there is only one kind of routing protocol per network, and that the Onix "application" is a program that manipulates the NIB graph to set up the routing the way that the application developer requires. There are several protocols for exchanging messages, but one easy way is using the OpenFlow protocol. Network bootstrapping depends on the particular configuration. An attendee from UCSD asked how the Onix ARP cache could be refreshed. Jeremy said that hosts sent out ARPs periodically. Eddie Kohler from UCLA asked about Jeremy's favorite implementation trick in Onix. Jeremy likes the fact that Onix provides a lot of the distributed mechanisms that are hidden from the programmer, as well as the fact that distributed storage can be swapped out transparently to the programmer.

Can the Production Network Be the Testbed?

Rob Sherwood, Deutsche Telekom Inc. R&D Lab; Glen Gibb and Kok-Kiong Yap, Stanford University; Guido Appenzeller, Big Switch Networks; Martin Casado, Nicira Networks; Nick McKeown and Guru Parulkar, Stanford University

Rob Sherwood described a new technique and prototype implementation called FlowVisor which allows realistic evaluation of new networking services within production networks. It's difficult to evaluate new services in practice. Services may not be fully ready for production, but there's a need to test them. Rob described real networking as a black-box system, with hundreds of thousands of devices, while no one really knows realistic Internet topologies. Some testbeds use software routers, but performance can suffer due to limited hardware scale, artificial topologies, and limited adoption, leading to the use of synthetic data. Subsequently, the driving motivation for this technique is to allow production networks to serve as testbeds while providing strong isolation between production and testbed networking. Rob believes that the production network can indeed be the testbed.

Rob described *network slicing,* a new technique for accomplishing this isolation. The network is partitioned into logical copies called slices, each of which controls its own packet forwarding. Users pick which slice controls their traffic, and existing production services and testbed services can run in separate slices. This enforces isolation, while allowing the logical testbed to mirror the production network topology. Network slicing can be accomplished by multiplexing the data plane, a custom, high-performance ASIC within the router that enforces rules between multiple control planes, which compute forwarding rules and push them to the data plane. The data plane is unmodified, allowing forwarding without performance penalty, while the multiple slices share the general-purpose CPU on the router.

Rob described FlowSpace and FlowVisor, an implementation of network slicing. FlowSpace maps packets to different network slices according to twelve OpenFlow header types, including IP address, MAC address, and TCP port. This allows users to opt into slices at a fine granularity, like HTTP, VoIP, or other network services. FlowVisor controls the control plane using the OpenFlow protocol, allowing external control. FlowVisor handles exceptions from the data plane and forwards them to the slice controller, which checks the policy and forwards it to the router. Handlers are cached for scalability. To keep slices from monopolizing the CPU, the system currently rate-limits rule insertions and uses periodic drop-rules to throttle exceptions, although future systems should have proper rate limiters. FlowVisor is currently deployed on or will be deployed on eight campuses and with two ISPs.

Jeff Mogul from HP Labs noted that the CPU is a limited resource and that this might limit the flow set-up rate, which Rob acknowledged. An attendee from UBC asked about the effect on latency. Rob showed that the average latency is approximately half a millisecond but depends on the operation. Another attendee asked about the requirements for forwarding memory. Rob agreed that the number of rules is a scarce commodity on current routers, but hardware manufacturers are working on expanding this. Rob mentioned that there are several techniques (e.g., caching) for operating without a complete set of forwarding memory. Finally, Josh Triplett from Portland State University asked whether the authors had considered using some slices as control slices for the system. Rob responded that in practice the experimental slices use the production slice for the control plane.

Building Extensible Networks with Rule-Based Forwarding

Lucian Popa, University of California, Berkeley, and ICSI, Berkeley; Norbert Egi, Lancaster University; Sylvia Ratnasamy, Intel Labs, Berkeley; Ion Stoica, University of California, Berkeley

Lucian Popa presented rule-based forwarding (RBF), a technique for allowing flexible Internet packet forwarding. The goal of this work was to allow more general forwarding directions, which provide routers with information on how to send their packets. This generality could provide the ability to use middleboxes, source routes, or in-network packet processing. Their thesis is that flexibility needs to be balanced by providing policy-based access, and there is a balance between flexibility and constrained access. The idea here is that forwarding directives are carried inside packets; routers only need to verify that the packet complies with the policies of all involved parties and forward the packet. Lucian argued that this allows the appropriate balance between flexibility and policy enforcement.

Lucian presented a rule-based forwarding architecture. Rules are leased from and certified by trusted third parties, and all entities involved in the rule are certified. The RBF control plane consists of both a distribution infrastructure and a certification infrastructure. RBF assumes an anti-spoofing mechanism, the existence of rule-certifying entities, and a DDoS-resistant rule distribution. Rules are represented as a sequence of if-then-else statements that are comparison operations on router state, along with several actions (e.g., drop packet); however, rules cannot modify router attributes. Thus, the rules are flexible (allowing many policies), compliant (certified), and safe (cannot modify state). In practice, rule forwarding incurs little size overhead (between 60 and 140 bytes, or 13% on standard IP packet, 27% with RSA signatures) and limited runtime overhead. They incur negligible overhead on a software router on a fast path and a 10% slowdown on the slow path, when verifying RSA signatures and using real traffic.

Michael Walfish from UT Austin asked about the feasibility of determining if a particular inter-domain path was taken. Lucian answered that cryptographic guarantees can solve this problem. Michael also asked about unnamed stakeholders in the network—how can we name rule entities in the network a priori? Lucian claimed that if we treat rule-based forwarding like an overlay network, this is not a problem; after considerable discussion, this question was taken offline. Andreas from AT&T Research asked about the effect on latency, which is negligible in the current implementation. Helen Wang from Microsoft Research asked about the deployability on today's ISPs. Lucian answered that a partial deployment is possible and might be able to detect whether some packets are or are not policy-compliant, but some packets would still be received. Rule-based forwarding enables more services, Lucian claimed, so this is still a benefit for ISPs.

Mobility

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones

William Enck, The Pennsylvania State University; Peter Gilbert, Duke University; Byung-gon Chun, Intel Labs; Landon P. Cox, Duke University; Jaeyeon Jung, Intel Labs; Patrick McDaniel, The Pennsylvania State University; Anmol N. Sheth, Intel Labs

William Enck explained how to balance fun and utility with privacy. He detailed how applications downloaded to a smartphone have full access to potentially private information, such as the GPS unit, microphone, camera, address book, and the phone's unique ID. Too often, users are surprised at what information is transmitted from the phone. In one such example, a popular wallpaper application was sending users' phone numbers back to the developer. While this particular example was found to be non-malicious, it serves as a warning that our software may be revealing information that users are not comfortable disclosing.

TaintDroid is a VM-based taint tracking system for Android phones. In order to determine when private information has left the phone, the authors modified the Dalvik interpreter to store and propagate taint tags for variables. This allows private data to be tracked, for example, from its source in the address book until it is released to the network. Despite operating in a resource-constrained environment, TaintDroid runs in real time. It displays warnings during live use of an application and was shown to have only a 14% overhead. The authors evaluated the system on 30 applications randomly selected from the 50 most popular applications in the Android Marketplace. Ultimately, 105 connections transmitted private data, while only 37 of those messages were clearly legitimate. Of the 30 studied applications, 15 shared location information with advertisement servers, and 7 shared device identifiers with a remote server. In no cases were these disclosures described in the EULA or evident from the user interface.

Iqbal Mohomed from Microsoft Research asked about implicit information flows, such as using timing perturbations to leak information to another process. Enck acknowledged that this is an existing problem, one that can be addressed with static analysis, although doing so introduces trade-offs such as high false-positive rates. Volodymyr Kuznetsv from EPFL exclaimed that the work almost made him throw away his mobile device! He encouraged the authors to provide the tool as a Web site that can test applications uploaded by concerned users. In answering other questions, Enck clarified that TaintDroid identifies when private data has left the phone, as opposed to detecting privacy violations. He also explained that many instances of private data leaking were non-malicious, so even as some application writers may attempt to circumvent the system, it may still be widely useful in the future.

StarTrack Next Generation: A Scalable Infrastructure for Track-Based Applications

Maya Haridasan, Iqbal Mohomed, Doug Terry, Chandramohan A. Thekkath, and Li Zhang, Microsoft Research Silicon Valley

Maya Haridasan presented her work on a service designed to manage paths based on GPS location coordinates. Mobile devices are now capable of recording their paths as a series of location and time tuples, which Haridasan calls a track. Once stored, these historical tracks can be a valuable source of information for users. Among many other examples, StarTrack could enable applications that provide personalized driving directions that take into account routes the driver is already familiar with. Using tracks from multiple users, ride-sharing applications could be developed.

To reach these goals, Haridasan and her team created the StarTrack service, designed to store and manage tracks and to provide a track-based programming interface. However, challenges arise in that tracks are error-prone, scalability is difficult, and applications require a flexible API. To address these concerns, Haridasan and her team developed a set of novel algorithms and data structures to compactly store, compare, join, and sort sets of tracks. After evaluating StarTrack against several other potential implementations, she closed by asking anyone interested in using the infrastructure to contact the authors.

Petros Maniatis from Intel Labs proposed a class of queries where information from multiple clients must be aggregated but not revealed. For example, the request "give me walking directions that don't intersect with my ex-partner" requires some knowledge of another's location, but ideally in a way that doesn't share private data. Haridasan and her team had considered similar use cases, but haven't found a solution yet. Michael Nowlan of Yale asked if there was any intention to associate intent or content with a track. Such a pairing could be used to separate driving tracks from other modes of transportation or to keep some paths private. Haridasan explained that there was already metadata associated with tracks, and ACLs could be used to preserve privacy, so such designs should already be possible. Stefan Sariou of Microsoft Research asked about handling errors and incorrect data. Haridasan replied that much of this was handled by the GPS location canonicalization algorithm, which converts a set of collected GPS samples into a path that goes through the underlying road network.

Virtualization

Summarized by Alan Dunn (adunn@cs.utexas.edu)

The Turtles Project: Design and Implementation of Nested Virtualization

Muli Ben-Yehuda, IBM Research—Haifa; Michael D. Day, IBM Linux Technology Center; Zvi Dubitzky, Michael Factor, Nadav Har'El, and Abel Gordon, IBM Research—Haifa; Anthony Liguori, IBM Linux Technology Center; Orit Wasserman and Ben-Ami Yassour, IBM Research—Haifa

Awarded Jay Lepreau Best Paper!

Muli Ben-Yehuda presented the Turtles Project, which was awarded one of two Jay Lepreau Best Paper awards for OSDI '10. The Turtles Project is an implementation of nested virtualization support for the Intel x86 architecture, which means it allows software written to use virtualization hardware support to run inside a hypervisor that already uses that hardware. Nested virtualization support allows for new applications, like hardware virtualization support for OSes that already are hypervisors (e.g., Windows 7 with XP mode) and deployment of virtual machines in the cloud. The x86 architecture supports only one level of virtualization in hardware natively, so multiplexing the virtualization hardware is necessary. The difficulty is to perform this efficiently, as changing the state used by the virtualization hardware requires expensive VM exits, and multiplicatively more exits per level of nesting.

Ben-Yehuda focused on MMU and I/O virtualization efficiency improvements. For MMU virtualization, he described three schemes in increasing order of efficiency. The key problem is that even with extra Extended Page Table (EPT) hardware support for more efficient translation, with any nesting depth greater than one there will be more translations necessary than hardware can provide, so translations must be compressed into fewer mappings. The most efficient "multi-dimensional" paging scheme compresses EPT mappings, since they change less frequently. For I/O virtualization efficiency, Ben-Yehuda described the most difficult case they had to tackle: the direct assignment of devices to nested VMs. Direct assignment requires an IOMMU for safety, but for nesting there is added difficulty, since the IOMMU itself must be emulated and requires analogous mapping compression.

The most important take-away measurements were approximately 10% CPU overhead per level of nesting, that multi-dimensional paging can be a several hundred percent performance win for page-fault heavy loads, and that multi-level device assignment can obtain equal device performance but at significant CPU cost. A significant portion of the added cost for multi-level device assignment could be removed if direct interrupt delivery to nested VMs was supported. The code was mature and efficient enough to be added to KVM.

An audience member from NICTA asked whether the performance would continue to get 10% worse per level of nesting. Ben-Yehuda responded that the exit multiplication effect would get worse as the level of nesting increased, and thus nesting performance would generally get worse by more than 10% per level with higher degrees of nesting. Nathan Taylor of the University of British Columbia asked about the security implications of this work, whether this would facilitate VM-based rootkits and whether I/O would remain safe with this implementation. Ben-Yehuda noted that OSes can easily detect when they are being virtualized (via timing irregularities, for example), so that VM-based rootkits are no less detectable with this work. He also said that trusted computing technology could potentially be employed to ensure the underlying hypervisor boots from a known safe environment. Sorav Bansal of IIT Delhi asked whether binary translation could help further reduce overhead, and Ben-Yehuda said this was possible but difficult for nested virtualization, due to lack of knowledge of hypervisor memory layout.

mClock: Handling Throughput Variability for Hypervisor IO Scheduling

Ajay Gulati, VMware Inc.; Arif Merchant, HP Labs; Peter J. Varman, Rice University

Ajay Gulati presented mClock, a new algorithm for scheduling I/O requests from VMs. He pointed out that controls on CPU allocation for VMs are fairly mature by now, but that there aren't equivalent controls for I/O requests (IOPS). In particular, reservation and limit controls of a fixed number of IOPS/time are not available, which is problematic in that storage is often from a shared network device, causing variable total capacity for all VMs on a host and making pure proportional sharing inappropriate.

Scheduling algorithms for VMs are often phrased in terms of time tags, in which the VM with the minimum time tag is scheduled. Gulati explained that the key points of mClock were real-time tags and separate time tags for reservations, limits, and proportional shares of resources beyond reservations. Real-time tags are used to ensure that the procedure can track actual rates of IOPS/time. Tags are prioritized so that VMs not making their reservations take precedence and that VMs exceeding limits are not scheduled. The effectiveness of mClock in maintaining limits and reservations was demonstrated empirically with a graphical side-by-side throughput comparison. A separate enhancement allowing VMs to gain credit for being idle was made, which is important as I/O traffic is often bursty. mClock is general enough to be employed for other resources (such as network I/O) as well.

Etienne Le Sueur from NICTA pointed out that it appeared that the total IOPS/time dropped under mClock. Gulati said that the workloads of VMs in their experiments have different read-write ratios, degrees of randomness, and I/O sizes, so that the degree of variance Le Sueur observed (several hundred IOPS) was not unexpected. Gulati pointed to workload details on a backup slide that is reproduced in the paper. Another audience member asked how mClock balances latency and throughput tradeoffs. Gulati responded that mClock is more about dividing IOPS among VMs with reservation, limit, and share controls, but that prior VMWare work (PARDA) in FAST '09 dealt with latency control by throttling hosts. He said that for stronger guarantees, ideally one would have underlying hardware guarantees from vendors.

Virtualize Everything but Time

Timothy Broomhead, Laurence Cremean, Julien Ridoux, and Darryl Veitch, Center for Ultra-Broadband Information Networks (CUBIN), The University of Melbourne

Darryl Veitch presented an architecture for tracking time more accurately in the Xen VMM. The motivation for this work is that there are a number of applications—including finance, network monitoring, and distributed gaming—that require accurate timing information, but current timekeeping methods produce inaccurate results in VMs. Factors like clock drift combine with variable latency caused by other VMs in ways that can cause feedback mechanisms in ntpd, the current de facto standard for Linux timekeeping, to become unstable. Additionally, it is difficult to preserve correct times during live migration of VMs.

Veitch described his group's prior work with the Robust Absolute and Difference (RAD) clock, and he explained why it is a good fit for Xen. The key design point of RADclock is that it is "feedforward"-based. This means that system clock timestamps, which already have corrections applied, are not used to timestamp timing packets. Instead, raw counter values are used, and clock calibration is achieved through variables that convert raw counters into real times. VM clocks can then all read one hardware counter and calibration variables hosted in Dom0, which has hardware access; this is referred to as a dependent clock design. VM migration becomes easier in this design, since hardware counter values can be converted into new system clock times merely by using the calibration variables of Dom0 on the new machine rather than transplanting feedback-related state calibrated on one machine onto another whose counters have origins, drift, and temperature environment that are completely different.

The RADclock-based design was tested against Xen's current and prior timekeeping mechanisms. Veitch said that the measured error in standard RADclock operation appears to be low enough that the primary source appears to be air-conditioning flow in the room. Also, migration caused inaccuracy of the order of tens of microseconds in RADclock, as opposed to several seconds with current Xen timekeeping.

David Cock of NICTA asked whether a dependent clock design like RADclock could be used with feedback-based mechanisms without compensating clocks, as in Xen's current timekeeping. Veitch claimed that this would be difficult. Another audience member wanted clarification on what accuracies are achievable with RADclock. Veitch said that this depends primarily on characteristics of the connection to a time server. He pointed to his prior work showing that many kinds of load (e.g., high temperature) are not problematic, and estimated maximal accuracy in the range of tens of microseconds.

Workshop on Supporting Diversity in Systems Research (Diversity '10)

October 2–3, 2010
Vancouver, BC, Canada

Summarized by James Mickens (mickens@microsoft.com)

Research Agendas: Picking Good Ones and Publishing Successfully

Dawn Song, University of California, Berkeley; Anthony Joseph, Intel and University of California, Berkeley

Dawn Song, a professor at Berkeley, provided several pieces of advice for students who were struggling to publish or pick compelling project ideas. First, she observed that many students only read papers in their particular subfield of computer science. Song recommended that students read a broad range of literature in computer science and beyond; by doing so, they will learn about important problems and techniques in other areas which may relate to their own subfield. In a similar vein, Song advised students to communicate frequently with professors and other students, both through formal channels like reading groups and informal channels like lunch meetings. Such face-to-face interaction provides students with valuable networking opportunities and also exposes them to a wider variety of perspectives on computing research. Finally, Song also advised students to not treat class projects as busywork, but as opportunities to create publishable research.

The next presenter was Anthony Joseph, a professor at Berkeley and director of Intel Research Berkeley. Joseph encouraged students to do industrial internships and gain exposure to pressing real-world problems. However, Joseph also advised students to avoid working on problems that are too short-term or whose solutions only require an incremental improvement to the current state of the art. Joseph described his personal research methodology, explaining how he decomposes large projects into separate, short projects so as to get early results and quickly determine whether the overarching goals are actually as interesting as they originally seemed. Joseph said that with the advent of utility cloud infrastructures such as Amazon's EC2, even graduate students can evaluate their system's behavior at a large scale. Joseph recommended the use of these platforms to generate realistic evaluation results, and he encouraged students to take a statistics class so that they can properly validate their raw data. Joseph also emphasized that the research community will count the number of projects you finish, not the number you start. Thus, it is important to prefer simple solutions to complex ones, gather external feedback during a project's life cycle, and be flexible in adapting a project's goals and scope.

Getting the Most from a Conference

Carla Ellis, Duke University; Yvonne Coady, University of Victoria

Carla Ellis and Yvonne Coady gave an interactive presentation which taught students how to derive the maximum benefit from a conference. The presentation began with a skit in which Ellis and Coady pretended to be shy graduate students who were too intimidated to talk to a well-known researcher. The pair used the skit as a launching point for a discussion about how to successfully network. Ellis described the importance of having an elevator pitch which succinctly describes your research problem and your proposed solution. Coady observed that most academics love to talk about their own research, so students should not be afraid to initiate conversations with more established members of the community. Throughout the presentation, audience members asked questions or provided additional advice. Jonathan Appavoo, a professor at Boston University, reminded students of the importance of sincerity, and said that whether you talk to a first-year graduate student or a senior professor, you should always treat people with respect and kindness. James Mickens of Microsoft Research said that the academic community is actually quite small, but that attending conferences will never be fun until you know other people within the community. Thus, he encouraged students to network widely; for students who are shy, Mickens recommended that they talk to other students and build confidence before talking to more senior members of the community.

Hot Topics in Systems Research

Monica Lam, Stanford University; James Mickens, Microsoft Research

Monica Lam gave the first of two purely technical presentations in the Diversity workshop. She began with an overview of her research focuses, which have included compilers, software bug detection, system management, and security. Lam spent the rest of her talk describing her most recent research, which focuses on devising social networking applications that are decentralized, easy to use, and do not force all users to entrust all of their data to a few large companies. Lam observed that Facebook is headed towards a monopoly on personal information similar to the monopoly that Microsoft and Intel have over the desktop computing platform. The key challenge in creating a privacy-preserving alternative for social networking is that Facebook already has an enormous user base, and these people will be loath to move to a new privacy-preserving system, both for reasons of convenience and because many users do not treat privacy leakage as a first-order concern. Thus, Lam's research goal is to allow users to interact socially using an open, federated standard that lacks a central authority.

Lam gave several concrete examples of this architecture. In her Partyware project, people within immediate physical proximity share their profiles using their mobile phones, creating an ad hoc social network only consisting of people in a certain place at a certain time; individual users or supportive businesses like coffee shops run a lightweight rendezvous server that forwards traffic between phones or other user devices. Lam also described how email can be used as the transport protocol for decentralized social networking applications. Email is an attractive medium for several reasons. First, email providers typically provide much stronger privacy policies than those provided by social networking companies. Second, users can create new online personas simply by creating new email addresses. Finally, since there are multiple email providers (Google, Microsoft, Yahoo, etc.), no one party can control the aggregate social graph.

James Mickens began the second technical presentation by describing the fundamental insight behind his recent research: using JavaScript, Web pages have sufficient computational abilities to act as heavyweight participants in distributed systems. Mickens then described two projects that leverage the power of Web pages running within unmodified browsers. The first project, named Silo, exploits two insights to reduce the load time for a Web page. First, a Silo Web server aggressively inlines the textual objects in a Web page (e.g., HTML, CSS, and JavaScript), reducing the number of fetches a browser needs to collect the objects. Second, the server and the Web page running on the client use JavaScript and AJAX to engage in a custom delta-encoding protocol similar to that of the LBFS distributed file system. Thus, the inlining reduces the number of round trips needed to build the page, and the delta-encoding allows the browser to cache data for the inlined objects even though the page no longer references them using explicit URLs. The resulting protocol can reduce end-to-end load times for some Web pages by over 50%.

After describing Silo, Mickens provided an overview of Mugshot, a tool for recording and then replaying the execution of JavaScript-based Web applications. Mugshot leverages JavaScript's extensive facilities for reflection to introspect upon all of the nondeterminism within a Web page (e.g., GUI activity, the reception of AJAX data, random number generation, etc.); this introspection works using standard JavaScript running on unmodified browsers. If the user encounters a problem with a Web page, she can opt to send her event log to a developer, who can then replay the events, recreating the session and stopping the event flow at any point to examine the page's dynamic state in a debugger.

Mickens concluded his talk by posing several open-ended questions about the future of Web research. First, he asked whether the standard Web stack (HTTP, HTML, and JavaScript) was becoming ossified like TCP/IP, and whether this discouraged exciting yet disruptive research that would break backwards compatibility. Mickens also asked whether the same domain policy should be revamped, and what kinds of fundamental programming abstractions are needed to expose cloud servers to Web pages.

Graduate School: The Things I Wish I'd Known

Lakshmi Ganesh, Cornell University; Nalini Belaramani, Google; Susan Horowitz, University of Wisconsin—Madison

This session contained presentations from three people at different points in their careers. The first presenter was Lakshmi Ganesh, a fifth-year graduate student at Cornell University. She described a variety of factors to consider when a student must pick an advisor. A good rapport is obviously critical, but Ganesh explained that a professor's level of funding is also important to consider, since working for a professor with few grants may force a graduate student to become a teaching assistant or a grader, resulting in less time for research and a longer time to graduation. Ganesh also advised students to consider a potential advisor's tenure status. Younger professors are often more motivated to publish frequently, but may micromanage. In contrast, tenured professors may be more relaxed and more interested in deeper, more impactful research, but they may not push students to work as hard as they can, and students may feel uncomfortable betting the early part of their career on high-risk, high-reward research.

The next presentation was given by Nalini Belaramani, who graduated from the University of Texas at Austin in 2009 and now works as a software engineer at Google. Belaramani emphasized the importance of communication skills, saying that the ability to explain one's work in a confident, articulate manner is crucial for getting papers published and succeeding in the job market. She encouraged students to practice writing and presenting as often as possible. In particular, Belaramani said that successful graduate students excel at motivating their work to others, either through the introduction section of a paper or the first few minutes of a verbal presentation. Seeking early feedback from professors or other students may result in criticism, but it will ultimately improve your research skills and your presentation abilities. Belaramani also mentioned that students should not bet their entire career on a single paper getting published in a specific venue; instead, students should work on multiple projects and multiple papers, and be willing to adapt the scope of projects in reaction to paper reviews or newly discovered knowledge.

The final presentation was given by Susan Horwitz, who received her PhD in 1985 and is now a professor at the University of Wisconsin—Madison. Horwitz told the student attendees to expect occasional hardships during the PhD process, but to take solace in the fact that everyone will experience challenges during graduate school. Horwitz encouraged students to be social and have fun during their graduate studies, and to converse with other students and faculty during the inevitable periods of feeling lost or unmotivated. Horwitz also told the students that if they were considering a job at a research university, they should prepare not just to do research but to teach, write grants, and review papers. Horwitz encouraged students to experience all of these tasks while in school so that they could determine whether a job at a research university would be a good career path for them.

Finishing the Dissertation: Techniques for Success

Anne Rogers, University of Chicago; Jinyang Li, New York University

The session began with a presentation from Anne Rogers, a professor at the University of Chicago. Rogers' talk was grounded in her experience as the director of graduate studies for her school's computer science department; this role gave her unique insights into the pitfalls that students often encounter as they try to complete their dissertations. Rogers said that many students initially think of their dissertation as a movie, but it should really be an excellent short story—articulate and impactful, but no longer than necessary. Rogers emphasized that continual dialog between a student and her committee is crucial for ensuring that the dissertation is finished on time and with minimal revisions. Rogers also stressed the usefulness of communication with people outside the dissertation committee. By exchanging ideas with other students, visiting researchers, or people in completely unrelated fields, students can remain intellectually stimulated and get crucial feedback on their thesis work.

Jinyang Li, a professor at New York University, advised students not to worry too much about the thesis. Instead, students should focus on devising good projects and publishing papers about those projects. Once a student has two or three strong papers, a thesis will often naturally emerge. Li observed that individual publications will be read by your peers much more often than your thesis, so students should not agonize over creating a perfectly polished thesis. However, Li said that the thesis provides an excellent opportunity for poor writers to focus on their prose. In particular, writing the introduction for the thesis provides good practice in the art of selling your work to the larger academic community, a skill which is invaluable for writing grants and giving public presentations.

Job Choices: Academia versus Industry

Jonathan Appavoo, Boston University; Cary Gray, Wheaton College; John Wilkes, Google

Jonathan Appavoo described his personal career path from graduate student at the University of Toronto, to researcher at IBM Watson, to his current post as professor at Boston University. Appavoo said that a key factor in deciding where to work is the quality of the people who work there, not just in terms of their intellectual caliber but in terms of whether they create a friendly and productive workplace environment. Appavoo also emphasized that wherever you work, it is extremely important to be passionate about what you do. People who are energized by working with students may not thrive in an industrial environment; similarly, people who like to have impact on ready-to-ship projects may become frustrated with a university job. Appavoo said that he eventually transitioned from industry to academia because he felt that he had more freedom to explore his research agenda without regard to whether that research immediately impacted a company's revenue.

Cary Gray from Wheaton College provided another perspective from academia; however, in contrast to Appavoo, who worked at a large research university, Gray worked at a smaller academic institution which focused on undergraduate education. Gray said that he loved working at such an institution because it allowed him to form deep relationships with students and have a direct impact on their intellectual growth. However, Gray said that undergraduate-focused institutions are not suitable for people who do not like to teach, since these institutions require professors to teach two, three, or sometimes four classes per semester. Gray also mentioned that many teaching-focused institutions are in smaller cities, which may or may not be an advantage, depending on one's affinity for the big-city lifestyle.

The session concluded with a talk from John Wilkes, an industrial researcher who worked at HP Labs for 25 years before moving to Google in 2008. Wilkes said he enjoyed industrial research because it continually introduced him to interesting real-world problems whose solutions could immediately impact millions of people. Like the prior two speakers, Wilkes emphasized the importance of being passionate about what you do, and he encouraged the workshop's student attendees to think carefully about what really excited them. Wilkes advised students to do industrial internships to gain experience with the different workflow in that environment. Wilkes also encouraged students to take on bold projects during these internships, since impressing people during an internship can lead to fruitful collaboration or even a future job offer.

Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML '10)

October 3, 2010
Vancouver, BC, Canada

Invited Talk

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

QPS, KW-hr, MTBF, ΔT, PUE, IOPS, DB/RH, . . . : A Day in the Life of a Datacenter Infrastructure Architect

Kushagra Vaid, Microsoft

Kushagra Vaid, principal datacenter infrastructure architect at Microsoft, presented this talk on challenges in datacenter design. Data centers are complicated, and datacenter design needs to take into account datacenter and server architecture, platform architecture, and reliability analysis.

Challenges in datacenter architecture include finding ways to optimize power distribution and cost efficiency. The metric of interest for the former is power utilization efficiency (PUE), computed as total facility power/IT equipment power. Typical industry averages range between 1.5 and 2.0. One common design choice that affects power distribution efficiency is whether to propagate AC all the way to individual machines' power supply units or to convert to DC at the entry point to the data center. Vaid showed that DC configurations are more efficient at low loads and that AC configs prevail at higher loads, but that at highest efficiency both configurations are within 1–2% of each other. With regard to cost efficiency, Vaid showed how the scale of modularization has increased over time within Microsoft's data centers so as to increase this metric.

There are several challenges in the platform architecture area; for example, determining how to analyze workloads to find their optimal CPU requirements (frequency, number of cores, etc.) and determining whether it is worthwhile to pursue new hardware technologies (e.g., replacing desktop CPUs in datacenters with mobile CPUs or replacing hard drives with SSDs). With regard to the latter, Vaid showed that overall TCO for mobile processors, such as Atom, is 2.5x worse than regular desktop CPUs for both performance per dollar and performance per watt. Future Atom CPUs should either provide much better performance or much lower power in order to be considered feasible alternatives.

For reliability analysis, the main challenges involve determining how MTBF corresponds to environmental operating ranges. For example, Vaid showed how hard drive failure rates increase with temperature.

At the end the talk, Vaid made the case that finding an optimal solution for all of the areas together is essentially a multi-dimensional optimization problem, for which data-mining techniques and machine learning are required. Erik Riedel asked whether Vaid knew the distribution of hard drive failure modes with temperature. Vaid replied that the statistics collected were an aggregate and he did not know the breakdown. Has Microsoft considered releasing datacenter traces to researchers, so that they can investigate techniques for optimization? Microsoft already has released traces of datacenter workloads. Is there anything that keeps current ML tools from being useful for the problems presented by the author? Data formatting is a problem—the logs that contain the information ML tools need aren't in standard formats, making it difficult to use them.

Refereed Paper

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

Creating the Knowledge about IT Events

Gilad Barash, Ira Cohen, Eli Mordechai, Carl Staelin, and Rafael Dakar, HP-Labs Israel

Gilad Barash from HP Labs presented research about a tool for answering user queries about IT events. Instead of spending time searching Web forums for answers to questions such as, "Why does my HP printer driver always return error message X?" users can pose their questions to the tool directly; the tool returns a ranked list of possible answers by either directly scraping IT Web forums or looking up the answer in a pre-computed knowledge database. The tool is composed of four distinct components—a search composer, a searcher, a ranker, and a knowledge database. The search composer simply creates progressively more generic search terms from the user input. The searcher queries standard search engines (e.g., Google) using the search terms and stops when a pre-determined number of results have been returned. The ranker is faced with the challenge of creating a better ordering than that returned by the search engine, by using domain and content-specific information. The knowledge database simply stores the results of previously stored queries.

The ranker uses three metrics to rank results: the source rank, the quality of information of each result, and the relevancy of each result. The source rank is simply computed by Web domain—if a search query is about HP printers, results from HP Web forums will be given a higher source rank than those from IBM Web forums. The overall quality of information (QOI) of each result is computed by combining several indicators of QOI—whether or not the relevant forum thread is marked as "question answered" or "not answered," the date the thread was last modified, the number of replies to the original poster, etc. In computing the QOI, the authors of the paper found that an important indicator of this value—whether or not the thread is marked as answered—tends to be noisy. That is, users often forget to update the label of a thread containing a valid answer from "not answered" to "answered." To deal with this problem, the authors developed a method for learning whether a thread contains a valid answer from the other QOI indicators. Finally, the relevancy of a result is computed by simple distance measures (e.g., string-edit distance) that compute the closeness of the search terms to words that appear in the result.

Barash concluded by saying that a prototype of the system has been implemented. An audience member asked how the tool's ranked results compared with just raw results from searching on Google. Barash stated that the ranks his tool yielded were often better than Google's results, because it takes into account domain-specific information, such as QOI scores. A concern raised was whether this conclusion was based on just the snippet of information returned by Google with each result, or whether it was based on looking at the actual documents. Barash stated that they had looked at the actual documents. Another audience member commented that this tool seemed great as long as the nmber of people using it were small compared to those creating data by posting on Web forums. He then asked whether it was possible to extend the tool so that it could feed back information about the most relevant results to the Web forums it scraped for data. Barash said that such feedback is something that they're thinking about. For example, he said the tool could add tags to "strengthen" specific posts that it has computed are very useful. It could also automatically answer new questions.

Logging Design and Visualization

Synoptic: Summarizing System Logs with Refinement

Sigurd Schneider, Saarland University; Ivan Beschastnikh, University of Washington; Slava Chernyak, Google, Inc.; Michael D. Ernst and Yuriy Brun, University of Washington

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

Ivan Beschastnikh, a student at the University of Washington, presented his research on algorithms for generating concise graph-based summaries of system log information. Ivan presented arguments for refining graphs of systems logs as opposed to coarsening. The former is the process of starting with a very coarse-grained graph in which related events are merged into single nodes and iteratively splitting them until a list of invariants is met. The latter is the process of starting out with a fine-grained graph and merging nodes until some invariant is violated. It is easier to satisfy important invariants by refinement than by coarsening, but refinement can create graphs that are too constrained. For such cases, combining refinement with coarsening can yield less constrained graphs that still satisfy all invariants.

The hybrid algorithm presented, called BisimH, starts by creating a graph of system-log events based on invariants specified by the user. Events that can be grouped together without violating the invariants are merged into partitions and depicted as nodes in the graph; edges depict dependencies between partitions.

BisimH then mines the system logs for additional invariants and uses them to generate examples that the current graph allows, but which the invariants do not. It then iteratively re-partitions the graph so as to satisfy the mined invariants. The problem of finding the most general graph that satisfies all of the invariants is NP-hard; as such, BisimH uses heuristics to explore the search space, often resulting in graphs that are too strict or too refined. As such, after each iterative refinement of the graph BisimH uses coarsening algorithms to determine whether a slightly coarser graph would satisfy the same invariants.

Ivan concluded by presenting case studies in which the BisimH algorithm, implemented in Synoptic, is used to understand the behavior of two protocols: the Peterson leader election algorithm and reverse traceroute. For both, he showed that BisimH yielded more accurate summaries than kTail, a popular coarsening algorithm. He also showed that BisimH was faster than coarsening with invariants.

An audience member asked Ivan to clarify the size of the logs used in his study and the processing overhead. He used only one machine for the case studies presented. He said that runtime was exponential in log events, but then changed his mind and said that it probably wasn't exponential.

A Graphical Representation for Identifier Structure in Logs

Ariel Rabkin and Wei Xu, University of California, Berkeley; Avani Wildani, University of California, Santa Cruz; Armando Fox, David Patterson, and Randy Katz, University of California at Berkeley

Summarized by Peter Hornyack (pjh@cs.washington.edu)

Ariel Rabkin presented a new system to uncover flaws in the coverage and consistency of application console logs. This work was motivated by the fact that while log messages are a primary tool for debugging applications, analysis of these logs is often hampered by missing, incorrect, or inconsistent information. The goal of the system is to improve future log analysis by visualizing the log message types and identifier fields in a way that makes common flaws easily visible and facilitates comparison across logs.

The system analyzes application logs offline and visualizes several aspects of them in the form of a graph. The nodes in the graph represent either message types or identifiers such as transaction IDs. Edges in the graph indicate the identifiers that appeared in messages of certain types; other information, such as the relative frequency of message types, is also visualized in the graph. Rabkin presented several example visualizations and pointed out the flaws they reveal; for example, missing identifier errors are easily seen as messages in the graph without edges to any identifiers. Another example showed that inconsistency in the identifiers used across multiple message types appears in the graph as an identifier connected to only a single message. Finally, Rabkin showed how the system can be used to identify logging errors by visually comparing the resulting graphs of logs from production systems.

One audience member noted that the example graphs were for the most part planar, and wondered if the authors had produced any graphs that were too messy to visualize easily. Rabkin replied that for most programs, the set of identifiers and message types is not that large and that manageable graphs usually result, and also emphasized that some of the example graphs were from large real systems, such as production Hadoop clusters. Another audience member asked how difficult it is to find the corresponding bug in the code when a flaw is observed in the visualization. Rabkin answered that the process is usually straightforward, since the origin of each message type is usually easy to find in the code, and noted that the time spent fixing these logging bugs pays for itself by enabling better error detection and debugging using the logs in the future.

SIP CLF: A Common Log Format (CLF) for the Session Initiation Protocol (SIP)

Vijay K. Gurbani, Bell Laboratories/Alcatel-Lucent; Eric Burger, Georgetown University; Carol Davids and Tricha Anjali, Illinois Institute of Technology

Summarized by Peter Hornyack (pjh@cs.washington.edu)

Vijay Gurbani presented work on the development of a common log format for SIP. Most enterprises have SIP servers and clients for IP telephony and other uses, but these are often obtained from multiple vendors, each of which uses a different log format today. The authors argue that a CLF for SIP is needed to allow trend analysis and anomaly detection across equipment from multiple vendors, and to encourage the development of third-party tools for troubleshooting SIP. The success of the HTTP CLF in these respects and some recent publications on the complexity of SIP parsing were presented as support for a SIP CLF.

Gurbani presented some background information on SIP, then described the HTTP CLF and pointed out the myriad differences between the SIP and HTTP protocols that make defining a CLF for SIP more challenging than for HTTP. For example, unlike HTTP, SIP is not a linear protocol with exactly one reply for every request. The need for multiple responses per request and the potential for long delays between requests and responses in SIP increase the complexity of the state that must be recorded in a SIP CLF. Gurbani presented the work done to create a canonical record format, with an emphasis on its extensibility and the ease with which it can be parsed. He then showed some complex SIP flows and demonstrated how simple grep commands could be used to perform useful queries on logs in the canonical format. The IETF is currently in the process of standardizing the SIP CLF proposed by the authors.

Many audience members wondered why there isn't already a CLF for SIP, despite it being in use for many years. Gurbani replied that this is not unexpected, since the primary focus of the IETF has been to stabilize the protocol (SIP) itself and reduce ambiguities in the specification, and accoutrements such as logging are being looked at now. Furthermore, unlike the dominance of Apache in HTTP which fostered an HTTP CLF, there isn't an open source equivalent in SIP to do the same for a SIP CLF. Have the authors investigated how logs in the proposed format could be transformed into graphs or other useful structures for visualizing the log contents? So far they have been concerned strictly with getting the syntactic specification of the SIP CLF finished in the IETF, and uses such as visualization tools, graphs, correlation engines, and others will be much easier to develop with a canonical format in place.

Applications of Log Analysis

Summarized by Peter Hornyack (pjh@cs.washington.edu)

Experience Mining Google's Production Console Logs

Wei Xu, University of California at Berkeley; Ling Huang, Intel Labs Berkeley; Armando Fox, David Patterson, and Michael Jordan, University of California at Berkeley

Wei Xu presented early findings from his group's investigation of console logs from Google production systems, beginning with some of the challenges in using console logs in large systems: they are usually stored with just best-effort retention, log messages are often generated ad hoc using custom logging libraries, and new message types are constantly introduced. The data set mined by the authors was produced by systems consisting of thousands of nodes, with five orders of magnitude more messages and many more message types than they used in their previous work.

One problem that the authors focused on is using global system state, rather than local node state, to detect problems in the system. By applying machine learning techniques for anomaly detection, the authors were able to find features in the log messages that correlated with system alarms. The authors also used sequence-based detection to identify problems on single nodes in the system. Finally, Xu presented techniques that the authors used for removing sensitive information from data gathered on production systems, a process termed "log sanitization." The authors' evaluation of their log mining techniques demonstrates that the techniques scale and apply to the extremely large production data set.

Analyzing Web Logs to Detect User-Visible Failures

Wanchun Li, Georgia Institute of Technology; Ian Gorton, Pacific Northwest National Laboratory

Most Web applications suffer from unreliability, which causes downtime and transaction errors that directly impact users. Wanchun Li pointed out that early failure detection can mitigate later failures, but detection itself is difficult to perform in complex Web apps, and existing automated techniques are often ineffective.

Li presented a system that detects failures that users experience when using Web applications. The authors' approach to detecting these failures is based on the principle that when users experience failures, they will respond to the failure in a way that breaks from the usual navigation paths in a Web app. The system models a Web app as a graph with pages as nodes and users' navigation as edges, then uses a trained Markov model to estimate the probability of a given navigation path. If the computed probability of a navigation path is less than some threshold, then the system raises a failure alarm, indicating that the anomalous path may have been the result of the user experiencing some failure. The authors evaluated the system using an access log of HTTP requests from NASA's Web site, and found that at the optimal balance point between detection rate and false-positive rate, the system correctly detected 71% of failures with 26% false positives.

One audience member asked if the system would have to construct and train a completely new model when the Web application is modified. Li replied that the graph is constructed incrementally and can adapt to changes in the set of pages and navigation paths. Since even the least popular pages on large Web sites are visited regularly, would the system classify these visits to the least popular pages as failures? With sufficient training data, the visits to even the least popular pages and the typical navigation paths to them would be captured by the model, and only navigations that don't follow some typical path would be marked as failures.

Industry Track—Experiences

Summarized by Ivan Beschastnikh (ivan@cs.washington.edu)

Optimizing Data Analysis with a Semi-structured Time Series Database

Ledion Bitincka, Archana Ganapathi, Stephen Sorkin, and Steve Zhang, Splunk Inc.

The workshop's industry track featured two papers, both of which focused on Splunk, a platform for collecting, searching, and analyzing time series data from many sources with possibly varying formats. Archana Ganapathi presented the first paper, and gave an overview of how Splunk works.

Archana quoted Joe Hellerstein's statement that we are in the "industrial revolution of data." Managing the growing explosion of data is a key challenge, and one of the most difficult aspects of big data is enabling efficient analysis. After all, what's the point of storing it all if there is no means of extracting valuable insights? The Splunk platform enables one to search, report, monitor, and analyze streaming and historical data from a variety of sources—for example, in an IT infrastructure: logs, configurations, messages, traps and alerts, scripts, custom code, and more. As Archana stated, "If a machine can generate it, Splunk can eat it."

Splunk's design uses three tiers: (1) forwarders collect data and send it to the indexers; (2) indexers denormalize the data, attach keywords, and of course index the data; (3) a search head provides a single query point to which users can then submit queries. These queries are converted into MapReduce jobs which run across the indexers. Because co-temporality is crucial to many queries, Splunk uses a modified MapReduce hashing function to map data from the same time-window onto the same machine.

One of Splunk's important features is its streaming indexing system, which enables real-time search results. However, possibly the most attractive feature in Splunk is its advanced query language. Expressions in this language eventually compile down to MapReduce jobs, but the user is relieved from thinking about the map and reduce functions—the conversion is entirely transparent. Because Splunk does not use data schemas, the query language supports searches over heterogeneous and constantly evolving formats. Combine operators, which are essentially UNIX pipes, are used to string multiple expressions into complex programs. Archana gave detailed examples of uses of this query language for outlier detection, clustering, and data munging (combining data from multiple sources and with different formats).

In the Q&A, a few of the audience members wanted to gain a more detailed understanding of how Splunk works. For example, the presentation did not describe how the various features of the language were realized in the MapReduce framework. These questions were taken offline.

Bridging the Gaps: Joining Information Sources with Splunk

Jon Stearley, Sophia Corwell, and Ken Lord, Sandia National Laboratories

The second presentation in the workshop's industry track was presented by Jon Stearley, who discussed his experiences with using Splunk to make sense of supercomputer logs. Jon set the stage by describing Sisyphus, a tool he developed to collate unstructured logs from across many systems. The many features of Splunk, however, convinced Jon to try the new system. In particular, he found Splunk to be robust in dealing with logs lacking a well-defined schema, and that Splunk's built-in support for collaborative log exploration made it easy to involve many people in the process and to capture and share knowledge.

During the bulk of his presentation, Jon focused on a few Splunk features. One of these is Splunk's ability to treat unstructured input logs as relational entities. This allows one to, for example, compose queries that join logs across log fields. Another feature is Splunk's ability to associate event types with those messages that satisfy at least one query pattern in a set (or some other constraint) and to write queries over event types. Yet another feature is subqueries, which provide a powerful composition mechanism. On top of a complex log analysis feature set, Splunk makes it easy to document, save, and share queries. This captures communal log knowledge and allows more people to participate in log analysis tasks. As a summary of what Splunk is capable of, Jon emphasized that Splunk "takes care of all the ugly preprocessing" that log analysis typically involves.

In the Q&A an attendee asked how Splunk deals with logs that have unsynchronized clocks. Jon hadn't dealt with such logs before and therefore couldn't comment. Is Splunk difficult to learn? Splunk is simple, and many features of its query language have UNIX command-line analogs. How well does Splunk integrate external analysis tools? It is straightforward to plug other tools into Splunk.

Panel Discussion

Summarized by Ivan Beschastnikh (ivan@cs.washington.edu)

John Hammond, University of Texas at Austin; Jon Stearley, Sandia National Laboratories; Eric Fitzgerald, Microsoft

The SLAML panel discussion revolved around many outstanding issues in the log mining and analysis research community. It also touched on how these issues relate to the challenges faced by industry.

The first theme to generate interesting discussion was log completeness. Eric Fitzgerald pointed to the incompleteness and a lack of standardization of existing logging formats as a major problem for log analysis and tool interoperability. Eric described and advocated that the community pay attention to the Common Event Expression (CEE) effort. This effort aims to define a standard for event interoperability across a wide range of logging sources and establishes which events must be raised when, and what fields an event of a particular type must include.

The CEE proposal generated heated debate. Ledion Bitincka from Splunk responded that log mining software, such as Splunk, offers the only practical solution to unifying the multitude of existing logging formats. Moreover, the log line itself cannot accurately report on what caused it to appear; a mining tool to explore log patterns is therefore essential. Eric's response was that Splunk requires significant user expertise—the language may be simple to learn, but in an unstructured log one must still know how to recognize an error (e.g., HTTP code 404). Another retort to CEE came from Jon Stearley, who asked how a developer might be incentivized to switch and follow the CEE standard. Eric replied that developers today re-invent logging, including timestamp generation, setting delimiters, escaping multi-word messages, etc. CEE would help with all this, and in addition developers may be incentivized to use CEE if customers prefer software that generates logs in this format, as this would be an indicator of software quality and log interoperability.

Greg Bronevetsky asked what makes a good log and which is better, an informational or an activity-based logging strategy. Eric replied that activity-based logs, in which the log captures a change in state, are the most useful for security applications. Others in the audience thought that an informational logging strategy can also play an important role for debugging insidious errors that may span multiple components.

Jon asked whether anyone knew of a good survey paper in the field of log analysis. No one could name such a paper, and Jon suggested that this field is ripe for survey papers. Alice Zheng added that a survey paper focusing on diagnosis would be particularly welcomed. Greg mentioned a forthcoming survey paper by Felix Salfner et al. covering online failure detection methods (http://portal.acm.org/citation.cfm?id=1670680). Wei Xu also noted that his dissertation includes a survey in Chapter 7 (http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-112.pdf).

Raja Sambasivan asked the practitioners to imagine what they might want to find in logs if they contained perfect data. John Hammond responded with a description of supercomputing applications. Many of these were diagnostic—having observed some event, one wants to know what and who might have caused it. If there is uncertainty, a list of possibilities ordered by their likelihood would be useful. Alice wondered whether this is feasible if the data does not contain enough labels. To this, John said that one can label things by hand, as they have done with their own datasets. Ideally, however, such labeled datasets would be widely available to the community to experiment with.

This discussion was broadened by Adam Oliner, who pointed out that the log analysis field, if it is to be scientifically rigorous, needs a freely accessible log repository and a set of common metrics. Adam mentioned that he and Jon Stearley have made available a large, tagged, unfiltered dataset and that they encourage the community to make use of it (http://cfdr.usenix.org/data.html#hpc4). More generally, Adam pointed to the USENIX Common Failure Data Repository (CFDR—http://cfdr.usenix.org/) as an example of how logs can be made available to the broader research community. Greg proposed the idea that the SLAML community can organize around a few logs a year so that analyses reported for the same data source may be compared.

During the discussion of common analysis metrics, user studies were pointed out as a rigorous means to evaluate graphical log representations. To this Ari asked, what sorts of user studies in particular would the SLAML community trust and find useful? Adam responded that this is not something the community has a standard for. Wanchun Li noted that usability studies in security research face a similar issue. Everyone knows that evaluating usability is often an important aspect of the research, but there is little progress on establishing common usability metrics.

Another challenge touched on by a few participants is that the meaning of logged messages may change over time. This can, for example, make log priority levels meaningless (what used to be an error message is now an informational note). This is in part because there is no incentive to remove a log line after the error is fixed. Greg proposed fault injection as a potential solution. With fault injection one can see which messages correlate, and this reveals information about the underlying dependency graph. After all, Greg asked, isn't this the only thing we can find out, namely that certain events correlate, while others do not? Alice questioned whether fault injection can be a complete solution. In particular, she asked about how one translates information gained from fault injection in a testing environment into a production environment. Greg admitted that this is a limitation, but also emphasized that by performing fault injection across different configurations one can often glean important properties of the system, such as its scalability. Raja also thought fault injection to be impractical because it is difficult to trust results gathered in an artificial setting. He pointed out that interesting real-world bugs always seem to be much more involved, and reproducing them in fault injection studies is a research study in itself.

The mention of many limiting features of system logs led Ivan Beschastnikh to ask whether the community should instead consider bridging log analysis with program analysis, as program source can offer logs analysis key contextual clues. Ari Rabkin mentioned relevant work by Ben Liblit on cooperative BUG isolation, which leverages large numbers of execution observations (execution logs) for debugging. Ari also indicated that Ivan's proposal is impractical because program analysis rarely scales to large software and that system software analysis is especially difficult, as it may involve tracing complex execution (e.g., across multiple bash scripts). Alice pointed out that instrumenting real code has an overhead and it's difficult to tell which pieces are important to instrument and which ones do not add much more value.

Alice mentioned that a key challenge in applying machine learning to logs is the lack of labels. She suggested that crowd sourcing (e.g., via Mechanical Turk) can be leveraged to label existing logs. For example, Google improves its search by instrumenting their pages and monitoring the click-through behavior of its many users. Vijay Gurbani noted that the lack of labeled logs is primarily an issue with computer science education—students are taught how to program, but not how to properly organize their program's log output nor how to label and then study the output to understand their programs. Also in response, Wei noted that labeling is especially difficult, because the same log message may mean different things to different people (e.g., a developer versus a system administrator). Mitchell Blank raised the related logging incentives challenge—developers will always opt for an easier way, so one must provide an incentive for them to produce meaningful and easy-to-analyze logs.

Sixth Workshop on Hot Topics in System Dependability (HotDep '10)

October 3, 2010
Vancouver, BC, Canada

Distributed Algorithms

Summarized by Hussam Abu-Libdeh (hussam@cs.cornell.edu)

Storyboard: Optimistic Deterministic Multithreading

Rüdiger Kapitza, Matthias Schunter, and Christian Cachin, IBM Research—Zurich; Klaus Stengel and Tobias Distler, Friedrich-Alexander University Erlangen-Nuremberg

Rüdiger Kapitza opened his talk by noting how nowadays conventional infrastructure is replaced with network-based services where redundancy via state machine replication is used to balance load and achieve high availability. In a typical deterministic state machine replication setting, clients talk to the replicated service via agreement nodes that produce an ordering among client requests, which are then forwarded to execution nodes. As a consequence, we expect that every non-faulty replica will produce the same non-faulty output for the same sequence of client requests. Even though this sounds simple, multi-threaded execution at the replicas complicates things by introducing nondeterminism due to scheduling.

To solve this issue, Rüdiger introduced the Storyboard design for lock prediction and controlled execution, where an oracle is used to predict replica concurrency issues by representing execution paths as ordered lists of lock accesses, which are then executed in a controlled multi-threaded fashion. In the Storyboard design, clients talk to agreement nodes, which then talk to predictor nodes that predict and forecast locks usage, and finally a controlled execution is carried out by the replica nodes which operate according to the forecast Storyboard. In a controlled execution, threads are allowed to execute at their own speed, but they are only allowed to enter into the predicted list of critical sections and are not allowed to overtake other threads into a critical section, and thus the execution will follow the forecast "story."

This was the main idea of Storyboard, and Kapitza proceeded to talk about implementation issues such as handling mispredictions and complex locking structures such as condition variables to synchronize multiple threads, and nested locks. Development is currently underway for a Storyboard prototype, and preliminary results showed an analysis of lock usage in CBASE-FS.

After the talk an audience member asked whether execution rolls back if a thread needs to take an unpredicted lock. Kapitza answered that there is no need for rollbacks, since the current system pauses the execution at the point where an unpredicted lock is requested, and that will enforce the new story across the different replicas. In response to another question, the presenter acknowledged that the current Storyboard design does not address situations with data races, although that is a point for future work.

Scalable Agreement: Toward Ordering as a Service

Manos Kapritsos, UT Austin; Flavio P. Junqueira, Yahoo! Research

Common practice in reliable services is to deploy replicated execution nodes that are preceded by ordering nodes that order client requests so that replicas execute in the same order. However, ordering is left for service developers to deploy, which requires us to provision for nodes that do not do computation and that can additionally become a bottleneck if the core service becomes popular. With this setting, Manos presented his vision for request ordering as a utility service.

A problem is that ordering uses agreement protocols which do not scale, and in fact, generally, adding more machines to agreement protocols increases complexity and not throughput. This is because in most agreement protocols, clients contact a primary node that proposes an order for the requests and broadcasts it to replicas that do an all-to-all communication to agree on the order and finally execute.

Scalable ordering protocols are needed to enable Ordering-as-a-Service, and here Manos proposes leveraging multiple small ordering clusters to compute partial orders and using virtual slot space to get the full order. In virtual slot space each ordering cluster is assigned a color, and the full order is composed by interleaving a slot from each color. For example, given three ordering clusters—blue, red, green—a full ordering schedule can be obtained by executing the first request from the blue cluster first, followed by the first request from the red cluster, followed by the first from the green. Next comes the second request from the blue cluster followed by the second from the red, and so on.

After explaining the general idea, Manos described implementation details such as mapping clusters to physical nodes to balance load, keep faults independent, exploit multicore, and reduce bottlenecks. A preliminary evaluation was made on Emulab with a micro-benchmark of reading/writing key-value pairs from a hashtable. The evaluation demonstrated the scalability of the ordering service by adding more nodes to the configuration and achieving higher ordering throughput. The scalability was measured in terms of adding more clusters and more machines to a fixed set of clusters.

One attendee wondered whether it would make more sense to consider correlation at the rack level rather than on a per-core/process level, and Manos acknowledged that failure correlation depends on the actual deployment environment and the availability required of the service. Hakim Weatherspoon from Cornell University asked about the performance of a service in case of a failure, and Manos responded that a machine's failure will require reconfiguration and reallocation of replicas to other machines. Finally, Atul Singh from Princeton University questioned the usefulness of having a single common ordering service for applications that do not share data, as this extra service might not be all that crucial to the applications. Manos answered that their motivation is to allow scalability such that ordering is never an issue. An added benefit of having an ordering service is that it is one less thing for developers to worry about; just use "the ordering service" and you're done.

Active Quorum Systems

Alysson Bessani, Paulo Sousa, and Miguel Correia, University of Lisbon, Faculty of Sciences

Alysson argues that state machine replication is conceptually simple and it usually provides linearizability, which is a stronger consistency model that is not required in many applications. This makes it difficult to implement tasks like housekeeping/cron jobs, asynchronous messaging, or multi-threaded services. The current mantra is that strong consistency should be avoided at all costs and that led us to embrace eventual consistency. However, eventual consistency is not always adequate and some applications just require strong consistency.

The research question posed is, would it be possible to build dependable and consistent services that rely on strong synchrony only when it is absolutely necessary? To answer this question, Alysson looked at high-level abstractions like coordination services, and low-level abstractions like read/write quorum systems, leader election, and barriers.

Along those lines, Alysson proposes Active Quorum Systems (AQS), which he describes as a Byzantine quorum system with synchronization power. AQS breaks the system state into small objects, where instead of having the entire service as a replicated state machine, the service is viewed as a set of replicated objects. AQS supports three types of low-level operations: read, write, and read-modify-write, which updates the state of an object using its old value. Read and write operations are implemented as in typical quorum-based asynchronous protocols. The read-modify-write operation is implemented as an extension to PBFT (Practical Byzantine Fault Tolerance) where the primary acts locally on a received request and then broadcasts the triple (start state, command, result) to all the replicas. If the replicas approve, then the change is committed; otherwise the most recent copy of the state is sent back to the primary and the operation is repeated until consensus is achieved. A final design principle of AQS is that the service specification is exploited in order to find opportunity for optimization (so not based on the environment, because it can change). An example of that is determining the level of consistency by the service needs, and the same goes for writer access control. Alysson argued that the benefits of AQS are that it makes minimal assumptions, achieves communication optimality, and provides stability for non-favorable executions.

In response to a question from the audience, Alysson noted that unfortunately AQS adds complexity to building systems for non-experienced users. Hakim Weatherspoon asked what would happen if the service assumptions were violated. For example, what would happen if the system were to evolve to allow multiple writers? Alysson responded that the idea of having multiple writers is not associated with contention but with access control. They in fact encountered a case of evolving the system when working on LDAP but it has not been completely worked out.

OS Reliability

Summarized by Mark Spear (mspear@cs.ubc.ca)

We Crashed, Now What?

Cristiano Giuffrida, Lorenzo Cavallaro, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Cristiano Giuffrida presented an operating system model that addresses many problems involved in crash recovery. He immediately brought laughs to the audience with a Blue Screen of Death slide; in this particular BSOD, a device driver bug brought down the entire operating system. Much of the related work on crash recovery focuses on isolated subsystems (e.g., device drivers, file systems). In these works, a portion of the system is trusted to monitor the untrusted portion (e.g., the driver). But when extending crash recovery to the entire system, that model would require monitoring the monitor, ad infinitum, "like a dog chasing its tail."

Instead, Cristiano's group elected to combine OS design and lightweight instrumentation to scale crash recovery to the entire OS. They break down the operating system into independent user-space processes, resulting in a multiserver microkernel-based OS architecture. Each process follows an event-driven model and is solely dedicated to carrying out a specific task in a loop, termed the task loop. By design, the top of the task loop is a local stable state. The task loop can generate idempotent messages throughout, and any non-idempotent messages generated while handling a request are pushed to the end of the loop. Lightweight recovery code is added through instrumentation by LLVM and is used to revert to the last stable state in the event of failure. A shadow state region is used and memory allocations are tracked, as are object state changes, etc. These changes are all committed at the top of the event loop. When the system manager detects a crash (e.g., in the Process Manager component), a replica of the component has the last stable state transferred to it and resumes operation as if nothing bad happened. The system manager then cleans up the dead component. The authors have prototyped the ideas described in the paper on top of the MINIX 3 microkernel.

One audience member voiced concerns about several issues, including multi-threaded servers, communication through shared memory, and blocking on I/O. Cristiano responded by saying that they are not aiming for backward compatibility but, rather, designing a new system using the event-driven model. This model would use asynchronous IPC instead. The performance evaluation was questioned, as this system was compared to unmodified MINIX 3 rather than Linux. Cristiano called attention to the fact that the scalability graph was normalized data, only showing relative performance/overhead. How long did this work take? It was hard for Cristiano to separate the time involved for different parts, but he said that about one year was spent on implementation, with previous work already having been done on the design.

Improved Device Driver Reliability Through Verification Reuse

Leonid Ryzhyk, NICTA and University of New South Wales; John Keys, Intel Corporation; Balachandra Mirla, NICTA and University of New South Wales; Arun Raghunath and Mona Vij, Intel Corporation; Gernot Heiser, NICTA and University of New South Wales

Leonid Ryzhyk observed that while hardware device verification and device driver development have a remarkable degree of similarity, the two processes are currently completely disjoint. Thus we are robbed of an opportunity for more reliable driver development. Leonid's presentation began in the same fashion as the previous one, with a Blue Screen of Death, and the audience continued to find this gag funny.

Current techniques for dealing with driver reliability include runtime isolation, static analysis and model checking, safe languages, etc. At the end of the day, drivers are still much less reliable than we'd like, and Leonid proposes a complementary approach for improving driver reliability. He presented the observation that the most common class of driver bug is device protocol violation (e.g., using an invalid sequence of commands, wrong use of DMA descriptor, interpreting data incorrectly). Hardware designers communicate to the driver developers through a datasheet, which often contains inaccurate information. However, the hardware verification engineers are privy to more details about the device, and lots of effort is put into the verification testbench. The testbench has several layers, including scenario, agent, and the device under test, which have analogs in the OS I/O stack. The scenario layer should be extended to be OS-based and simulate how the OS uses a driver. The agent layer has a similar role to that of a driver (translating a high-level request to low-level operations, and changing state of the device). Leonid suggests that an actual driver could replace the agent layer, so that hardware and software are being co-verified. In order to facilitate cross-platform use of the single driver under test, Leonid proposes unified driver interfaces (per device class), instead of naively emulating existing OS interfaces in the testbench (which would lead to OS-specific testing). The resulting driver could then be used without modification in real operating systems.

This could result in several benefits, including a reduced dev cycle and (naturally) fewer bugs in the end product. They found a number of defects in USB and UART drivers, an Ethernet hardware race condition, and several other bugs that weren't found using the conventional development method.

A number of questions involving different testing configurations were asked. The first inquiry was about different versions of a hardware device. That could require updated testbenches in both the conventional and the proposed model. The same questioner also asked about different versions of a driver. The response was that drivers would require testing anyway, so now it would be done through the testbench. When an operating system changes, is it the job of the hardware vendor to retest the driver? It would require work if the testbench was emulating the OS, but instead a generic interface is assumed, so that isn't an issue.

Towards Automatically Checking Thousands of Failures with Micro-specifications

Haryadi S. Gunawi, University of California, Berkeley; Thanh Do, University of Wisconsin, Madison; Pallavi Joshi and Joseph M. Hellerstein, University of California, Berkeley; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison; Koushik Sen, University of California, Berkeley

Thanh Do presented a mechanism for exploring complex failures and error recovery scenarios. In the era of cloud computing and using thousands of machines, failures are not so rare anymore. The reliability has to come from the software, but even the big players get it wrong sometimes (e.g., Sidekick data loss, Facebook photo loss). Thanh argues that current testing of failure recovery is insufficient. In response to this problem, his group has developed a pair of tools that work in concert to explore failures. In the time since their paper was submitted to HotDep, these tool, FTS (Failure Testing Service) and DTS (Declarative Testing Specification), have been renamed FATE and DESTINI, respectively.

FATE is a failure injection framework. It targets I/O points and can exercise many combinations of failures. A "Failure ID" is their representation of a failure: It contains a failure point (the system/library call that performs disk or network I/O), a failure type (crash, exception, etc.), a stack trace, and some domain-specific information (e.g., source, destination, message). The hash of a failure ID is used to log the failure history when exploring the failure space. Aspect-oriented programming (AspectJ) is used to instrument Java programs with no changes required to the system under test. Multiple failures are injected, including failures during recovery. DESTINI is responsible for testing whether actual behavior is consistent with what was expected. Violations are detected through evaluation of Datalog style rules.

The authors applied their system to three cloud systems, HDFS, ZooKeeper, and Cassandra, writing 74 recovery specifications at an average length of 3 lines per specification. Their system found 16 bugs and reproduced 74. The bugs caused reduced availability and performance, data loss during multiple failures, and errors in the recovery protocol.

The first questioner pointed out that one man's bug is another man's feature: What do you do when you don't have a precise specification? Thanh noted that even without a precise specification, you should at least have a high-level expectation of how the system works. If necessary, the specification could be refined later, and there is no need to start at a low level. Another questioner asked about one of the hardest situations to test: arbitrary corruption. The framework supports various kinds of corruption (e.g., network packets), but the full answer about arbitrary state corruption was taken offline. Finally, an audience member noted that when considering multiple possible failures, the state space explodes. Thanh mentioned that they were looking at heuristics for pruning the failure space, prioritizing some failures, and focusing on "interesting points," and that some of that work was submitted to NSDI.

Management and Debugging

Summarized by Brendan Cully (brendan@cs.ubc.ca)

Focus Replay Debugging Effort on the Control Plane

Gautam Altekar and Ion Stoica, UC Berkeley

Gautam Altekar began his presentation with the observation that debugging datacenter software is particularly difficult, for three reasons: it is large-scale, data-intensive, and non-deterministic. Altekar argued that static techniques do not scale to the state space of these large systems, and so we need a way to do deterministic recording and replay of production systems in order to reproduce nondeterministic bugs. But production systems will not tolerate a great deal of overhead in either performance or logging data rate.

Altekar proposed that datacenter applications will typically have two somewhat distinct components: a control plane for managing data flow and maintaining replica consistency, and a data plane of relatively simple data processing engines. He then hypothesized that the control plane, being complex, would have a much higher relative bug rate than the data plane. At the same time, it would have a much lower data rate. For reproducing bugs in the control plane, which he argued were the most important and difficult, Altekar claimed that it would suffice to maintain deterministic recordings of the control plane.

To test this hypothesis, he chose three applications (Hypertable, KFS/CloudStore, and OpenSSH) and used taint tracking to classify code that accessed user data as data plane code (this classification needed manual refinement because of the high rate of false positives produced by taint tracking at the CPU level, and because classification based on observed execution had poor coverage). Bearing in mind that the results were not very scientific, his initial analysis appeared to justify his hypothesis: 99% of the bugs reported in these applications were in "control plane" code, but this code accessed only 1% of the data processed during execution. Altekar believed that this result warranted further investigation.

There were a lot of questions about how cleanly control and data plane code could be separated in practice. Derek Murray (Cambridge) wondered how this would work for systems like Google Percolator, in which the results of the data plane could affect the control plane. Dutch Meyer (UBC) asked whether data traffic could be distinguished by directly examining the data. Altekar responded that the distinguishing feature of data plane data was volume. Steve Hand (Cambridge) noted that the chosen applications were application frameworks and wondered how well the observed bug rates would correspond with those for actual applications built on the frameworks. Altekar intended to look into that.

A Rising Tide Lifts All Boats: How Memory Error Prediction and Prevention Can Help with Virtualized System Longevity

Yuyang Du and Hongliang Yu, Tsinghua University; Yunhong Jiang and Yaozu Dong, Intel Research and Development, Asia-Pacific; Weimin Zheng, Tsinghua University

Yuyang Du began his talk with the surprising claim that RAM errors cause the plurality of system failures (including both software and hardware causes). While these could be prevented with hardware fault tolerance or even simple ECC, Du claimed that cost prevented widespread deployment of these techniques. He further noted that in the increasingly important cloud hosting economy, consumers don't have direct control over the hardware on which their applications run. He also claimed that virtualization compounded the problem of unreliable memory, because the physical hardware was multiplexed across multiple servers.

Du noted that memory chips produced both correctable (soft) and uncorrectable (hard) errors. He proposed that for virtualized servers, a cost-effective approximation of ECC could be achieved by using observed soft errors to predict future hard errors and migrate virtual memory off the failing physical RAM before they happened. He had not yet evaluated his failure predictor or any gains in reliability versus the cost of disabling physical RAM. He noted that this sort of project was very slow to test, since it depended on waiting for memory to fail.

The questioning focused on two topics: the accuracy of Du's hard memory error predictor, and on the idea of creating graduated memory protection. Steve Hand (Cambridge) asked about using ECC memory for the hypervisor but only best-effort, predictive protection for virtual machines. Karthik Pattabiraman (UBC) took the idea further, proposing that individual applications might be able to make use of memory of varying reliability. Du agreed that these were interesting avenues of research.

A Design for Comprehensive Kernel Instrumentation

Peter Feiner, Angela Demke Brown, and Ashvin Goel, University of Toronto

Peter Feiner and his co-authors would like to protect systems from faulty device drivers, using techniques like Microsoft Research's Byte Granularity Isolation. The trouble is that such techniques require source code, but often the source for device drivers is unavailable. Applying this technique to binary code requires a dynamic binary instrumentation (DBI) framework (like Valgrind, DynamoRIO or Pin), but Feiner claims that no such framework exists that can operate on kernel code.

The authors see two approaches to constructing a kernel-level DBI framework: either port an existing tool like Pin to an existing hypervisor, or build a minimal hypervisor with just enough code to support DBI. Claiming that a port would be difficult due to the amount of code that would need changing, they instead investigated the features required of a custom DBI hypervisor. Feiner spent the rest of his talk enumerating many of the tricky issues involved in performing invisible code translation at the kernel level.

I was surprised both by the claim that a port approach was more difficult, and that it hadn't been done before. I asked Feiner how his model compared to PinOS, which combined Pin with Xen and was described in a VEE paper in 2007. Feiner said that the disadvantage of PinOS was that the TCB for the translation engine was much larger. Others asked about overhead, and issues dealing with self-modifying code. Feiner noted that his talk was about a proposed architecture rather than an implemented system, and so discussions of mechanics and performance were not yet relevant.

Storage and File Services

Summarized by Mark Spear (mspear@cs.ubc.ca)

Behavior-Based Problem Localization for Parallel File Systems

Michael P. Kasick, Rajeev Gandhi, and Priya Narasimhan, Carnegie Mellon University

Mike Kasick described a method of diagnosing problems in parallel file systems by analyzing system behavior via CPU instruction pointer sampling and function call tracing. This work was motivated by real problems experienced by the developers of PVFS (Parallel Virtual File System): limping-but-alive servers that reported no errors, faulty and overloaded switches, buggy RAID controllers, and a variety of other problems that may pass their respective diagnostic tests. Previous work has shown instances where performance manifestations of problems were masked by normal deviations. However, behavioral manifestations may be more prominent than performance manifestations.

Fault-free peers have similar behavior: e.g., large I/O requests are striped across all servers, and small I/O requests, in aggregate, also equally load all servers. A system exhibiting a fault (e.g., "Write-Network-Hog Fault") will have a behavioral manifestation (e.g., a gross discrepancy in calls to the kernel tcp_v4_rcv function). Each server has a feature vector of several metrics from a sliding window of time. The features include samples, function call counts, and time. To detect anomalous behavior, Manhattan distances between feature vectors are computed pair-wise between servers. The median distance for a given server is tested against a threshold: The (per-server) threshold for "anomalous" is set via a fault-free high-stress training phase, to find the maximum deviation expected under normal conditions. This requires training on each cluster/filesystem combination, but not based on workload.

Examining the most anomalous metrics facilitates root-cause analysis. Given an indicted node, and its feature vectors over time compared to others, the system can present a list of possible anomalies for manual inspection. Different metrics are good at different kinds of faults. Disk faults are best detected by the time metric, since blocking I/O calls are used. Count metrics are good for detecting packet loss: dropped packets cause more non-blocking reads, resulting in a higher function call count. Sampling is useful for detecting the "network hog" fault, as TCP retransmits increase the CPU load.

Steve Hand (Cambridge) asked a question about how the threshold was selected. It appeared to be a constant number; why not set the threshold as a function of what is observed at runtime (like a number of standard deviations)? Mike noted that it is not really an absolute number, although the example slide may have made it seem that way. Instead, it is a maximum degree of tolerable deviation.

What Consistency Does Your Key-Value Store Actually Provide?

Eric Anderson, Xiaozhou Li, Mehul A. Shah, Joseph Tucek, and Jay J. Wylie, Hewlett-Packard Laboratories

Xiaozhou (Steve) Li presented an analysis of key-value stores. Service-level agreements for consistencies of key-value stores are likely on the horizon. Most stores only promise eventual consistency, but for some workloads (depending on the number of updates, how much contention there is), it may perform better.

Key-value stores are commercial black boxes. What can you do to analyze the consistency to see if you need a higher level of service? Client machines can record sequences of get/put requests and record the time that requests are sent and replies are received. Based on the observed sequence, one can attempt to analyze atomicity, regularity, and safety (properties from Lamport's work on register-based consistency). The presentation focused on atomicity, and a graph theoretical approach was offered. The vertices represent operations and edges represent precedence. The sequence is a "good" sequence if and only if it is a directed acyclic graph. A cycle would imply that a vertex should happen before itself (because edges are precedence). There are three types of edges: time (if an operation precedes another entirely), data (write 0->read 1; the value of the write should appear in the value of the read), and "hybrid" edges (which enforce the invariant that "all writes time-preceding a read should happen before the read's dictating write"). Cycles are counted via DFS, and the number of cycles detected is representative of how severely inconsistent a trace is.

Measurements of the same sequence of operations from the service provider's side are necessarily shorter, because they wouldn't include network latency. Therefore, more time edges would be added to the graph (from the service provider's perspective). Thus, a user detecting a violation (i.e., a cycle in the graph) would imply that the service provider would also know there was a violation. Some evaluation was done, and it was noted that their key-value store, Pahoehoe (which is eventually consistent), with sufficiently low contention, is about atomic.

Steve agreed with an interesting possibility presented by an audience member: With these commercial systems exposing the same interface, if violations are noticed, one could take action and switch to another provider. Another audience member asked, if a service provider violation is detected, how can you prove it? Steve noted the service provider, if running this analysis, would also detect the violation, because they would have more time edges (because of the apparent shortening of operations). But to actually "prove" the violation, you would need to incorporate non-repudiation techniques to convince a third party, using digital signatures and related techniques. The final question was whether the algorithm could be run in parallel if keys were in disjoint cliques, and the answer was yes: it is straightforward, as if they are different keyspaces.

2010 Workshop on Power Aware Computing and Systems (HotPower '10)

October 3, 2010
Vancouver, BC, Canada

Impact of Hardware Trends

Summarized by John McCullough (jmccullo@cs.ucsd.edu)

Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns

Etienne Le Sueur and Gernot Heiser, NICTA and University of New South Wales

Etienne Le Sueur observed that dynamic voltage and frequency scaling is a technique commonly used to reduce the power of a running system. The dynamic power of a system scales linearly with frequency and quadratically with voltage. If we can drop the voltage and frequency, then we can expect a drop in power. In 1994 we saw 25–65% savings up through 2009, when work saw a 30% energy savings with a meager 4% performance loss. Unfortunately, there is also a static power component that includes leakage current, memory refresh power, hard drive motors, etc. Thus, running more slowly can potentially degrade the overall power.

The authors consider the mcf and gzip benchmarks and observe that mcf gets the largest benefit from DVFS because it is memory bound rather than CPU bound. Looking across three different Opteron CPUs, they found that the older two had a power-optimal point around 1.6GHz but that, with a more modern processor, DVFS was ineffective at saving power. One shortcoming of this technique is that it assumes that the computer is not consuming any power after the benchmark completes. To consider the idle power following the experiment run, Etienne described their padding methodology to measure the idle power up to the time of the longest running DVFS-scaled benchmark. Using this technique, we start to see some reduction in energy, but we only see improvements in energy delay if running on all system cores.

Moving forward, we can expect that shrinking feature sizes and increasing cache size and memory bandwidth will make improvements by scaling down via DVFS even less likely. Surprisingly, however, features such as Intel's Turbo-Boost can actually reduce total power by scaling up the clock frequency and racing to idle effectively.

An audience member asked how DVFS can impact embedded platforms. Etienne observed that the CPU power can be very low relative to the total system power and that DVFS won't be able to impact total power. Another audience member observed that AMD has introduced DVFS on the cache and memory controllers.

A Case for Opportunistic Embedded Sensing in Presence of Hardware Power Variability

Lucas Wanner, Charwak Apte, Rahul Balani, Puneet Gupta, and Mani Srivastava, University of California, Los Angeles

Puneet Gupta demonstrated how shrinking feature sizes leads to immense variability in the physical manifestation of hardware designs. For example, in an experimental 80-core processor the performance spread across cores on a single die was 80%. The degree of variability is not tied to the design alone, as the same design sourced from different manufacturers can get different degrees of variability. Furthermore, aging can cause wires to slow and reduce performance by 20–30%. Today, variability is masked by guard bands. Processor manufacturers typically bin processors by functional speed, with space for any aging effects. Unfortunately, scaling with the guard band is much worse than the nominally achievable results.

Puneet advocates that exposing aspects of this variability to software can allow improved functionality. Such exposure could happen via active measurement or prior testing, but it can have a significant impact. For instance, in sensing applications the sleep power is dominant and can affect the amount of data that can be acquired on a given power budget. The authors found that for 10 off-the-shelf Cortex M3 processors, the active power varied by 10%, but the sleep power varied by 80%. Furthermore, the sleep and active power vary with temperature. Using a power model calibrated to temperature and sensors for power, Puneet demonstrated that they can achieve more effective sensing by energy-aware duty cycling. Thus, a node with lower sleep power can sample 1.8x more data than it would have if all nodes were timed to the worst sleep power. Moving forward, one challenge is to discover the right interface for exposing the variability and sense data to software.

An audience member observed that this data is already integrated in modern CPU power management units for managing the frequency within temperature and power bounds. Puneet responded that it is important to actually expose this information to the software layer, which most of these techniques fail to do. Another audience member asked about the overhead of these techniques. Puneet observed that the information is already being collected for quality control but it is not exposed to software. Finally, an audience member asked about the difficulty managing other system components. Puneet said that the complexity would depend on the abstraction.

Invited Talk

Summarized by John McCullough (jmccullo@cs.ucsd.edu)

Datacenter Power Efficiency: Separating Fact from Fiction

Kushagra Vaid, Microsoft Corporation

Kushagra Vaid posed the question of how to maximize power efficiency at a large scale. A data center consists of a power substation, chillers, batteries, an ops room, generators, fuel, and computing equipment. The efficiency of a facility is often measured by PUE, which is the facility power divided by the IT equipment power. Common PUE values range around 1.5–2.0, but good deployments reach close to ideal at 1.05.

Kushagra observed that the cost of power is fairly low relative to the overall costs of the facility. Amortizing using a three-year replacement model, power consumption consists of 16% of the cost. Thus, reducing dynamic power and energy-proportional computing has limited benefits. The capital expense of servers accounts for the largest portion of the total cost of ownership.

To minimize costs in provisioning, Microsoft is moving towards modular data centers. A video showing the modules is available at http://www.microsoft.com/showcase/en/us/details/84f44749-1343-4467-8012-9c70ef77981c. The modules function using adiabatic cooling with outside air and only using top-of-rack fans to adjust for cooling/heating as appropriate. To reduce the cost of power, there are techniques for eliminating conversion steps. Surprisingly, running AC to the servers is not significantly different from DC in total conversion efficiency. Right-sizing for density suggests a sweet spot, with the lowest-voltage processor getting a surprising performance/watt/cost benefit. Right-sizing storage can be achieved by in-depth storage trace analysis to understand workload patterns and dynamic range to pack better.

Overall, researchers need to be sure that they take a holistic view. Current research areas are in optimal provisioning for high dynamic range workloads, addressing energy proportionality via system architecture innovations, power-aware task scheduling on large clusters, and energy-conscious programming using controlled approximation.

One audience member asked why their efficiency approaches don't work in all data centers. Kushagra responded that getting all of the layers of UPSes and voltage conversion requires control of the entire data center, which few have the scale to accomplish. What are the implications of low-power CPUs in data centers? For Bing workloads, Xeon systems are 2.3x better in performance/watt/cost. Someone asked why they don't turn off servers. Kushagra replied that it can lead to response time spikes, that turn-on time can be long, and that if you can turn off servers, you brought too many online or you are doing poorly at task scheduling.

Data Center I

Summarized by John McCullough (jmccullo@cs.ucsd.edu)

Analyzing Performance Asymmetric Multicore Processors for Latency Sensitive Datacenter Applications

Vishal Gupta, Georgia Institute of Technology; Ripal Nathuji, Microsoft Research

Asymmetric multicore processors (AMPs) are being explored in multiple dimensions, where cores are combined with different performance characteristics and deployed with different functional characteristics. Vishal Gupta presented their technique for understanding the impact that AMPs can have on data centers with respect to power and throughput.

Vishal described two use cases: energy scaling and parallel speedup. Energy scaling involves execution on a combination of the small and large cores to achieve the same computation within a deadline at lower power. Parallel speedup occurs when a larger core on an AMP can execute serial sections faster. To ascertain the effects of these use cases, Vishal treats each processor as an M/M/1 queue, which models processing times as an exponential distribution where the processing time is parameterized in proportion to the chip area. Overall completion time is parametrized by the paralellizable fraction of the code. Using this model, Vishal finds that for a higher fraction of parallelizable work, power savings increase with AMP use. While there are practical considerations, AMPs offer more potential for parallel speedup than for energy speedup.

Energy Conservation in Multi-Tenant Networks through Power Virtualization

Srini Seetharaman, Deutsche Telekom R&D Lab, Los Altos

Networks are typically power oblivious and it is hard for network users to ascertain the impact. By packing flows into fewer devices and turning off unused devices, the system can turn off individual ports and even entire switches. Given a multi-tenant data center, how can tenants be influenced to reduce their system usage? Switching from a flat rate for networking to a power-based price can incentivize power savings.

Srini Seetharaman proposed the idea of virtual power. Because network power is not proportional usage, the most intuitive definition for virtual power is to split the power consumption of a component over all sharing tenants. This has the effect of penalizing a tenant for being the only occupant and encourages reuse of pre-paid/pre-powered-on elements. An implementation is in progress but there are no results yet. The specific methods of billing and pricing can influence the outcome; for instance, auctions might introduce different behavior than allocations that degrade over time. In the future, the question is how we can achieve good performance while conserving power.

In the Q&A, Srini clarified that it makes more sense to conserve in a reactive mode, turning on devices as necessary. One audience member asked whether rate-based power differences can affect power. Srini replied that the power differences are typically small. The same person also asked whether the placement of tenants in the data center could penalize them in terms of the available pricing. Srini replied that this was a concern.

Data Center II

Summarized by Etienne Le Sueur (elesueur@cse.unsw.edu)

Energy Savings in Privacy-Preserving Computation Offloading with Protection by Homomorphic Encryption

Jibang Liu and Yung-Hsiang Lu, Purdue University

Yung-Hsiang Lu presented this paper, which discusses the issues that arise when compute-intensive tasks are offloaded from mobile devices to a centralized server. The main issue their work addresses is that of privacy, when sensitive data needs to be transferred off the mobile device, using a public network, to a server which may not be trusted.

Their work mitigates this privacy issue by protecting the data using a homomorphic encryption algorithm. Homomorphic encryption is unlike traditional encryption techniques in that computations can be done on the cipher-text itself rather than being decrypted first. This way, the server operating on the data need not actually know what information the data contains.

The use-case they describe in the paper deals with image-matching—for example, when a user of a mobile phone takes a photo and wants a remote server to do some analysis to try and determine what objects the photo contains. They used an iPad portable device and a server with a 2GHz CPU for processing the images, with evaluation based on how many positive matches were made. Using their modified Gabor filter, they were able to get a correct match approximately 80% of the time when the analysis was performed on the cipher-text.

The work seems promising, and a future direction was clearly given which will address the issues with noise in the encryption system.

An audience member asked whether there were other classes of functions where it makes sense to offload computation. They haven't reached a point where there's a general rule for when to apply the technique. How strong is the encryption and can it easily be broken? The strength of the encryption depends on the key length. The discussion was continued offline.

Green Server Design: Beyond Operational Energy to Sustainability

Jichuan Chang, Justin Meza, Parthasarathy Ranganathan, Cullen Bash, and Amip Shah, Hewlett Packard Labs

Justin Meza presented this paper, which discusses a way to quantify sustainability when designing data centers. The usual motivations for the work were given: e.g., reduction of carbon footprint, secondary costs (power and cooling), and government regulation.

To measure sustainability, several costs need to be addressed: extraction of materials, manufacture of systems, operation and infrastructure, and recycling once end-of-life is reached.

They use a metric called "exergy," which basically constitutes TCO (total cost of ownership) plus the energy required to manufacture devices. Objects such as servers accumulate "exergy" by breaking them down into components, such as hard drives and processors, and trying to determine the cost of the raw materials that go into making them.

Supply-chain information is included in the "exergy" calculations, which include cost of transportation. For an example server, they find that the cost of manufacturing the server was roughly 20%, 27% was infrastructure-based cost, and 53% was operational costs like power and cooling.

The next part of the talk discussed how using certain techniques to alter energy-efficiency affected "exergy." On one hand, they looked at consolidating servers (reducing total idleness) and on the other hand they looked at energy-proportional computing techniques, such as reducing CPU frequency when there is idleness. They found that if reducing total "exergy" is the goal, then consolidation is the best approach, but if reducing operational "exergy" is the goal, then energy-proportional computing techniques give better results.

GreenHDFS: Towards an Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster

Rini T. Kaushik, The University of Illinois at Urbana-Champaign and Yahoo! Inc.; Milind Bhandarkar, Yahoo! Inc.

Rini T. Kaushik presented the authors' attempt to leverage the distributed nature of the Hadoop file system. The basic premise is that the cluster of servers is divided into two sections: a hot section, which contains recently used data, and a cold section, which contains data that has been untouched for some time.

Initially, they did an analysis of the evolution of files in an HDFS (Hadoop Distributed File System) cluster, by looking at three months' worth of traces from Yahoo. They found that 90% of data is accessed within two days after a file is first created. Additionally, they found that 40% of data lies dormant for more than 20 days before it is deleted. This essentially means that data is "hot" for a short period after creation, and then "cold" for a much longer period.

Computation follows the 'hot" zones, and the "cold" zones see significant idleness and can be scaled down or turned off. They determined the best division was 70% hot and 30% cold. They claim that these techniques can save 26% of the energy used in the cluster they were testing.

An audience member asked the presenter to clarify how they were able to save 26% energy. A large number of the servers in the "cold" zone were in fact never turned on during the three-month simulation.

Myths, Modeling, and Measurement

Summarized by Lucas Wanner (wanner@ucla.edu)

Demystifying 802.11n Power Consumption

Daniel Halperin, University of Washington; Ben Greenstein and Anmol Sheth, Intel Labs Seattle; David Wetherall, University of Washington and Intel Labs Seattle

Anmol Sheth started by observing that WiFi is becoming ubiquitous and that the increasing bandwidth demands of networked applications are leading to the widespread adoption of 802.11n, the latest version of the standard. In battery-operated devices such as mobile phones, radio interfaces can account for a significant portion of total power budget. There is little data available to help designers operate 802.11n devices in an energy-efficient way. This work presents measurements of 802.11n in various configurations and modes of operation.

802.11n devices may use multiple active RF chains. The characterization study in this work found that power increases sub-linearly with additional antennas, and increase in signal processing power is negligible. Power consumption is hence not a multiple of active RF chains and is asymmetric for transmission and reception. Nevertheless, the use of wider RF channels is more power-efficient than multiple spatial streams.

Another source of energy efficiency for 802.11n communication is "racing to sleep," i.e., transmitting data in high-rate bursts and subsequently putting the card in sleep mode. Because sleep mode may use approximately 10x less power than simply leaving the card in reception mode all the time, this may lead to significant savings.

The first question addressed the issue of energy-efficient operation for bandwidth-intensive applications, such as streaming video. In these cases, racing-to-sleep may be impossible. A second question was about backward compatibility between n and g devices in the same network. This can potentially decrease energy efficiency, as devices must operate at the lowest common denominator. Finally, the test environment was discussed: in the tests conducted in the study, the hosts were in close proximity. Further work is required to evaluate lower-quality links typical of homes and offices.

Chaotic Attractor Prediction for Server Run-time Energy Consumption

Adam Lewis, Jim Simon, and Nian-Feng Tzeng, University of Louisiana

Full system power models are used to estimate energy consumption in servers. This is accomplished by looking at complete system energy and trying to approximate the energy used by various components. Linear methods are simple and have fairly average median error but potentially very high maximum errors. Power traces for a series of benchmarks suggest both correlation and chaotic behavior between samples.

This work showed that models constructed from autoregressive methods demonstrate behavior that makes them problematic for predicting server energy consumption. This work proposed a Chaotic Attractor Predictor which overcomes the limitations of the previous linear regression-based methods by addressing the non-linear aspects of energy consumption in time and captures the underlying chaotic behavior of the system.

During the questions session, it was pointed out that only single-threaded applications were used to predict power consumption in the work. This may be one of the reasons for the non-linear behavior found in power consumption, as the power manager can put the CPU to sleep when it is idle.

Automatic Server to Circuit Mapping with the Red Pills

Jie Liu, Microsoft Research

The objective of this work is to map servers in a data center to the circuit that powers them. Due to complex wiring setups, it's hard to identify which circuit powers each server. Having this information would be beneficial for failover analysis, accounting, power provisioning, and balancing.

The basic idea in this work is to manipulate a server's workload to generate a power signature that can be identified by circuit power measurements, using the power line as a communication channel. A pure, single-frequency signal would be ideal for this identification, but is hard to generate by manipulating CPU utilization. Periodic square wave signals are easier to generate. In the Red Pill system, a manager requests an identification signal from the server through the network and detects this through the power measurement connected to the manager. IDs with 64 samples can be detected with high probability, even with fairly low amplitude signals. Each signature with 64 samples would take about 16 minutes for detection. As the number of servers increases beyond 20, detection likelihood decreases.

Is it hard to identify server-to-power circuit mapping in general, or is this only a problem with "incorrect deployments? Because of the dynamic nature of datacenter deployments, this is a fairly common problem. How can correctness be verified in the system? In the tests conducted for the paper, the mapping (ground truth) was known. In deployment systems, metrics of confidence could be included, and repeated measurements could be used to increase accuracy.

Save the Date!

USENIX Federated Conferences Week

June 12–17, 2011, Portland, OR

www.usenix.org/events/#confweek11

REGISTRATION DISCOUNTS AVAILABLE!

USENIX ATC '11

2011 USENIX Annual Technical Conference

Wednesday–Friday, June 15–17

www.usenix.org/atc11

WebApps '11

2nd USENIX Conference on Web Application Development

Wednesday–Thursday, June 15–16

www.usenix.org/webapps11

WIOV '11

3rd Workshop on I/O Virtualization

Tuesday, June 14

HotCloud '11

3rd USENIX Workshop on Hot Topics in Cloud Computing

Tuesday, June 14

www.usenix.org/hotcloud11

HotStorage '11

3rd Workshop on Hot Topics in Storage and File Systems

Tuesday, June 14

www.usenix.org/hotstorage11

And more!

Visit www.usenix.org/events/#confweek11 for new workshop announcements and registration information.

usenix

www.usenix.org

Stay Connected...

http://www.usenix.org/facebook

http://www.usenix.org/linkedin

http://twitter.com/usenix

http://blogs.usenix.org

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

Register Now!

nsdi '11

8th USENIX Symposium on Networked Systems Design and Implementation

March 30–April 1, 2011, Boston, MA

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

NSDI '11 will focus on the design principles, implementation, and practical evaluation of large-scale networked and distributed systems in a 3-day technical program including topics such as:

- Data-intensive computing
- Debugging and correctness
- And more!
- Energy and storage
- Security and privacy

Take advantage of this opportunity to join researchers from across the networking and systems community in fostering a broad approach to addressing common research challenges.

Register by Monday, March 7, and save!

usenix

Additional discounts available!

http://www.usenix.org/nsdi11