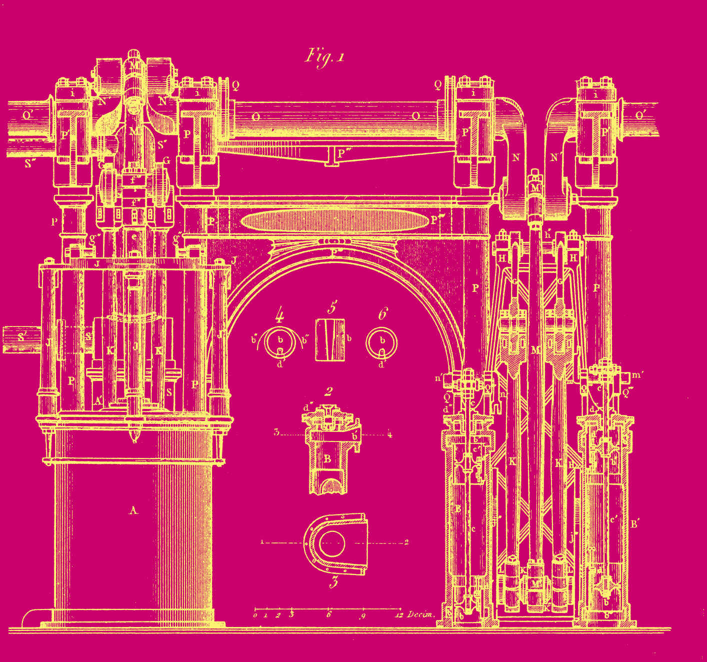
OCTOBER 2008 VOLUME 33 NUMBER 5

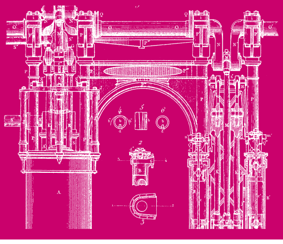
;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events



22nd Large Installation System Administration Conference (LISA '08)

Sponsored by USENIX and SAGE

NOVEMBER 9–14, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/lisa08

Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT '08)

Sponsored by ACM in association with USENIX

NOVEMBER 14–15, 2008, SAN DIEGO, CA, USA
http://www.chimit08.org

ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008)

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

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

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/hotdep08

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

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/wasl08

Workshop on Power Aware Computing and Systems (HotPower '08)

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/hotpower08

Workshop on Supporting Diversity in Systems Research (Diversity '08)

Co-located with OSDI '08

DECEMBER 7, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/diversity08

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

Sponsored by USENIX in cooperation with ACM SIGOPS

DECEMBER 8–10, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/osdi08

First Workshop on I/O Virtualization (WIOV '08)

Co-located with OSDI '08

DECEMBER 10–11, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/wiov08

Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML08)

Co-located with OSDI '08

DECEMBER 11, 2008, SAN DIEGO, CA, USA
http://www.usenix.org/sysml08

1st Workshop on the Theory and Practice of Provenance (TaPP '09)

Co-located with FAST '09

FEBRUARY 23, 2009, SAN FRANCISCO, CA, USA
http://www.usenix.org/tapp09
Paper submissions due: December 5, 2008

7th USENIX Conference on File and Storage Technologies (FAST '09)

Sponsored by USENIX in cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee (MSSTC), and IEEE TCOS

FEBRUARY 24–27, 2009, SAN FRANCISCO, CA, USA
http://www.usenix.org/fast09

2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)

Sponsored by ACM SIGPLAN and SIGOPS in cooperation with USENIX

MARCH 11–13, 2009, WASHINGTON, D.C., USA
http://www.cs.purdue.edu/VEE09/

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

contents

;login:

VOL. 33, #5, OCTOBER 2008

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
David Couzens
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Michele Nelson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

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

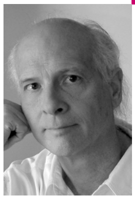
$90 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $120 per year.

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

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

RIK FARROW

musings



rik@usenix.org

VIRTUALIZATION IS ALL THE RAGE these days. We now have multiple alternatives to both server and desktop virtualization software, and virtualization is fast becoming the new "green." As I watched the rush to virtualize unfold, I wondered who had considered the security implications of virtualizing servers?

As with many other shiny new technologies, people don't want to poke too deeply into the works because they might not like what they see. It is human nature to focus on the good side of things and to ignore the messy parts for as long as possible.

Perhaps you are one of those who believe that virtualization makes running servers more secure. Whether you are or not, I invite you to replicate an experiment I ran that helps to resolve the security issues.

You can relax, because the experimental setup is trivial. I so love running thought experiments for this reason, even if the outcomes can vary depending on the hardware or software used to run the experiment. But enough talk. Let's get this experiment running!

The Experiment

First, we need a control. For our control, we will use a modest rack of 15 servers. Each server runs a single application, as we learned a long time ago that running a single service per hardware host supports ease of management, patching, fault isolation, and security.

For our test case, we will take these same 15 servers and virtualize them. This isn't a radical idea; it's merely the rage these days, as we get to utilize our systems much more fully. When running servers on platforms that we outfitted with excess resources because we really didn't know how much we needed and overprovisioning is always the safe bet, we had been wasting resources and energy. The virtualized servers run nicely on a single, if built-out, server, and we can migrate any of them to another virtualization server if they need more resources.

HYPOTHESIS

Okay, now for the hypothesis: Has moving our 15 servers into a single virtualization host made the collection more or less secure? Recall that we simply moved the existing servers over. We didn't patch software, replace buggy software, or move to more secure scripting languages or database services. We just installed the same software within guest domains. I believe the answer here is obvious, but I will spell it out just in case: No, the systems are not more secure than they were. How could they be, as we have not changed anything about the services they were running, and the supporting software?

If they are not more secure, are they less secure? Consider that we have added a new software layer, the hypervisor, along with its supporting software. As with any other software, the virtualization software itself has bugs, including security vulnerabilities. You can visit VMware's security advisory page [1] to get a feeling for this, or create a search using the words "vulnerability" and the name of your favorite virtualization software.

Adding virtualization software increases the attack surface. The attack surface represents what portions of a system are vulnerable to a potential attack; it includes everything from PHP scripts, the Web server, and the system libraries to the underlying OS. To this stack we have added both the hypervisor and its supporting software. In the case of VMware, the bare metal hypervisor, ESX, is 32 megabytes of software. We don't really know how many thousands of lines of code go into making up compiled code with a disk footprint of 32 MB, but surely this took tens of thousands, more likely hundreds of thousands of lines of code. We know there are bugs in the hypervisor code, as some have been patched already. I believe that adding a hypervisor must increase the attack surface beyond where we were before we combined our 15 servers on a single server.

The same will also be true if we decide to use Xen or some other virtualization software as our hypervisor. We have added software, and since software has bugs, the attack surface increases.

ATTACK SURFACE

But let's examine our experimental setup more carefully, On the one hand, we have our legacy rack of independent servers; on the other, we have the virtualized servers, all running on the same hardware. We located our servers on separate hosts partially as a means of increasing their security, and we gave that physical isolation up when we virtualized them. Looking at recent vulnerabilities in virtualization software, we can see that bugs in the hypervisor can give a local attacker root or local system access to the entire system. Thus, we have given up the protection we once had in isolated systems by going virtual. An exploit in one virtualized server can provide unfettered access to all servers, as they are hosted on the same hardware.

There is nothing magic about virtualization. It is merely another OS technology, newly developed outside the world of IBM mainframes, that suffers from the vulnerabilities inherent in any software. And, as with any software, the more features that get added, the greater the potential attack service. Dan Bernstein's DNS software is secure largely because it is so featureless. You cannot bind both an authoritative DNS server and a caching server to the same IP address with djbdns, and the related simplicity reduces the attack surface.

MIGRATION

Did you get the same results running the thought experiment on your hardware that I got on mine? I suspect so, unless adding positive numbers together produces a zero or negative result using your hardware/software stack. But let's not stop there, as there are file systems to consider.

I actually first approached the issue of virtualization security from the other angle: how having virtualization can make services more secure. I imagined an organization that only runs virtualized desktops and pondered how this would impact patch management. If these desktops get rebooted daily, then patching a single image overnight means that all desktops get the same patched image when they reboot the next day. Fantastic!

But that picture presumes that all users run the same software and is overly optimistic. At best, we have simplified our update procedures to patching just a handful of desktop images and having assurance that they will be the only versions used. And we have created a network rush hour by booting all those virtualized desktops, using networked file servers to do this.

I also wondered about the ability to patch servers by installing the patches to their disk images. If the disk images are not in use, this is simple to do. When the images are being used by a guest, the issue is similar to patching any mounted and in-use file system. Binaries and libraries that are currently loaded cannot be overwritten, but there are tricks, such as renaming the binary, that can be used. I will have more to say about disk images later.

One of the cool features promised by virtualization is the ability to migrate guest operating systems. Suppose a virtualization host doesn't have the resources to support all of the guests we have installed there? We can simply "migrate" that system, even without shutting it down! Although this sounds really cool, consider that we are migrating an entire system over the network. In my darkest thoughts, I have installed a network sniffer and seen not only the entire guest but also the contents of kernel memory, including any cached credentials, as the system gets migrated. I suspect that encryption of guests as they are being migrated is on the drawing boards of virtualization providers, but that is the least of the issue.

One of the cool features of Xen and VMware is that they do use disk images. You can download these images from the Internet or build them yourself. If you need to load-balance a Web service by adding a new server, you just point the guest at the image you prepared earlier and fire it up. Let's ignore for the moment the notion that you may have created the disk image months earlier and not patched it since, as you need to spin it up now. And what about the disk image itself?

Guest disk images have the marvelous property that they can be mounted and manipulated just like any other file system. But there is a large difference here, in that when you mount a disk image, the access controls that were present under the guest host no longer apply. If you can mount the disk image, you are root (or an administrator) and now have total access. This really is no different from having root access to a file server that contains sensitive data or one that is used for network booting of systems. But it does mean that all these same problems exist in the virtualized world.

When I was learning about Xen, I made a mistake in editing the /etc/fstab file that prevented a guest from booting (a change in the name of the swap device). I could have started over and rebuilt the Xen guest, but that would have taken me many hours and could result in unfixing things I had already fixed, or the introduction of new mistakes. Instead, I figured out how to use the losetup command and loop devices to mount the image and edited /etc/fstab. I've done this with VMware images as well [2].

This useful ability to mount disk images implies that it can be used by attackers as well. Access to the root domain, where guest images may be stored, means access to everything that appears within those images as well. The hypervisor also has access to the memory granted to guest images, so there really are no secrets that are not available to the hypervisor. Running guest images is akin to running software within a debugger and all that that implies (see Chow et al. [3] for an example).

Lineup

I really don't want to convince you that running virtualized servers is not a good idea. I think it is a wave of the future, appearing now, and it is pretty unstoppable. What I do want to do is suggest that you don't "drink the Kool-Aid" that hypes that idea that virtualization is more secure than isolated servers. Virtualization is not more secure, and it cannot currently be more secure. Perhaps someday we will have hardware that includes real support for isolating guests, but that day has not arrived yet (and appears to be uncomfortably far in the future, beyond the five-year horizon). You can and should use virtualization and you must be aware of the added vulnerabilities in doing so.

In that vein, Wenjin Hu, Todd Deshane, and Jeanna Matthews, who are among the authors of *Running Xen*, offer us a great explanation and comparison of the virtualization possibilities available in Solaris 10. Not only do they compare these, but they also define the different types of virtualization possible in a way that will help you understand similar technologies under Linux or other operating systems. You can also find a book review of *Running Xen* in this issue.

Next up, Edward Walker takes a look at cloud computing clusters. Walker wondered how Amazon's EC2 cloud computing would compare to a dedicated research cluster in terms of performance, and his benchmarks may surprise you.

Alva Couch then considers how the laws of thermodynamics apply to sysadmin. Couch writes about transforming problems through the use of virtualization and explains the tradeoffs involved in so doing.

Robert Solomon presents a case study in setting up Asterisk. Solomon had set up simpler Asterisk VoIP systems before, but this installation replaces an aging proprietary one for a medium-sized office with very specific requirements. He explains the hardware as well as the Asterisk tweaks necessary to perform an all-page and to unlock the front office door.

Brad Knowles explains how best to populate your network with your own NTP servers. NTP will not work well if you configure NTP servers as you would other servers. Getting the most efficient setup from the perspective of network traffic and server load is an interesting challenge, as is choosing the right hardware. In this issue Knowles also gives us a review of *The Book of IMAP*.

Sandeep Sahore shares his cfsize program. Sahore wondered why there weren't UNIX applications for decreasing the size of files without first splitting them or truncating them to zero length, and cfsize is his answer to these problems.

Jason Dusek examines the problems with concurrency. Dusek became intrigued by the mistake of conflating parallelism with concurrency, and he digs deeply into why concurrency is both a difficult and a currently critical problem.

David Blank-Edelman explains some of the tools you can use in Perl for handling MIME attachments, offering some concrete examples. Pete Galvin continues his thread, started in the August issue, about system analysis. In this issue, Galvin focuses on Solaris-specific tools that help in analyzing problems. Dave Josephsen then encourages us to create Event Brokers for Nagios, providing us with a great example of his own. We have Nick Stoughton explaining the role the USENIX Association (that is, you) has in certain standards bodies. Doing this work requires funding, most of it just to cover travel expenses, and we need to understand this role and decide whether the organization should continue to support it. I certainly think we should. Nick's discussion is followed by more pages than ever of book reviews.

Finally, we have conference reports. The 2008 USENIX Annual Technical Conference is covered in great detail, followed by reports on Hot Topics in Autonomic Computing and on Storage and File Systems Benchmarking.

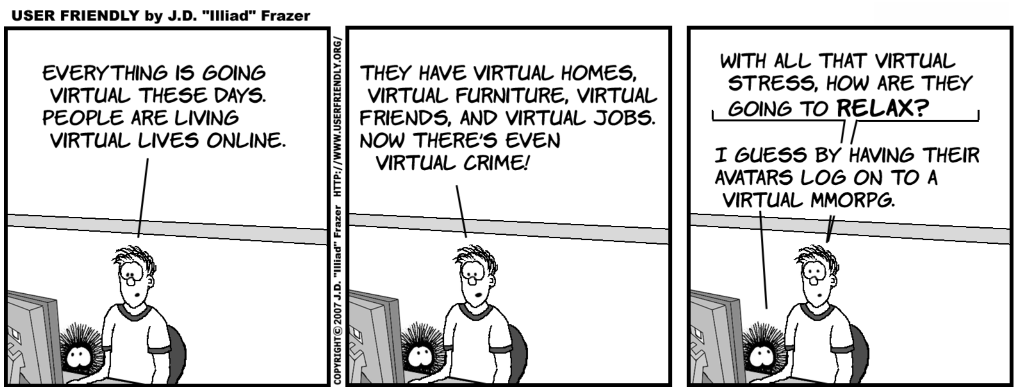
You may have noticed that a lot of this issue is devoted to virtualization. Virtualization is hot, useful, and important, yet, as I suggested above, it comes with its own share of security problems. Most of these are not new. All that I ask is that you remain aware that adding another abstraction layer to an already deep software stack won't make security problems vanish. Instead, simple arithmetic suggests that these problems can only increase.

REFERENCES

[1] VMware security advisories: http://www.vmware.com/security/advisories/.

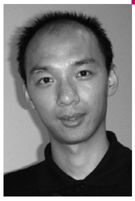
[2] Mounting VMware disk images under Linux: http://legroom.net/2007/08/05/how-mount-vmware-disk-images-under-linux; http://www.cromoteca.com/en/blog/mountflatvmwarediskimagesunderlinux/index.html.

[3] Jim Chow, Tal Garfinkel, and Peter M. Chen, "Decoupling Dynamic Program Analysis from Execution in Virtual Environments," *Proceedings of the 2008 USENIX Annual Technical Conference*, pp. 1–14: http://www.usenix.org/events/usenix08/tech/chow.html.


USER FRIENDLY by J.D. "Illiad" Frazer
EVERYTHING IS GOING VIRTUAL THESE DAYS. PEOPLE ARE LIVING VIRTUAL LIVES ONLINE.
COPYRIGHT © 2007 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
THEY HAVE VIRTUAL HOMES, VIRTUAL FURNITURE, VIRTUAL FRIENDS, AND VIRTUAL JOBS. NOW THERE'S EVEN VIRTUAL CRIME!
WITH ALL THAT VIRTUAL STRESS, HOW ARE THEY GOING TO RELAX?
I GUESS BY HAVING THEIR AVATARS LOG ON TO A VIRTUAL MMORPG.

WENJIN HU, TODD DESHANE, AND JEANNA MATTHEWS

Solaris virtualization options



Wenjin Hu is a PhD student in Computer Science at Clarkson University. He focuses on applying virtualization and file system techniques to provide security and reliability on the desktop.

huwj@clarkson.edu

Todd Deshane is a PhD student in Engineering Science at Clarkson University. His current research deals with using virtualization to provide security on the desktop.

deshantm@clarkson.edu

Jeanna Neefe Matthews is an associate professor of Computer Science at Clarkson University in Potsdam, NY, where she leads an incredible team of students. She is currently on sabbatical and working with VMware in Cambridge, MA.

jnm@clarkson.edu

THE VIRTUALIZATION OPTIONS FOR Solaris have been expanding rapidly. In this article we discuss three types of virtualization systems available in OpenSolaris for x86: Solaris Containers, Solaris xVM, and Sun xVM VirtualBox. We include instructions on how to deploy each solution and a detailed comparison of the three to help system administrators and virtualization fans alike choose the appropriate virtualization technology for their needs. Even if you don't use Solaris, we do explain the differences among OS-level virtualization, paravirtualization, and full-virtualization clearly.

Solaris has included Containers (also called Zones) since Solaris 10 was released in 2005. Containers are an operating-system-level virtualization facility, meaning that the OS itself provides the illusion of multiple independent systems, each with its own IP address and file system, all based on the same base kernel. More recently, support for paravirtualization in the form of Xen (called Sun xVM on Solaris) has been added and now, with the acquisition of VirtualBox, full virtualization on Solaris is also an option. Unlike OS-level virtualization, paravirtualization and full virtualization both offer the ability to run guest operating systems that are different from the underlying OS. Full virtualization can run unmodified operating systems, whereas paravirtualization requires targeted changes to the hardware interface and therefore corresponding changes in the OS source code. As a result, proprietary operating systems such as Microsoft Windows can typically only be run on virtualization systems that support full virtualization. Some virtualization systems, such as Xen, require hardware support for virtualization, such as Intel VT or AMD-V, to support full virtualization.

Getting Started with Solaris Containers

In Solaris Containers/Zones, a virtual machine is called a *zone* and a zone with resource limitations is called a *container*. The basic command to operate a zone's configuration file is `zonecfg –z newzone`. This will bring up a shell in which you can issue a variety of commands to manipulate the specified zone. As shown in Listing 1, you `create` a zone, add the attached devices such as `add net`, set zone options such as `set autoboot=true`, display the configuration, `verify` the configuration, and finally `commit` a zone's resources, which writes out a final configuration file for the zone. You can also use zonecfg to browse the characteristics of an existing zone and modify it as desired.

```
# zonecfg -z newzone
newzone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:newzone> create
zonecfg:newzone> set zonepath=/export/home/newzone
zonecfg:newzone> set autoboot=true
zonecfg:newzone> add net
zonecfg:newzone:net> set address=192.168.16.109
zonecfg:newzone:net> set physical=pcn0
zonecfg:newzone:net> end
zonecfg:newzone> verify
zonecfg:newzone> commit
zonecfg:newzone> exit
```

LISTING 1: THE STEPS SHOWING THE ZONE CONFIGURATION

Solaris provides a number of options to manage the resources that a zone can own, including the CPU, the memory, and the number of the processes. This is the heart of making a zone into a container with resource limitations. In Listing 2, we illustrate how to add a variety of restrictions to our new zone. (These commands are also issued at the zonecfg prompt.) The capped-cpu command limits the CPU cycles assigned to the zone to a fourth of one CPU. Similarly, our capped-memory command assigns the zone 128 MB of memory and 512 MB of swap space. It also guarantees that 64 MB of the zone's memory will be resident in physical memory at all times. Finally, the set max-lwps command illustrates how we can place limits on things besides physical resources. It limits the number of light-weight processes in a running zone and is useful for preventing problems such as fork-bombs from taking down the whole machine.

```
add capped-cpu
   set ncpus=0.25
end

add capped-memory
   set physical=128M
   set swap=512M
   set locked=64M
end

set max-lwps=175
```

LISTING 2: THE OPTIONS LIMITING THE CONTAINER'S RESOURCES

Once our new zone is configured, we are ready to instantiate it with the command zoneadm –z newzone install and then run it with the command zoneadm –z newzone boot. zoneadm can also be used for other zone administration functions such as listing the configured zones, installing or uninstalling the zones, and booting, pausing, and halting installed zones. Listing 3 shows the output of running zoneadm list after installing our new zone. The parameter –c will display all the configured zones' information; –v will display the detailed information of zones.

```
# zoneadm list -vc
ID  NAME     STATUS   PATH                  BRAND   IP
0   global   running  /                     native  shared
1   newzone  running  /export/home/newzone  native  shared
```

LISTING 3: A ZONEADM LIST SHOWING ALL ZONES

Finally, we are ready to log in to our new zone with the command `zlogin newzone`. It will be running the same version of OpenSolaris as the base operating system, as is required with OS-level virtualization. However, the zone will behave like a separate system, with its own IP address, its own file system, and its own set of installed applications.

Getting Started with Solaris xVM

A virtual machine in xVM is called a *domain*, just as it is in other Xen implementations. Domains are divided into two major categories: paravirtualized (PV) domains and Hardware-assisted Virtual Machine (HVM) domains. In a PV domain, the hardware abstraction presented to the guest VM is not identical to the underlying physical hardware; instead, strategic changes are made to make the system easier to virtualize. To run properly on the paravirtualized hardware, the guest operating system must be modified to be aware of these changes. This requires source-code-level changes to the OS and is therefore rarely available for proprietary operating systems such as Windows. Two common choices for paravirtualized guest domains are Solaris and Linux.

In an HVM domain, the hardware abstraction presented to the guest is identical to the underlying hardware, so any operating system that runs on x86 hardware architecture can be installed. To make this possible, an HVM domain relies on special hardware support for virtualization such as Intel V-T or AMD-V. If your system does not have this hardware support for virtualization, then paravirtualized domains are your only option.

Xen is a virtual machine monitor, called a hypervisor, which intercepts the guest domain's system calls. It is necessary to boot the Xen-enabled Solaris kernel on the physical machine rather than a normal Solaris kernel. Since Solaris Nevada Version build 75, Xen has been developed and well integrated into Solaris Express Community Edition through a variety of boot options in the GRUB menu. For example, the standard GRUB menu displays three choices: Solaris Express Community Edition snv87 X86, Solaris xVM, and Solaris Failsafe; the second grub option, Solaris xVM, should be chosen.

Xen also relies on a special, privileged domain, called Domain0, and the Xen control daemon, Xend, for communication between the hypervisor and the guests. Domain0 is granted the full privileges of managing the guest domains and the physical devices by the Xen hypervisor, similar to a "normal" Solaris OS instance.

Device drivers can also be fully virtualized or paravirtualized. Even a system that does full virtualization of the CPU and memory can load paravirtualized drivers to handle external devices such as a disk or network interface. In a paravirtualized driver, the driver running in the guest operating system is aware of the hypervisor and explicitly participates in communicating its requests to domain0 where the real physical device drivers are running. In a fully virtualized driver, the real device access still occurs on the Domain0 drivers, but the guest driver is unaware of this, so the hypervisor must trap accesses to I/O space or DMA operations in order to forward them on to the proper device driver in Domain0. PV drivers have much lower overhead, because they avoid this expensive process of trapping and forwarding.

Our first step in running Xen guests is to make sure that the hypervisor, Domain0, and Xend are all running. After Solaris xVM boots up, you can use the command `virsh list` as shown in Listing 4 to check whether Domain0 is running.

```
# virsh list
   Id       Name            State
---------------------------------------------------
    0       Domain-0        running
  110       newSolaris      blocked
```

LISTING 4: VIRSH LIST SHOWING DOMAIN0

Next, we check that Xend is running and specify the default network interface in Xend first so that our guest domains are able to set up the virtual network based on that physical NIC. Xend is wrapped as an application service in Solaris. Listing 5 illustrates the use of `svccfg` and `svcadm` facilities to configure and restart Xend. The `–s` parameter specifies the Xend service; `setprop` specifies the Xend configuration options. After making this (or any change) to the Xend configuration, you can apply the change by refreshing and restarting Xend as shown. The `svcadm` facility can be used to `enable`, `disable`, `refresh`, or `restart` the Xend service at any time.

```
# svccfg –s xvm/xend setprop config/default-nic="bge0"
# svcadm refresh xvm/xend
# svcadm restart xvm/xend
```

LISTING 5: THE STEPS FOR CONFIGURING AND RESTARTING XEND

With that, we are ready to create a new guest domain. In this section we will show three primary examples: a PV Solaris guest domain, a PV Linux domain, and an HVM Solaris domain. In Solaris, virt-install is a tool used to create the guest domain images regardless of whether it is PV or HVM. For example, a paravirtualized Solaris image can be created with the following command:

```
# virt-install --nographics -n newSolaris --paravirt -f /export/home/newSolaris.\
img -r 1024 -s 30 -l /export/home/sol-nv-b87.iso
```

where `-n` is for specifying a domain name to be newSolaris, `--paravirt` is for selecting the mode to be paravirtualized, `-f` is for specifying the domain image name newSolaris.img in the path /export/home, `-r` is for assigning the domain memory size to be 1024 MB, `-s` is for creating the domain image size in gigabytes, and -l is for choosing the installation location. Note that in Solaris xVM, for paravirtual guests (both Solaris and Linux), the video card and CD-ROM drivers are not yet fully ported, will not have a graphical window, and cannot use fully use the CD-ROM. However, we were able to use an ISO file as the guest CD-ROM during the install of the paravirtual Solaris guest by using the –l option to specify the ISO location. But for the HVM guest, the guest can fully use a standard CD-ROM driver within it and then have a CD-ROM device.

We can also install the guest via a Solaris NFS share on Solaris xVM. If we use ZFS volumes for the guest disk storage, it should have better performance and more reliability than using a file-based image in the Domain0 file system.

After running this command and finishing the normal Solaris Installation process, you can use `virsh list` again to see the newSolaris guest domain running, as illustrated in Listing 4. To access the running domain newSolaris, we can use the command `virsh console newSolaris`. To get out of the guest domain, the combination key Ctrl+] is needed.

If we wish to save the guest domain's configuration file for later use, we can use the virsh tool to write an XML format configuration file when the guest domain is running, with `virsh dumpxml newSolaris > newSolaris.xml`. With the guest domain's xml configuration file, we can directly boot the image with the command virsh create newSolaris.xml. Furthermore, we can use the command virsh shutdown newSolaris to turn off the guest.

Next, we will walk through an example of installing an HVM guest domain. Before creating the Solaris HVM image, we may need to enable VNC in Xend, as Listing 6 shows. In the svccfg command shown, 0.0.0.0 indicates that the VNC server is listening to any IP and passwd should be replaced with the actual password. If you don't explicitly set the password, it will default to the empty string, which is, of course, not secure. If you don't want to use VNC for remote access to the guest at all, you can use the command svccfg -s xvm/xend delprop config/vnc-listen to remove that option and then refresh and restart Xend as we did earlier.

```
# svccfg -s xvm/xend setprop config/vnc-listen = astring: \"0.0.0.0\"
# svccfg -s xvm/xend setprop config/vncpasswd = astring: \"passwd\"
```

LISTING 6: VNC SETUP IN XEND

Next, we can use virt-install with the --hvm argument to start an HVM Solaris guest as, for example:

```
# virt-install -n SolarisHVM --hvm -r 1024 --vnc -f /export/home/SolarisHVM.\
img -s 30 -c /export/home/newbie/sol-nv-b87.iso
```

Notice that we use almost the same options as we did when creating the paravirtual Solaris guest; the only differences are --hvm, which specifies the guest works in HVM mode, and --vnc, which specifies that the guest will have a VNC connection. Also, the argument –c is used for specifying the virtual CD-ROM rather than –l when installing the paravirtual guest. In this case, it indicates that the installation is from the ISO file /export/home/sol-nv-b87.iso and, from the guest's perspective, the installation is from the guest CD-ROM. We could install guests based on any other operating system, including Windows, in this same manner.

When you run the virt-install command, it will first display a window asking for a VNC password, which is passwd as we have set up in Xend at the beginning of the HVM guest setup procedure. The rest of the installation process will look exactly like installing Solaris on a real machine. As with the PV guest, we can use the virsh facility to create and shut down the guest domain.

We used virt-install to create Solaris guests and it's also possible to use it to create Linux guests in a similar way. However, for Linux guests there are some additional options. In this section, we will illustrate the use of one of these, virt-manager, to install a Linux guest. Refer to our Web site [1] for additional information on options for Linux guests such as netinstall, isoinstall, and cdrom-install.

Virt-manager is a GUI installation and management tool for guest domains. It provides a GUI tool for the creation of new domains, an integrated VNC viewer for accessing domains, and other useful tools for the management of domains and their resources. Virt-manager is available on newer releases of CentOS and Red Hat Enterprise. It is also available in the latest version of Solaris now. You can find its icon from menu->All Applications->System Tools->Virtual Machine Manager. You can also run it as root with the command virt-manager. It will first display a window asking you to connect to Xen hypervisor. Single-clicking the "Connect" button will show you the Virtual Machine Manager GUI interface displaying the Domain0 and guest domain's running status and resource usage. Click the "New" button and it will guide you step by step through the guest creation wizard. It only supports network installation for paravirtual guests, but both ISO and network installations for HVM guests. Here we give an example for creating a paravirtual CentOS guest.

Virt-manager first asks for the guest domain's name, which later becomes the guest domain id and the type of the guest domain (paravirtualized or fully virtualized). Then, it will request a network path address to the CentOS repository such as http://mirror.clarkson.edu/pub/centos/5/os/i386/. Later, you will be asked to choose a disk partition or a file to be the guest file system. If you have not set up a special disk partition for your new guest, a file is the safest choice. After allocating the memory size and the number of virtual CPUs, it will go through the normal CentOS network installation process. Our Web site [1] includes detailed screenshots of the entire process.

Getting Started with Sun xVM VirtualBox

Since VirtualBox was acquired by Solaris only recently, it is not yet automatically installed in Solaris. So the first step is to download VirtualBox from http://virtualbox.org/wiki/Downloads. To determine which Solaris package to download (32-bit x86 or 64-bit AMD64), you can use the `isainfo` command on your base Solaris system.

The VirtualBox installation package includes two packages: the kernel package, used to install the Virtual Disk Image (VDI) kernel module, and the VirtualBox package, which will install the VirtualBox application and GUI library. Once the proper package is downloaded, use the command `pkgadd` to first install the VirtualBox kernel package and then the VirtualBox package, as illustrated in Listing 7.

```
# pkgadd -G -d VirtualBoxKern-VERSION-OS-BIT.pkg
# pkgadd -G –d VirtualBox-VERSION-OS-BIT.pkg
......(package installing message are partly omitted)......
VirtualBox kernel module unloaded
VirtualBox kernel module loaded.
Creating links...
Done.
Installation of <SUNWvbox> was successful.
```

LISTING 7: THE STEPS FOR INSTALLING VIRTUALBOX

Once the packages are installed, simply issue the command `VirtualBox` so that the VirtualBox management window will pop up.

One important note for trying all the virtualization systems available on Solaris is that VirtualBox and Solaris xVM cannot currently operate at the same time. If you have been running in Solaris xVM, it is necessary to reboot the machine and switch to the first option, Solaris, in the GRUB menu. Otherwise, you will see the error `VirtualBox Kernel Driver not Loaded` when you do run the VirtualBox command, because the VirtualBox kernel has not yet been ported to the Xen kernel.

To create a virtual machine, when you choose the "NEW" button, it will lead you to the VM installation wizard. VirtualBox can run any guest operating system including Windows, Linux, and Solaris, or any x86 OS running on it without any modifications to your guest OS. All of their device drivers will work normally, with no need to port to the guest OS.

First, you need to choose your VM name and its OS type, then assign the memory size to the VM (which for normal Solaris installation requires at least 768 MB). Next, you need to specify the VM disk image. If it is your first time to create a VirtualBox File image, you have to click the "NEW" button. You must also specify a file for the disk image. You can either specify the size of the disk file or set it to grow dynamically over time. A standard Solaris SXCE requires 12 GB for the image file. If you want to further configure the VM's devices, you can click the "Settings" button. One important note is that so far VirtualBox only supports NAT network topology. If you want to install from the CD, choose the CD-ROM option; there you can click to mount the CD drive from the host CD-ROM or from a specific ISO image.

Once done, you can simply click the "Start" button to start the VM. The VM will boot from CD-ROM and the rest of the installation process is the same as normal Solaris installation. When the virtual machine is created and booting, VirtualBox will prompt you with a VNC window to display the screen of the guest VM; you can either click the mouse or press the Enter key to get into the guest VM. If you want to get out of the VM box, you can press the right Ctrl key to release yourself from the VM.

In VirtualBox, if you want to run a previously installed VM, you can operate on the existing vdi files. By default, the VM virtual disk images are stored in the /root/.VirtualBox/VDI directory. There is a virtual disk manager to manage them. You can press the combination key Ctrl+D to pop up the disk manager window. If you already have a system-installed vdi file, you can press the "Add" button to add the existing VM image into the VirtualBox. Then you can go through the previous process of creating a new VM procedure to run a VM. But if you want to remove a VM image from VirtualBox, you need to first `Release` the image and then `Remove` it from the virtual disk manager, because when a disk image is assigned to a VM, VirtualBox automatically registers it and grants a unique uuid to that VM and image.

Comparing Containers, xVM, and VirtualBox

Now that we have shown you the basics of getting started with three different virtualization options on Solaris, in this section we will present some comparisons among them.

One important point of comparison is ease of use. In our opinion, the easiest to use is VirtualBox. It is fully GUI-guided, straightforward, and simple. Solaris Containers are also relatively easy to use, especially because they are so well integrated into Solaris and have such a complete tool chain for configuration and management. However, since they have no GUI interface, they are better suited to server applications than desktop virtualization. Solaris xVM is the most complicated, but as more management tools, such as virt-manager, are extended and integrated into Solaris, the ease of use will improve.

Of course, ease of use is just one part of the story. It is also important to consider the features of each system. Containers can only run Solaris guests, so some common applications of virtualization (e.g., running alternate operating systems) simply won't work in Containers. Solaris xVM requires the running guest to be a modified OS and generic virtual device drivers need to be ported. If the unmodified OS is to run on Solaris xVM, VT or AMD-V hardware support will be needed on the CPU chip. VirtualBox, however, can run any type of unmodified guest even without hardware support for virtualization.

In terms of storage, Solaris Zones can either share files with the global zone or have their own version of files from the global zone. In other words, zones can use the same library files as the global zone or have older or newer versions of libraries than the global zone. You should also be aware that when you change the files in the global zone, you may also affect other zones that are sharing them. For VirtualBox, the system files in a VM of VirtualBox are only used by that VM and will not affect other VMs. Each VM is encapsulated in a separate vdi or vdmk files. However, if sharing is desired, VirtualBox does have a shared folder option that can mount a base OS directory to share with the guest. Similar to VirtualBox, in Solaris xVM each guest's file system is independently separated either by files or partitions or disks. Theoretically, we can dynamically add Domain0's disk or partitions to the guest domain, sharing with the guest domain. But we do not recommend attempting that, because there is no way to maintain the consistency of shared files or file systems. The preferred method of sharing files with xVM guests would be to use a network file server.

For the network topology, in Solaris Zones all zones share the network interface with the global zone's network interface in a bridged mode. There are no other network topology choices. Bridging is also currently the only option available for Solaris xVM guest domains. The routing and NAT topologies that are available to Xen on Linux are still in development for Solaris. VirtualBox supports only the NAT topology. This means that there is no way for an outsider to directly access the VirtualBox VM through the network. This is a crucial difference for running server VMs.

It is worth noting that, with the Crossbow project, changes in networking support should be coming for all virtualization systems. Crossbow is a Solaris network virtualization and flow control solution. It provides universal network architecture to the virtualization systems described here (Containers, xVM, and VirtualBox) to manage the flow control of those virtual NICs, such as bandwidth and packet types. Crossbow is not yet stable but is being tested as part of SNV build 91.

Finally, a critical aspect of the comparison is performance overhead from virtualization. A full performance comparison on various types of hardware and running a wide variety of tests is beyond the scope of this article. Here we present the results of some simple compilation tests on baseline Solaris and on each of the virtualization systems. Specifically, we report the time to compile the Apache Web server.

All our tests are run on Open Solaris Community Express Nevada build 87 (SNV b87) running on a Dell Optiplex (Intel-VT dual-core 2.4-GHz 6600 CPU, with 4 GB memory, a 250-GB disk, and 1-Gb NIC). The guest resource allocation can be seen in Table 1. Zone is a virtual machine in Solaris, Xen domU is the virtual machine in Solaris xVM, and VBox VM is the virtual machine running in VirtualBox.

Container is a zone with resource controls. Here the container is assigned the limited CPU to be 1 ncpus. If we use `zonecfg –z newzone info`, we can see the information in Listing 8. For more complicated resource configuration, you can look at the usage of project and task facilities.

	CPU (dual)	***Memory***	***Image size***	***Network***
Zone	-	-	-	Bridged
Container	1	1024M	-	Bridged
Xen domU	1	1024M	30G	Bridged
Vbox VM	1	1024M	30G	NAT

TABLE 1: VM RESOURCE ALLOCATIONS FOR EACH VIRTUALIZATION SYSTEM

```
# zonecfg –z newzone info
    capped-cpu:
      [ncpus: 1]
    capped-memory:
      physical: 1G
      [swap: 1G]
      [locked: 768M]
    rctl:
      name: zone.cpu-cap
      value: (priv=privileged,limit=50,action=deny)
    rctl:
      name: zone.max-swap
      value: (priv=privileged,limit=1073741824,action=deny)
    rctl:
      name: zone.max-locked-memory
      value: (priv=privileged,limit=805306368,action=deny)
    rctl:
      name: zone.max-lwps
      value: (priv=privileged,limit=200,action=deny)
```

LISTING 8: THE RESOURCE LIMITATION FOR THE BENCHMARKED CONTAINER

Figure 1 shows the relative overhead of the four virtual machines by a percentage of the baseline time to compile httpd. Overall, the zone has the least overhead compared to the baseline, because it has full access to the whole global zone's resources. Its performance is almost as good as the baseline. The container experiences delay because it is limited to half of the overall CPU cycles. The Xen guest domain is close to the overhead of the container, but it consumes substantially more system time. VirtualBox clearly has the highest overhead (250%). Note that, in our experiment on the same hardware, the overhead of Xen on Linux is less. In general, the overhead of Xen on Solaris is not necessarily the same as Xen on Linux.

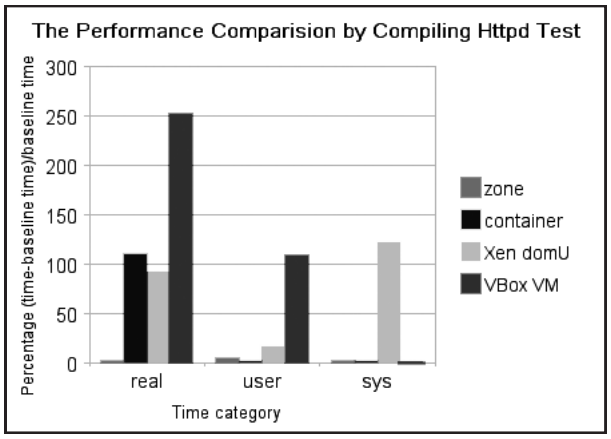


FIGURE 1: PERFORMANCE COMPARISON BY PERCENTAGE AGAINST BASE SOLARIS SYSTEM COMPILE OF APACHE

In Figure 2, we find that the container's sys and user time are almost the same as the zone's. But, overall, the container's total consumed time is almost doubled, which indicates that Solaris resource management is effectively giving the container a limited share of system resources. We recommend that system administrators use resource management facilities to avoid some zones' malicious or greedy resource usage and effect on the overall performance of other zones. Although the configurations may be a little complicated, it is worth taking the time to get it right. One successful experiment involves running a memory bomb (a loop constantly allocating and touching additional memory) in the zone and container: The zone leaves the global zone dead, but although the container suffers from running out of memory, the global zone is still alive and works well. For more details, refer to our previous paper [2].

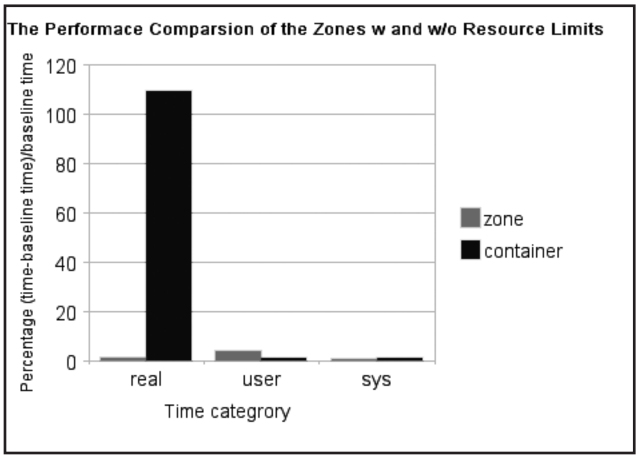


FIGURE 2: COMPARING A RESOURCE-LIMITED CONTAINER TO A ZONE

Conclusion

Overall, Solaris has offered us a variety of virtualization systems to use: Solaris Containers, Solaris xVM, and Sun xVM VirtualBox. Each of these has its own unique advantages. Sun xVM VirtualBox offers full virtualization, is straightforward to use, and has nice GUI windows, but its performance overhead is also high and, with an NAT-only network, running servers is difficult. Still, for easy-to-use desktop virtualization on Solaris, VirtualBox is probably the best choice. In contrast, Solaris Containers/Zones OS-level virtualization is targeted at server-level usage. It achieves good performance, but to make it work properly you need to master the resource management control tools, which can be somewhat complicated. Containers/Zones also do not give you a choice of guest operating systems. For fast Solaris servers, they are likely the best choice. However, if you want a choice of guest operating systems and good performance, then Solaris xVM is likely to be the best choice. Its performance is comparable with OS-level virtualization, and it is suitable for both desktop usage and server usage. Solaris xVM can be a bit complicated to configure, but there are a variety of configuration options, from GUI to command line, and the available tools continue to improve.

REFERENCES

[1] Clarkson Web site for screenshots and documents: http://www.clarkson.edu/projects/virtualization/solaris/login08.

[2] Quantifying the Performance Isolation Properties of Virtualization Systems: http://people.clarkson.edu/~jnm/publications/isolation_ExpCS _FINALSUBMISSION.pdf.

PRACTICAL RESOURCES

[1] Wenjin's blog on Solaris Virtualization Tutorials: http://deepenintocs.blogspot.com/.

[2] Xen introduction and tutorial: http://runningxen.com/.

[3] Solaris Containers online documentation: http://www.sun.com/bigadmin/content/zones/.

[4] Solaris xVM: http://opensolaris.org/os/community/xen/.

[5] VirtualBox user manual: http://www.virtualbox.org/wiki/End-user_documentation.

[6] Crossbow project: http://opensolaris.org/os/project/crossbow/.

EDWARD WALKER

benchmarking Amazon EC2 for high-performance scientific computing

Edward Walker is a Research Scientist with the Texas Advanced Computing Center at the University of Texas at Austin. He received his PhD from the University of York (UK) in 1994, and his research interests include designing fault-tolerant distributed systems, HPC programming languages, and user-centric operating/run-time systems.

ewalker544@gmail.com

Benchmark results can be downloaded from http://www.usenix.org/publications/login/2008-10/benchmark_results.tgz.

HOW EFFECTIVE ARE COMMERCIAL cloud computers for high-performance scientific computing compared to currently available alternatives? I aim to answer a specific instance of this question by examining the performance of Amazon EC2 for high-performance scientific applications. I used macro and micro benchmarks to study the performance of a cluster composed of EC2 high-CPU compute nodes and compared this against the performance of a cluster composed of equivalent processors available to the open scientific research community. My results show a significant performance gap in the examined clusters that system builders, computational scientists, and commercial cloud computing vendors need to be aware of.

The computer industry is at the cusp of an important breakthrough in high-performance computing (HPC) services. Commercial vendors such as IBM, Google, Sun, and Amazon have discovered the monetizing potential of leasing compute time on nodes managed by their global datacenters to customers on the Internet. In particular, since August 2006, Amazon has allowed anyone with a credit card to lease CPUs with their Elastic Compute Cloud (EC2) service. Amazon provides the user with a suite of Web-services tools to request, monitor, and manage any number of virtual machine instances running on physical compute nodes in their datacenters. The leased virtual machine instances provide to the user a highly customizable Linux operating system environment, allowing applications such as Web hosting, distributed data analysis, and scientific simulations to be run. Recently, some large physics experiments such as STAR [1] have also experimented with building virtual-machine-based clusters using Amazon EC2 for scientific computation. However, there is a significant absence of quantitative studies on the suitability of these cloud computers for HPC applications.

It is important to note what this article is not about. This is not an article on the benefits of virtualization or a measurement of its overhead, as this is extensively covered elsewhere [2]. This is also not an article evaluating the counterpart online storage service Amazon S3, although a quantitative study of this is also critical. Finally, this is not an article examining the cost benefits of using cloud computing in IT organizations, as this is amplified elsewhere by its more eloquent advocates [3].

Instead, this article describes my results in using macro and micro benchmarks to examine the “delta” between clusters composed of currently available state-of-the-art CPUs from Amazon EC2 versus clusters available to the HPC scientific community circa 2008. My results were obtained by using the NAS Parallel Benchmarks to measure the performance of these clusters for frequently occurring scientific calculations. Also, since the Message-Passing Interface (MPI) library is an important programming tool used widely in scientific computing, my results demonstrate the MPI performance in these clusters by using the mpptest micro benchmark. The article provides a measurement-based yardstick to complement the often hand-waving nature of expositions concerning cloud computing. As such, I hope it will be of value to system builders and computational scientists across a broad range of disciplines to guide their computational choices, as well as to commercial cloud computing vendors to guide future upgrade opportunities.

Hardware Specifications

In our performance evaluation, we compare the performance of a cluster composed of EC2 compute nodes against an HPC cluster at the National Center for Supercomputing Applications (NCSA) called Abe. For this benchmark study we use the high-CPU extra large instances provided by the EC2 service. A comparison of the hardware specifications of the high-CPU extra large instances and the NCSA cluster used in this study is shown in Table 1. We verified from information in /proc/cpuinfo in the Linux kernel on both clusters that the same processor chip sets were used in our comparison study: dual-socket, quad-core 2.33-GHz Intel Xeon processors.

	EC2 High-CPU Cluster	**NCSA Cluster**
Compute Node	7 GB memory, 4 CPU cores per processor (2.33-GHz Xeon), 8 CPU per node, 64 bits, 1690 GB storage	8 GB memory, 4 CPU cores per processor (2.33-GHz Xeon), 8 CPU per node, 64 bits, 73 GB storage
Network Interconnect	High I/O performance (specific interconnect technology unknown)	Infiniband switch

TABLE 1. HARDWARE SPECIFICATIONS OF EC2 HIGH-CPU INSTANCES AND NCSA ABE CLUSTER.

NAS Parallel Benchmark

The NAS Parallel Benchmarks (NPB) [4] comprise a widely used set of programs designed to evaluate the performance of HPC systems. The core benchmark consists of eight programs: five parallel kernels and three simulated applications. In aggregate, the benchmark suite mimics the critical computation and data movement involved in computational fluid dynamics and other “typical” scientific computation. A summary of the characteristics of the programs for the Class B version of NPB used in this study is shown in Table 2.

The benchmark suite comes in a variety of versions, each using different parallelizing technologies: OpenMP, MPI, HPF, and Java. In this study we use the OpenMP [5] version to measure the performance of the eight-CPU single compute node. We also use the MPI [7] version to characterize the distributed-memory performance of our clusters.

Program	Description	Size	Memory (Mw)
EP	Embarrassingly parallel Monte Carlo kernel to compute the solution of an integral.	230	18
MG	Multigrid kernel to compute the solution of the 3-D Poisson equation.	2563	59
CG	Kernel to compute the smallest eigenvalue of a symmetric positive definite matrix.	75000	97
FT	Kernel to solve a 3-D partial differential equation using an FFT-based method.	512 × 2562	162
IS	Parallel sort kernel based on bucket sort.	225	114
LU	Computational fluid dynamics application using symmetric successive over-relaxation (SSOR).	1023	122
SP	Computational fluid dynamics application using the Beam-Warming approximate factorization method.	1023	22
BT	Computational fluid dynamics application using an implicit solution method.	1023	96

TABLE 2. NPB CLASS B PROGRAM CHARACTERISTICS

NPB-OMP VERSION

We ran the OpenMP version of NPB (NPB3.3-OMP) Class B on a high-CPU extra large instance and on a compute node on the NCSA cluster. Each compute node provides eight CPU cores (from the dual sockets), so we allowed the benchmark to schedule up to eight parallel threads for each benchmark program. On the NCSA cluster and EC2, we compiled the benchmarks using the Intel compiler with the option flags "-openmp -O3."

Figure 1 shows the runtimes of each of the programs in the benchmark. In general we see a performance degradation of approximately 7%–21% for the programs running on the EC2 nodes compared to running them on the NCSA cluster compute node. This percentage degradation is shown in the overlaid line-chart in Figure 1. This is a surprising result; we expected the performance of the compute nodes to be equivalent.

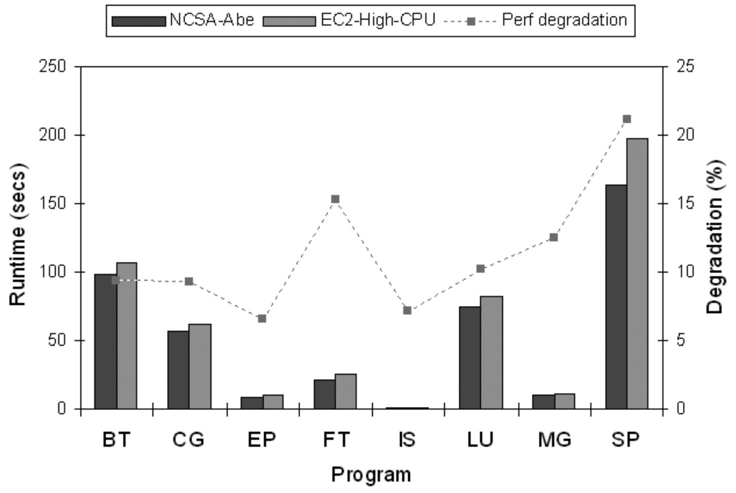


FIGURE 1. NPB-OMP (CLASS B) RUNTIMES ON 8 CPUS ON EC2 AND NCSA CLUSTER COMPUTE NODES. OVERLAID IS THE PERCENTAGE PERFORMANCE DEGRADATION IN THE EC2 RUNS.

NPB-MPI VERSION

We ran the MPI version of NPB (NPB3.3-MPI) Class B on multiple compute nodes on the EC2 provisioned cluster and on the NCSA cluster. For the EC2 provisioned cluster, we requested 4 high-CPU extra large instances, of 8 CPUs each, for each run. On both the EC2 and NCSA cluster compute nodes, the benchmarks were compiled with the Intel compiler with option flag `-O3`. For the EC2 MPI runs we used the MPICH2 MPI library (1.0.7), and for the NCSA MPI runs we used the MVAPICH2 MPI library (0.9.8p2). All the programs were run with 32 CPUs, except BT and SP, which were run with 16 CPUs.

Figure 2 shows the run times of the benchmark programs. From the results, we see approximately 40%–1000% performance degradation in the EC2 runs compared to the NCSA runs. Greater then 200% performance degradation is seen in the programs CG, FT, IS, IU, and MG. Surprisingly, even EP (embarrassingly parallel), where no message-passing communication is performed during the computation and only a global reduction is performed at the end, exhibits approximately 50% performance degradation in the EC2 run.

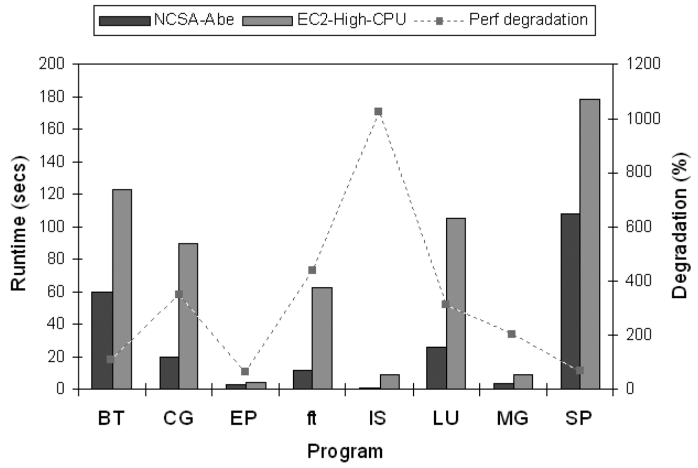


FIGURE 2. NPB-MPI (CLASS B) RUNTIMES ON 32 CPUS ON THE NCSA AND EC2 CLUSTER. BT AND SP WERE RUN WITH 16 CPUS ONLY. OVERLAID IS THE PERCENTAGE DEGRADATION IN THE EC2 RUNS.

We hypothesize that the Infiniband switch fabric in the NCSA cluster is enabling much higher performance for NPB-MPI. However, we want to quantitatively understand the message-passing performance difference between using a scientific cluster with a high-performance networking fabric and a cluster simply composed of Amazon EC2 compute nodes. The following results use the mpptest benchmark [5] to characterize the message-passing performance in the two clusters.

The representative results shown in this article are from the bisection test. In the bisection test, the complete system is divided into two subsystems, and the aggregate latency and bandwidth are measured for different message sizes sent between the two subsystems. In the cases shown, we conducted the bisection test using 32-CPU MPI jobs.

Figures 3 and 4 show the bisection bandwidth and latency, respectively, for MPI message sizes from 0 to 1024 bytes. It is clearly seen that message-passing latencies and bandwidth are an order of magnitude inferior between EC2 compute nodes compared to between compute nodes on the NCSA cluster. Consequently, substantial improvements can be provided to the HPC scientific community if a high-performance network provisioning solution can be devised for this problem.

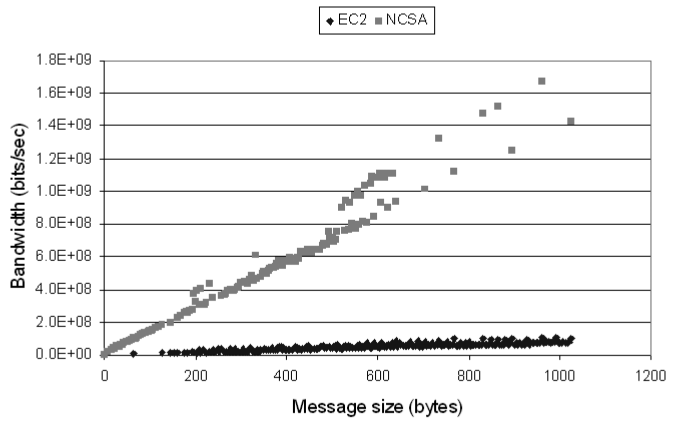


FIGURE 3. MPI BANDWIDTH PERFORMANCE IN THE MPPTEST BENCHMARK ON THE NCSA AND EC2 CLUSTERS

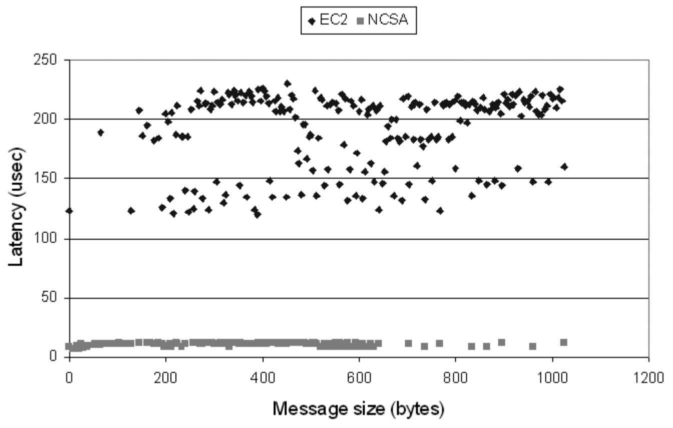


FIGURE 4. MPI LATENCY PERFORMANCE IN THE MPPTEST BENCHMARK ON THE NCSA AND EC2 CLUSTERS

Conclusion

The opportunity of using commercial cloud computing services for HPC is compelling. It unburdens the large majority of computational scientists from maintaining permanent cluster fixtures, and it encourages free open-market competition, allowing researchers to pick the best service based on the price they are willing to pay. However, the delivery of HPC performance with commercial cloud computing services such as Amazon EC2 is not yet mature. This article has shown that a performance gap exists between performing HPC computations on a traditional scientific cluster and on an EC2 provisioned scientific cluster. This performance gap is seen not only in the MPI performance of distributed-memory parallel programs but also in the single compute node OpenMP performance for shared-memory parallel programs. For cloud computing to be a viable alternative for the computational science community, vendors will need to upgrade their service offerings, especially in the area of high-performance network provisioning, to cater to this unique class of users.

REFERENCES

[1] The STAR experiment: http://www.star.bnl.gov/.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *ACM Symposium on Operating System Principles*, 2003.

[3] Simson Garfinkel, "Commodity Grid Computing with Amazon's S3 and EC2", *;login:*, Feb. 2007.

[4] NAS Parallel Benchmarks: http://www.nas.nasa.gov/Resources/Software/npb.html.

[5] OpenMP specification: http://openmp.org.

[6] Message-Passing Interface (MPI) specification: http://www.mpi-forum.org/.

[7] mpptest—Measuring MPI Performance: http://www-unix.mcs.anl.gov/mpi/mpptest/.

ALVA L. COUCH

system administration thermodynamics

Alva Couch is an Associate Professor of Computer Science at Tufts University, where he and his students study the theory and practice of network and system administration. He served as Program Chair of LISA '02 and was a recipient of the 2003 SAGE Outstanding Achievement Award for contributions to the theory of system administration. He currently serves as Secretary of the USENIX Board of Directors.

couch@cs.tufts.edu

VIRTUALIZATION PROVIDES SEVERAL ways to transform the question "Why does this fail?" into the related question "Is this fast enough?"

Fellow system administrators, do you find yourself troubleshooting systems more and enjoying it less? Do you spend most of your time correcting the "same old problems"? Are legacy systems millstones around your neck? Then, from what I can tell, you are like most system administrators. For those of you in this situation, I have a controversial message: *The troubleshooting you are doing now is already obsolete.*

In the following, I will outline techniques for minimizing common kinds of trouble by use of virtualization. Most of these techniques are common knowledge, and I apologize in advance for stating the obvious. But, in my experience, many system administrators of good faith and stronger character than my own still endure these various tribulations. This article is written for them because I think they remain in the majority.

No strategy I am going to suggest actually eliminates trouble. Instead, trouble is *transformed* into a hopefully more manageable form. Virtualization allows one to replace configuration troubleshooting with performance troubleshooting. One key to understanding this transformation is to consider it as part of the "thermodynamics of system administration." System administrators, like mechanical engineers and physicists, have to cope with conservation laws, and one thing that is conserved is trouble. We cannot eliminate trouble, but we can make choices that transform it into a perhaps more manageable (and hopefully "user-friendly") form.

The Three Laws of Thermodynamics

Trouble is a form of entropy, and thus it is subject to the laws of thermodynamics. Ginsberg once described the three laws of thermodynamics as "One can't win, one can't break even, and one can't get out of the game." In system administration terms, we might restate these laws as follows:

- There is no way to prevent trouble.
- There is no zero-cost way of transforming trouble into other forms.
- Trouble approaches zero only as system use approaches zero.

The theme of this article is the second law. In system administration, as in thermodynamics, one can, by applying some energy, transform trouble to a (hopefully) "more convenient" form.

As a physical analogy, suppose that you are careening down a hill at high velocity toward an obstacle. To mitigate this, you can apply a brake, but the action of applying a brake has its own problems, including heat buildup. Your action can transform the problem of careening down the hill into the problem of controlling the heat from a brake, but it helps to know how to handle heat from the brake before applying it!

In the same way that brakes convert velocity to heat, I will outline several techniques for transforming configuration issues into performance issues. I believe that the most powerful tool the system administrator has for dealing with trouble is *architectural design*. The proactive system administrator strives to make trouble easier to handle by employing virtualization to *limit the forms in which trouble arises*. When one changes the form of trouble, one may need new skills to mitigate new kinds of trouble. But if one designs cleverly, the total time one actually spends, the amount of downtime, and the knowledge needed to cope can all be dramatically reduced.

Minimizing Coupling and Maximizing Cohesion

Our first two steps borrow principles from software engineering. A good architectural design minimizes coupling and maximizes cohesion [1]. Two components are coupled to the extent that they interact; couplings correspond to "things to remember." By contrast, a cohesive component groups related functions together inside one entity.

Unnecessary coupling is a major cause of troubleshooting and maintenance cost. Examples of coupling problems include version skew in libraries and/or packages, disagreement between two parameter values that should agree, or conflicting (and thus impossible) requirements for assuring the function of two co-resident applications.

As a trivial example, it is impossible to install both php4 and php5 Apache modules at the same time. Such dependencies arise from application requirements (e.g., one php4 application and one php5 application that are intended to execute on the same physical server).

The main trick I will use to transform trouble is to trade performance problems for combinatorial problems. A "combinatorial problem" is an error in how software is configured or how it interacts with other software, whereas a "performance problem" is a situation in which an operation executes properly but perhaps more slowly than might be desired.

For example, one can solve the php4/php5 problem by creating two virtual operating system instances, each running its own Apache server. One server includes php4 and the other includes php5. The illusion that both are running on the same machine can be maintained by making the original machine into a proxy server.

Segregating services onto distinct components changes the kind of trouble that can arise for the services. If they are running on separate servers (either through physical or virtual separation), then the services are prevented from interacting in ways that co-located services can, so there is absolutely no problem in supporting php4 on one instance and php5 on the other. But we may have to maintain, by other means, the illusion that the applications execute on the same server (e.g., by some form of service switching). One form of complexity replaces another.

It is possible, though, to minimize coupling too much. One should also strive for *cohesion*. Two services are cohesive if they interact with a shared information domain. For example, putting DNS and DHCP on the same server is (usually) cohesive because both pertain to IP, but co-locating DHCP and a Web server is (usually) not cohesive, because the information domains of the two services (usually) overlap very little.

The concepts of coupling and cohesion are borrowed from software engineering but the justifications are perhaps even stronger for system administration. In software engineering, coupling between program modules leads to a need for increased *communication between module authors,* which delays software development. In system administration, coupling between components leads to a need for increased *knowledge* on the part of the individual administrator trying to make them work together, which means more time spent in initial setup and in troubleshooting the interacting components.

Through use of virtualization, *dependency troubleshooting of co-located services is an obsolete skill,* because two software packages that implement services can be positioned within different (virtual) platforms that cannot "depend" on one another. The whole process of installing an instance becomes centered on one application and its needs. But in the latter case, a new form of trouble can arise, in the form of *resource dependencies* (e.g., shortage of CPU cycles or I/O bandwidth among two or more instances). These dependencies cannot break an application, but they can cause it to execute unexpectedly slowly. We do not eliminate trouble; we merely transform its nature.

One side effect of using virtualization is that some of the complexities of configuration management are also obsolete. One thing that makes configuration management difficult is change. In a virtual environment, one can often afford to build a new server instance while existing server instances are live, so that one can start afresh whenever a change is needed. This mitigates several kinds of configuration management problems.

Exploiting Social Pressure

A second design guideline is so obvious that many of us might forget it. Software cannot ever be completely tested. Therefore, it makes sense to design one's systems around software environments that others have aggressively utilized and tested, because each application is more likely to have been thoroughly debugged for those environments than for others. In particular, bugs resulting from configuration problems (e.g., hidden dependencies) are much less likely to arise in commonly deployed environments. The simple reason for this is *social pressure;* the widespread use of a particular environment means that most bugs for that environment will be discovered, reported, and, hopefully, repaired. The most common environments for an application thus naturally become the most tested and functional, because there is a higher incentive for developers to address the bugs with the widest social exposure. Thus, it is typically much more likely that an application will run properly in a vanilla environment (e.g., the default installation of a Linux distribution) than in a customized one. If problems do arise, it is more likely that others have seen them before and have already found and published work-arounds.

By using virtualization one can arrange, much more easily than ever before, for an application's environment to be the one with the greatest social footprint, because the environment for each application can be chosen independently.

Horror stories about failing to exploit social pressure abound. One should not adopt software on the bleeding edge unless one expects to bleed along with it.

For example, our site was one of the first adopters of Sun's NIS+ directory service, because it had many neat features we wanted. Unfortunately, it also had many painful bugs we did not want. What we did not understand or account for at the outset of this project was the power of social pressure. NIS+'s deployment footprint never became large enough for the bugs to be addressed (we heard later that Sun had not used it internally), and we replaced NIS+ with LDAP before the problems we encountered were resolved. By contrast, by possessing an enormous and multi-platform social footprint, LDAP has been pounded upon by a large number of conscientious users and has thus been forged by social pressure into a reliable tool.

This is an object lesson in the danger of creativity. Becoming a follower rather than a leader often involves less pain and suffering. This principle takes many forms, from avoiding first adoption of a tool to avoiding being the first to apply a new security patch [2].

It may seem obvious that our jobs as system administrators do not involve making developers fix their bugs but, instead, require us to provide mechanisms for getting useful work done in the presence of those bugs. Although we file bug reports as a public service, no system administrator can reasonably expect a user to wait for a bug fix. We are instead the masters of the work-around, not the masters of the software, and if anyone has managed to make it work, we are expected to know exactly how and why. Again, virtualization allows us to synthesize almost any software environment needed by an application, without breaking any other one.

Softening Hard Boundaries

A third trick in the contemporary system administrator's arsenal is to use virtualization to control which attributes of a network are "hard" and which are "soft." A *hard attribute* is an attribute of a system that can only be controlled by a human being, such as the physical location of a machine or the location of the access point to which it binds. A *soft attribute* is one that can be manipulated by setting values of parameters via software and/or automation.

The easiest example of hard and soft boundaries involves the computing power of servers. In a non-virtualized environment, the amount of computing power available to a service is a hard attribute, whereas in a virtualized environment it can be considered a soft attribute (e.g., a configuration parameter of the hypervisor). As another example, virtual LANs make the network to which a host is connected a soft attribute, whereas in non-virtual LANs this is a hard attribute.

The overall purpose of softening a hard boundary is to turn a decision whose implementation requires major work into one requiring setting parameters. For example, consider the example above for php4 and php5. If the two applications are installed on two servers, then changing the response time for one application requires rebuilding the service on another server, but if the applications are virtual instances on one server, changing response time can be expressed as a parameter change in the hypervisor.

In both of these cases, softening does not eliminate entropy; rather, it transforms it into a new form. Even the very best virtualization strategies exact a performance penalty, because sharing resources among more than one operating system takes time (thus invoking the second law).

Edge Cases

Alas, there are always cases in which one cannot straightforwardly eliminate troubleshooting of combinatorial problems. Mostly this is because the user explicitly requires several conflicting services to be co-located on the same device, such as a workstation. Then we are faced with the same old combinatorial problems. What to do?

Fortunately, there are several evolving approaches to this problem, all involving advanced forms of virtualization that the system administrator controls. Operating systems do not represent the only grain at which virtualization can function; one can also virtualize file access, registry access, or library access for different applications running within one operating system instance. Some visionaries in the virtualization community believe one will be able to routinely virtualize the software environment for each application without virtualizing the underlying operating system. The net result of this strategy is the same as before but is much lighter in weight; virtualizing the "open" call has a much lower overhead than virtualizing the whole operating system.

For example, IBM's prototype Progressive Deployment System (PDS) [3] virtualizes library and registry access in Windows without virtualizing the whole operating system. Each application thus executes in a custom environment in which registry or library conflicts cannot occur. This is done without virtualizing the whole operating system, which makes it much less resource-intensive to use.

Understanding Resource Contention

In all of the examples cited so far, we have transformed entropy arising from combinatorial conflicts into entropy arising from resource conflicts. In the first case, we traded speed for combinatorial complexity, preferring a simpler, slower solution to a faster, more complex one. In the second case, we traded customizability for robustness, preferring a mainstream, well-understood solution to a perhaps more customized but less-tested option. In the third case, we traded space and time for flexibility, preferring to control state via software rather than by rebuilding servers. The good news is that a few common forms of system failure, including downtime from configuration conflicts, are "virtually" eliminated.

But in system administration, as in thermodynamics, entropy remains. We have only changed the way it can be expressed. We have ensured that the various and sundry state machines making up our applications have the configurations and environmental conditions that they need to react correctly, but not that resources that they need will be available when they need them.

Addressing resource conflicts is a very different form of troubleshooting from those most system administrators are used to. Resource contention is a "quiet" kind of failure; systems fail "not with a bang, but with a whimper." Failures are subtle and sometimes nearly unnoticeable.

But there is also a subtle value shift involved. Virtualization has explicit performance penalties. In eliminating combinatorial issues, we have already departed from the old rubric of making systems function "as fast as possible," and we are forced to ask ourselves some difficult questions about what performance is "good enough." Once we know what is "good enough," we can ask ourselves the second question, "What changes will provide performance that meets that standard?"

What Is Acceptable Performance?

Old habits die hard. Most of us are used to squeezing the maximum possible performance out of our systems, so that the question of what is appropriate performance never arises. When we use virtualization tricks to invoke independence, social pressure, and softness, we trade optimal performance for robustness. Obviously, it is possible to trade away more performance than is reasonable. But what is "too much to trade"?

First, we need some reasonable definition of what performance actually means. There are several possible definitions, all involving some concept of *response time*. For a Web site, response time is the time it takes from when you send a request to when you receive content. For a shell, response time refers to the time between a key press and the associated change in screen state. In a batch environment (e.g., accounts payable), response time is the elapsed time between job submission and job completion.

Second, we need some way of measuring performance. There are many mechanisms, both direct and indirect. A direct performance measure quantifies what the user sees, whereas an indirect measure is related to, but is not exactly equivalent to, the user's experience. Direct measures include benchmarking and soliciting user feedback; indirect measures include server load, memory utilization, etc. The latter are functionally related to what the user sees, but the relationship is not (usually) easy to describe. For example, we agree that servers with high load averages are "bad," but "just how high is bad" depends upon what the user experiences, and not necessarily what the system administrator sees in the logs.

SLOs and SLAs

The next step is to define *acceptable* performance. Here we can borrow some terms from autonomic computing and outsourcing. A "Service Level Objective" (SLO) is a definition of what directly measured performance is "good enough." This is usually specified in very high-level terms, as end-to-end response time (e.g., "Users should obtain a response from the Web site within one second"). SLOs are determined by economic analysis of the business effects of service delays. For example, a few seconds of delay may be catastrophic for online stock trading, and it is generally accepted that response delays in online shopping lead to lost sales.

An SLO may also set different goals for each kind of service or each kind of client. It is common to refer to clients as "gold," "silver," or "bronze" to denote priorities for performance. For example, in a hospital, doctors need "gold" service levels, but for staff not involved in patient care (e.g., billing personnel), "bronze" response suffices. In the emergency room, an even higher "platinum" service level may be needed.

SLOs can also embody business strategy. Some analysts believe that in a sales situation, *it is better not to respond at all than to respond slowly*. Giving up on customers who have already waited too long diverts computational resources away from customers who represent lower sales potential, to customers who have not yet been made to wait (and thus represent higher sales potential). Such a strategy is sometimes called an *admission control policy*, because one only "admits" customers to one's site that one has the resources to serve in a timely manner and tells the customers whom one expects to experience long response delays to come back later (because, statistically, if one does admit them, they are likely to leave before buying anyway) .

By contrast with an SLO, a Service-Level Agreement (SLA) defines not only desirable objectives but also penalties and incentives in interacting with some external client. SLAs are common in defining expectations between a business and a hosting service. Whereas an SLO might say, “Response time should be less than one second,” an SLA might add, “Response times over one second will be billed to the provider at one cent per instance” or “The provider will be paid one cent more for each request whose response time is less than one-half second.” Incentives often vary for different kinds of clients.

The typical use of an SLA is to define interactions between autonomous service providers, but system administrators can utilize the concept as a way of describing service requirements between themselves and their organizations. In this case:

- A service objective is the minimum performance required by management.
- A service penalty (for not meeting the objective) or incentive (for exceeding the objective) can be interpreted in a human context (e.g., raises and promotions).

The Hard Question

In moving into a new territory it helps to understand the objectives and incentives. What is your SLA as a system administrator? If your site is a development site, there may not be a strong business reason for quick response time, so your SLO expectations may be low and your SLA may be undemanding, whereas fluidity and deployment agility may be very important instead. If your site engages in financial trading, there may be sound business reasons for your SLO to include high minimum expectations and high penalties for delays.

This can be a very difficult question to answer, because most managers may not ever have thought about system administration in this way and may not be aware of the thermodynamic principles (as outlined in this article) that give system administrators a *choice* between manageability and performance.

A New World

Virtualization gives us new choices. The profound impact of those choices is to trade one property of a system for another. This allows us to architect systems for robust behavior, effective automation, and autonomic control. But to reap the benefits, we cannot tune a system to run “as fast as possible,” and such an objective is now rather meaningless. The job of system administrator has changed, from doing “whatever it takes to make it work,” to making choices that are “good enough.” Before, we were thinking about cost; now it is time to concentrate on value.

REFERENCES

[1] Roger S. Pressman, “Design Engineering,” in *Software Engineering: A Practitioner’s Approach,* 6th ed. (New York: McGraw-Hill, 2004), chapter 9.

[2] Steve Beattie, Seth Arnold, Crispin Cowan, Perry Wagle, Chris Wright, and Adam Shostack, “Timing the Application of Security Patches for Optimal Uptime,” *Proc. LISA ’02*, USENIX Association, 2002.

[3] Bowen Alpern, Joshua Auerbach, Vasanth Bala, Thomas Frauenhofer, Todd Mummert, and Michael Pigott, “PDS: A Virtual Execution Environment for Software Deployment,” *Proceedings of the 1st International Conference on Virtual Execution Environments (VEE ’05),* ACM Press, 2005.

ROBERT SOLOMON

a 36-user Asterisk installation

Phone System Administrator is one of several hats Bob Solomon wears at a medium-sized non-profit in New York City. He would like to participate in more Asterisk projects.

bobsol@gmail.com

MY NON-PROFIT EMPLOYER NEEDED to replace an aging Toshiba KSU system because the phones, some of which date back to the '80s, were falling apart and the voicemail's hard drive was grinding loudly. I was able to convince management to replace the system with an Asterisk-based PBX by promising a lower end cost and the elimination of vendor lock-in. In this article, I walk you through the process I followed: selecting phones, modifying the Asterisk configuration, and training users.

Asterisk installations of the sort I had done previously, where Asterisk functioned as an answering machine or a small office phone system, have been documented in many places. This installation offered an opportunity to take my Asterisk skills to the next level. The system supports eight lines, 24 extensions, and over 36 voicemail users.

Selecting Phones

Originally, I proposed a system utilizing a channel bank and Aastra 9116 phones. The final design used Polycom IP430 and IP601 SIP phones. The keypad and voice quality of a sample 9116 was disappointing in comparison to our existing Toshiba EKT phones.

Business-quality analog phones are available at about the same price point as the Polycom SIP phones that we ultimately selected. To support the SIP phone the only hardware required is an Ethernet or, ideally, Power Over Ethernet port. The per port cost of POE is much less than that of an FXS (analog phone station) port.

The existing phone system provided voice-first intercom; that is, intercom calls were answered by the recipient's phone on speaker phone, without any action on the part of the recipient. A warning tone preceded the call so that the recipient knew when he or she no longer had privacy. An all-page feature similar to voice-first intercom, but connecting to most phones on the system at once, one way, was also provided. A conversation with management confirmed the importance of retaining these features. I tested a sample Polycom IP430. This phone could handle voice-first calls with better sound quality than the existing system. The keypad feel and sound quality compared favorably with our existing phones.

Polycom phones can be configured from the keypad, through an HTML interface or by XML files, or automatically downloaded by the phone at bootup. I strongly recommend setting up the Polycom phones to download their configuration from the Asterisk server rather than configuring the phones individually. Getting the phones talking with the server's dhcpd, (VS)ftpd, and ntpd was a day's work [1].

Infrastructure and Hardware

The Toshiba system used twisted pair, but SIP phones require CAT5 or better. I considered and rejected the use of our data network for telecommunications.

The separate voice network I constructed incorporated a POE switch, eliminating the need for an AC adapter at each phone. This is convenient for the users and makes the installation of wall phones away from an outlet cleaner and easier. The voice-only network avoided bottlenecks in the existing data network and segregated SIP phone registration and ftp configuration traffic, enhancing security.

I selected a Netgear 24-port POE switch, model FS728TP, based on POE watts available and price. The fan in this switch was noisy enough that someone in the adjoining room complained, so I removed the switch from the rack-mount and placed it on sound insulating material on a rack-mount shelf.

Asterisk is hosted here on a server built on an Asus TS300-E4/PA4 with a Xenon dual-core 3070 2.66-GHz processor with 1 GB RAM. Call volume is about 6,800 calls a month. I can rip a CD on the system for music on hold during times of heavy call traffic without degradation in voice quality.

Connection to eight POTS lines (FXO) and four FXS ports for analog extensions is provided by a set of three Sangoma A200 cards with optional hardware echo cancellation. Echo cancellation is always needed with modern analog (really digital) voice cards. Hardware echo cancellation is of better quality than freely available software offerings, reduces the load on the CPU, and saves several hours of time tuning settings to eliminate echo. Only one echo cancellation module is needed per set of Sangoma cards. There have been no complaints about sound quality on the calls handled using these cards except for rare reports of echo when someone is using a headset. This is resolved by turning down the volume on the extension. Another advantage of the Sangoma cards is that they do not require a PCI slot with a unique IRQ per four lines as did the Digium TDM400P cards used on a previous system. Setting up three TDM400Ps with unique IRQs can take hours on the wrong motherboard.

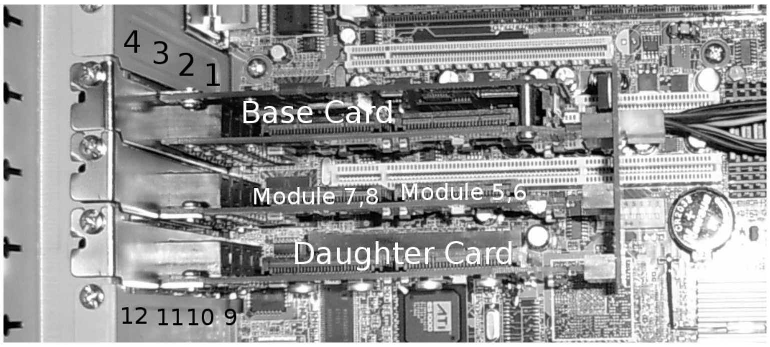
The Digium cards have been redesigned and I have no experience with the new version, TDM410P, which addresses these issues. The Sangoma cards have noisy FXS or station ports. Most of our extensions are SIP phones, so the noise has not been a problem here.

Hardware Issues

About three weeks after the system went live, while I was away for the weekend, the system crashed. When I checked the log files, I found that the system had crashed at a time when no calls were active. An updatedb cron job had been running. Within a week, the system crashed again early in the morning while running a backup. Initially I suspected a failing hard drive. When the system was taken offline, late at night, memtest86 was run on a hunch and memory failure was detected [2]. There have been no crashes since the memory was replaced. Please note that the symptoms of this memory failure were freezes during disk I/O and md5sum sometimes reporting bad check sums on good files.

The server was temporarily moved to a physically smaller SATA-only system that had no Molex power connectors. I spent the night trying to get more than one Sangoma card to work in the replacement server and learned some details about the Sangoma A200 card system that are not obvious from the sales literature, although they are more or less documented on the Sangoma wiki hardware page [3].

Sangoma A200 cards are assembled into a system consisting of a base card, daughter cards, and modules using a small backplane. As many as 24 ports can be installed using one PCI slot. In the case of this system, 12 ports are provided by a base card, two daughter cards, and six modules (see Figure 1). This assembly of cards uses one PCI slot, but it fills three openings in the back of the box. An advantage to Sangoma's system compared to others that use separate PCI slots for each group of modules is that all channels have common synchronous clocking [4].



That night, I tried installing the cards in many different configurations and combinations, but each time I rebooted the server, wancfg_zaptel would detect the cards, but loading the modules with "wanrouter start" would fail when more than one card was installed. This was because the Sangoma backplane requires a 12-V connector and the power it supplies even if no FXS (station) cards are installed.

Ports are numbered according to the backplane slot used, with the lowest ports in the leftmost slot. If a slot is skipped, there will be a gap in the port numbers.

The base card can be plugged into any socket in the Sangoma backplane. Clearance between the cards is acceptable with the base card in any position but is best when the base card is in the leftmost socket on the backplane.

The next day a co-worker suggested that I locate and remove the defective memory stick and switch back to the proper server. Service was restored to all ports while I waited for replacement memory to arrive.

Extensions Using Dialplan Pattern Matching

I used pattern matching in the dialplan for calls to the extensions, rather than a macro. In the global section of the dialplan a variable like the ones shown for extensions 12 and 13 is set for each real extension. To add an extension, all that must be done to the dialplan is to add another variable like those here:

```
x12=Sip/12
x13=Zap/11 ;door phone
```

The following dialplan excerpt handles intercom calls placed from an inside context:

```
; line below is for voicemail for calling self
exten => _[1-8]X,1,GotoIf($[${CALLERID(num)} =
  ${EXTEN}]?CheckVoiceMail)
; check if there is an extension if not go to voicemail
exten => _[1-8]X,n,GotoIf($[!${EXISTS(${x${EXTEN}})}]?Voicemail)
exten => _[1-8]X,n,SIPAddHeader(Alert-Info: Ring Answer) ; voice first
exten => _[1-8]X,n,Dial(${x${EXTEN}},20,tk)
exten => _[1-8]X,n,Voicemail(${EXTEN},u)
exten => _[1-8]X,n,GotoIf($[${VMSTATUS} = FAILED]?NoVoicemail)
exten => _[1-8]X,n,Hangup()
; "this extension has no voicemail or your message was
; too short"
exten => _[1-8]X,n(NoVoicemail),Playback(cust89)
exten => _[1-8]X,n,Hangup()
exten => _[1-8]X,n(CheckVoiceMail),VoiceMailMain(${EXTEN})
exten => _[1-8]X,n,Hangup()
```

If the extension matches the caller ID, we check voicemail. The Polycom phones place a call like this when voicemail is called from the voicemail button on the phone. If an extension variable does not exist, we go straight to voicemail, caller side. If voicemail fails, an error message is played to the caller.

Tuning the System

The Polycom phones were dumbed down and customized. Call forwarding was disabled in the site Polycom configuration file to prevent abuse by our users and guests.

```
<divert>
  <fwd divert.fwd.1.enabled="0" divert.fwd.2.enabled="0"
     divert.fwd.3.enabled="0" divert.fwd.4.enabled="0"
     divert.fwd.5.enabled="0" divert.fwd.6.enabled="0"/>
</divert>
```

Call waiting was turned off for each extension in sip.conf:

```
call-limit=1
```

Calls to the 601s, used by our receptionists, still came through on call waiting. I had to add the following line to the Polycom site configuration file to get these phones to return a busy signal:

```
<call call.callsPerLineKey="1">
```

This was important because I wanted calls that come through while the receptionist is talking to come through on the additional line buttons rather than as call waiting calls on the first button.

A DND (Do Not Disturb) hard key was added to every Polycom 430. I edited the site phone configuration file and physically changed a key cap on every phone:

```
<keys key.IP_430.32.function.prim="DoNotDisturb" />
```

To set up voice first I edited the Polycoms' configuration and the Asterisk dialplan. The site-wide Polycom configuration file was modified, providing a ring class with a ring type of ring-answer [5, 6]:

```
<alertInfo voIpProt.SIP.alertInfo.1.value="Ring Answer"
  voIpProt.SIP.alertInfo.1.class="4"/>
<RING_ANSWER se.rt.4.name="Ring Answer" se.rt.4.type="ring-answer"
      se.rt.4.timeout="1000" se.rt.4.ringer="11" />
```

In the dialplan a SIP header must be sent to the phone on a per-call basis to make that call voice first:

```
exten => #30,1,SIPAddHeader(Alert-Info: Ring Answer)
```

Users initiate an all-page by pressing #30, the code used on the previous system. In the dialplan, this is implemented with the Page command:

```
exten => #30,n,Page(${ALL_PAGE})
```

A context for blind transfers is provided and the __TRANSFER_CONTEXT variable set to avoid the blind transfers going through ring-answer: When testing the system, before this change was made, a transferred call would be automatically answered and on speaker phone whether the intended recipient was at her desk or not.

```
exten => s,n,Set(__TRANSFER_CONTEXT=t_c) ; context for blind transfer
```

I have not yet found an equivalent solution for supervised transfers. These transfers are currently set up to go through voice first. Ideally, the first contact, introducing the transfer, would go through voice first. When the receiving party accepts the call and the transferring party hangs up, the transferee's phone should ring.

The Polycoms and Asterisk do not play well where call parking is concerned. The Polycoms expect the user to provide the parking space number; Asterisk expects to provide the space number. To use the park feature key on the Polycoms, the user has to press "more" then "park," press a bogus extension number that Asterisk will ignore, and then press park again. If the phone has more than one line presence, Asterisk calls the user back with the parking space number, rather than just announcing the number. Users universally elect to park calls PBX style, dialing a two-digit feature code on the keypad rather the using the feature on the phone. In this case, all the user has to do is dial *2 and the system immediately parks the call and announces the parking space number to the user.

The two-digit park sequence is set in features.conf. I needed to adjust the interdigit timeout to 1,000 ms, a value that works for our users:

```
[general]
...
;featuredigittimeout = 1000
[featuremap]
...
parkcall => *2   ; Park call (one step parking)
```

The park key on the Polycom phones requires a callpark extension in the dialplan.

```
exten => callpark,1,ParkAndAnnounce(silence/1:pbx-transfer:
      PARKED|120|SIP/
${DIALEDPEERNUMBER}|internal,${DIALEDPEERNUMBER},1)
```

In either case, parkedcalls must be included in the context:

```
include => parkedcalls
```

Extension Aliasing

When an employee leaves the organization, I am often asked to forward that employee's calls to the extension or mailbox of the person taking over the departed employee's workload.

Initially, a documented solution could not be found [7]. Experimentation revealed that this can be done with a Goto:

```
exten => 69,1,Goto(74,1); 69 is an alias for 74
```

Door Intercom and Door Lock Control

The old system provided an intercom with a door lock control at the side door. To implement this feature with Asterisk, I provided a Viking analog speaker phone, model E-20B, with no dial and an auto answer feature. This phone looks like an intercom. Calls from this phone start in an immediate context in the Asterisk dialplan and ring selected extensions automatically.

```
[door]
; door intercom
exten => s,1,Answer()
exten => s,n,Dial(${DOOR_CALLS},30,tk)
exten => s,n,Playback(silence/1)
exten => s,n,Hangup()
```

The immediate keyword is set in zapata.conf so that the `s` extension executes without any action other than going off hook when the intercom user presses "call":

```
context=door
immediate=yes
callerid="Door Intercom"
group=2
signalling = fxo_ks
channel => 11
```

A Viking door control box, C-2000A, providing a relay for lock control, and a MIS1C DTMF relay board available from Mike Sandman were considered for door lock control. I selected the Sandman board because connection is in parallel with the phone on an available FXS, station, port. The C-2000A requires an FXO (central office) port and costs four times as much as the MIS1C. The C-2000A includes a metal case and offers many features that I did not need here.

To open the door, users press a programmable code while on the door intercom call [8]. I added the following line to the incoming context:

```
exten => 13,1,Goto(13,NoVoicemail)
```

When callers press the door intercom number, 13, from an outside call they hear "This extension has no voicemail." Callers are prevented from calling the door intercom from outside the building.

Managing Day, Night, and Holiday Mode Changes

Asterisk configuration is finer-grained then the proprietary systems I have worked with in the past. The area of day and night call handling provides a good example of this.

The system plays a different main message depending on whether we are open, closed for the night, closed for a holiday, or closed on an August Sunday. Also, calls to the operator go directly to voicemail when we are closed for any reason. Management stipulated that these changes in day and night call handling be automated.

Once per incoming call a global variable MODE is read from the persistent Asterisk database:

```
exten => s,n,Set(GLOBAL(MODE)=${DB(vars/MODE)})
```

The value of this variable has been set to 0 through 3, with 0 representing open and the positive integers representing various closed states. A main message is chosen based on the value of the variable:

```
; set and play the main message day|night|holiday|sunday-in-August
exten => s,n,Set(MSG2PLAY=cust02)
exten => s,n,ExecIf($[${MODE} = 1]|Set|MSG2PLAY=cust03)
exten => s,n,ExecIf($[${MODE} = 2]|Set|MSG2PLAY=cust07)
exten => s,n,ExecIf($[${MODE} = 3]|Set|MSG2PLAY=cust08)
exten => s,n,Background(${MSG2PLAY})
```

If MODE is true when a caller in the incoming context presses 0 a night message is played and the Operator extension does not ring:

```
exten => s,n,GotoIf(${MODE}?oper-night,s,1) ;play night if mode > 0
```

A cron job, changing the value of MODE in the Asterisk DB, is run at closing time every day.

```
# night mode, weekday, saturday
30 20 * * mon-fri,sat /usr/sbin/asterisk -rx \
  'database put vars MODE 1' &>/dev/null
```

At opening time in the morning, cron runs a Perl script that checks for a holiday and changes the value of MODE accordingly [9]. Luckily, we are open for business on Easter.

Advantages to this approach are persistence of the MODE variable over a restart of Asterisk or even a reboot and control of our schedule with cron and an easily maintainable Perl script. Holiday determination is made once per day rather than once per call.

In the event of a mishap, the mode can be changed from any phone, as follows:

```
[set-mode]
; "system is in day|night|holiday mode"
exten => s,1,Background(cust9${DB(vars/MODE)})
; "press 0 for day, 1 for night, 2 for h..."
exten => s,n,Background(cust88)
exten => s,n,Waitexten(5)
exten => s,n,Hangup()

exten => _[0-3],1,Set(DB(vars/MODE)=${EXTEN})
; "system is in day|night|holiday mode"
exten => _[0-3],n,Background(cust9${EXTEN})
exten => _[0-3],n,Hangup()

exten => i,1,Background(pbx-invalid)
exten => i,n,Goto(s,1)
```

Training Users

The previous KSU phone system provided presence for six of our outside lines at each phone. When the organization was smaller, with only three incoming lines, a KSU setup was advantageous because call handling was intuitive. We have had eight lines for many years now and Asterisk provides for better call handling than could be provided with line buttons on each phone. Because PBX call handling requires skills that will be new to most users, training is required for receptionists and others in the front line of call handling.

I reached an understanding with management about the need for user training prior to implementation. Management's cooperation in this area was key to the success of the project.

Training was provided for about half of our employees one on one and in small groups. I trained some staff myself and also trained others to train. A dialplan extension was provided which moves a call from the inside to the outside context for the purposes of training and testing. Trainings took place at a desk with two extensions and a cell phone.

Some useful training material was available on the Web, particularly at the site of the Woods Hole Oceanographic Institution (WHOI). Unfortunately, these pages have been moved or removed. I customized the material on the WHOI site and wrote additional material [10].

We use four techniques for handling calls here: supervised and blind transfers, call parking, and exclusive hold.

Blind and supervised transfers were new to our users. In a supervised transfer, the transferee speaks to the recipient before the call is connected. For nonemergency, nonexecutive calls, a blind transfer to the recipient's extension is the best way to handle a call.

On the old system, a call could be placed on hold and a wanderer paged: "So and So, please pick up a call on line three." Users are intimidated by call parking, even though the process is very similar. When a call comes in for a wanderer, the call is parked and the recipient paged: "So and So, please pick up a call on 701."

I got repeated questions about how to forward a voicemail. When I tried this myself, I found that upon completion of a successful voicemail forward the system says, "Your message has been saved." The user is left to wonder if the transfer has gone through. This prompt can't be changed because the system uses it for other purposes.

Is an Asterisk System Right for Your Site?

A bare-metal Asterisk installation offers savings in hardware costs over a proprietary system but adds costs for the time spent in configuration. Packaged four-line, eight-extension turnkey small business phone systems (without phones) are available for under $2,000 [11, 12]. Some of these systems are Asterisk-based. The server, switch, voice cards, and 24 phones used here cost $7,200. Components related to cabling and infrastructure cost $2,100. Twelve weeks of labor went into this project, including planning and research, requisition, premises wiring, system installation, configuration, and initial training of staff.

Asterisk must be able to provide the features expected by management and users with a reasonable amount of work. Voice-first intercom was a must-have feature here. Two-digit extension numbers and some feature codes that have been in use for over two decades were preserved. A door intercom was provided. Phones needed to be simple and intuitive to use.

You should have a good idea of what a properly working phone system sounds and feels like. Experience administering the existing or another similar system is a big plus. Compassion for the user experience is important. In a small to medium-sized business, cabling skills may be handy.

The sustainability of a phone system that is managed by a system administrator editing files in an organization this size concerns me. I have not investigated available Asterisk GUI front ends, under the assumption that these front ends would not be capable of the degree of customization needed here. Secondary personnel should be trained in the administration of the system so that changes and failures can be addressed when the primary administrator is not available.

Conclusion

Despite concerns about sustainability and a few too many rough edges, this Asterisk installation has been successful. Users and management are happy with the system. Sitting at the console and watching three dozen people interact daily with a system running on Patrick Volkerding's Slackware is satisfying.

Resources

The Asterisk beginner would do well to start with the book *Asterisk, the Future of Telephony* [13]. At the Asterisk console, man style information on any dialplan application or function can be accessed by typing:

```
fone*CLI> core show application ApPlIcAtIoN ;not case sensitive
```

or:

```
fone*CLI> core show function FUNCTION ;must be uppercase
```

When I need more detail than either of these resources provides, the voip-info.org Asterisk pages [14] are often helpful. Start with the voice board vendor's Web site for installation instructions for both the kernel modules supporting their boards and Asterisk itself. In the case of the Sangoma boards used in this project, the Sangoma wiki [15] was very helpful. The Asterisk Web site [16] provides a good overview of the Asterisk project. If Polycom SIP phones are used don't fail to download the *Administrator's Guide for the SoundPoint IP/SoundStation IP Family* [17].

REFERENCES

[1] http://www.voip-info.org/wiki/view/Asterisk%40Home+Handbook+Wiki+Chapter+7#7221WhychoosePolycomVOIPPhonesbspan.

[2] http://weaversrevenge.com/ast36/burned-out.jpg.

[3] http://wiki.sangoma.com/sangoma-hardware#A200.

[4] http://www.sangoma.com/products_and_solutions/hardware/analog_telephony/.

[5] http://www.voip-info.org/wiki/index.php?page=Asterisk+cmd+page.

[6] See p. 151 of the *Administrator's Guide for the SoundPoint IP/SoundStation IP Family:* http://www.polycom.com/common/documents/support/setup_maintenance/products/voice/SoundPointIP_SoundStationIP_AdminGuide_SIP3_0_Eng_Rev_A.pdf.

[7] http://asterisk-jtapi.sourceforge.net/setup.html.

[8] http://weaversrevenge.com/ast36/door-buzzer.

[9] http://weaversrevenge.com/ast36/holidays.

[10] http://weaversrevenge.com/ast36/voicemail-crib.odt.

[11] http://shop.talkswitch.com/details/CTTS001184001.asp?.

[12] http://store.digium.com/products.php?category_id=41.

[13] J. Van Meggelen, L. Madsen, and J. Smith, in *Asterisk: The Future of Telephony,* Second Edition (Sebastopol, CA: O'Reilly Media, 2007): http://www.asteriskdocs.org/.

[14] http://www.voip-info.org/wiki/index.php?page=asterisk.

[15] http://wiki.sangoma.com/.

[16] http://www.asterisk.org/.

[17] http://www.polycom.com/common/documents/support/setup_maintenance/products/voice/SoundPointIP_SoundStationIP_AdminGuide_SIP3_0_Eng_Rev_A.pdf.

BRAD KNOWLES

building scalable NTP server infrastructures

Brad has been a contributor to the NTP Public Services Project for over five years, in addition to working as a UNIX and Internet system administrator for almost two decades, specializing in Internet email and DNS administration for more than fifteen years.

knowles@ntp.org

IN THIS ARTICLE I DISCUSS HOW TO take the concept of a simple NTP service configuration for a small number of clients and then expand that to be able to serve many thousands, tens of thousands, hundreds of thousands, or even millions of clients. By choosing the right type of scalable service infrastructure, you should be able to handle a virtually unlimited number of clients with a relatively small number of servers, but it will take some work to get there.

I start out with some discussion about various typical traps that you might tend to fall into, if you're not deeply familiar with the NTP protocol and the Reference Implementation. I follow that by discussing the three different major modes of operation you can choose for your servers, highlighting weaknesses in each mode and other factors that need to be considered. Next, I mention some specific hardware recommendations for good-quality NTP servers, as well as some warnings about OS issues, and touch on the subject of "reference clocks." Finally, I come to some conclusions about which modes are the most scalable and under what circumstances. If you get lost, you may want to consult the list of NTP-related definitions [1].

I assume that you are already familiar with NTP in general and building simple NTP server configurations and that you know where the official NTP documentation is located [2], as well as the unofficial Community Supported Documentation as made available by the NTP Public Services Project [3]. You know what the difference is between the NTPv4 [4] Reference Implementation [5] and the older NTPv3 [6] clients that may be available from other sources (e.g., xntpd). You are also assumed to understand the difference between NTP and SNTP—the Simple Network Time Protocol [7].

Of course, before a machine can be an NTP server, it must first be an NTP client. In NTP parlance, the only difference between a server and a client is that an NTP server is a machine that has NTP clients that are depending on it for time, whereas an NTP client depends on NTP servers but does not have any other NTP clients depending on it. Any NTP client is a potential NTP server, and all NTP servers are also NTP clients. Therefore, I assume that you already know how to build a robust NTP client configuration, including things like knowing how many upstream NTP servers to configure and why defining two upstream servers for your client is the worst possible configuration, proper use of the “iburst” keyword, proper use of the “tos minclock” and “tos minsane” parameters, etc.

Common Misconceptions

For people who don’t understand how NTP works, it is very easy to assume that the way you build a scalable NTP service infrastructure is exactly the same way you would build a scalable infrastructure for any other kind of typical Internet application such as Apache or Sendmail.

In other words, throw the biggest, baddest, honkingest, multi-core, multi-thread, multi-CPU boxes at the problem, and then front-end them with a whole array of proxy servers, load-balancing switches, and clustering, playing tricks with the network layer to make the same service IP address visible from a variety of servers, and so on.

For NTP, this is the worst possible thing you could do. NTP is UDP, not TCP. It does not have a fork()/exec() model of execution. It is single-threaded and essentially does a huge select() loop on incoming UDP packets. Doing TCP or a fork()/exec() model of execution would add unnecessary and unpredictable latency to the process of trying to handle the packets, and it would defeat the entire purpose of accurate time-serving.

However, NTP is not stateless. In fact, it is about as stateful as you can get with a UDP protocol, since it tracks both short- and long-term variations in clock stability for all configured upstream servers, based on the smallest possible statistical samples of information for each system. None of what you would think of as the “standard scaling rules” can be applied to NTP.

The goal of NTP is to try to synchronize your system clock to the “One True Time” known as UTC (Universal Coordinated Time [8]), or at least always move your system clock closer to UTC, within certain statistical error boundaries. Since there is one and only one system clock per computer, you do not want to run more than one NTP daemon on a machine, because they will both be trying to modify the system clock at the same time and they will certainly cause the system to be highly unstable, if not frequently suffering from kernel panics. The Reference Implementation includes code that works hard to prevent more than one copy of ntpd running at the same time.

The algorithms inside NTP are extremely sensitive to the most minor changes, and over the 20+ years they’ve been in development, they have been tuned to seek out and eliminate the tiniest statistical errors they can find, whether the variation is short- or long-term. They also need to do loop detection and elimination, and for that they depend on a one-to-one correspondence between the system clock and the IP address. For multi-homed machines, this can pose a problem, since they don’t have just one IP address.

All of the NTP algorithms are also built on top of the basic assumption that if you contact a client or server at a given IP address, it will always be exactly the same machine with exactly the same system clock. So, for example, playing “anycast” games at the routing layer and making the same IP address available from multiple servers is a recipe for disaster. The same holds for using load-balancing switches or clustering.

NTP already has extensive capabilities for doing explicit failover between multiple upstream servers, and anything you do to try to hide the upstream servers behind something else will only get you worse reliability and worse quality of service.

Configuration Modes Between Servers and Clients

There are three basic modes of NTP server configuration that we are concerned with, and a variation on one of those, for a total of four modes of operation.

The simplest mode, "unicast" [9], is the classic client/server configuration for NTP. That is, each NTP client periodically sends UDP packets on port 123 to the servers they are configured to use, gets UDP responses back, goes through the NTP algorithms to select which of the designated servers is "best," and then tries to synchronize its clock to it. Note that there is no client authentication or authorization in the NTP protocol, but you do have the option to enable cryptographic server authentication to the client.

Unicast mode does have the advantage that it gives the client(s) the best possible chance for getting good time service from the upstream servers, if the upstream servers are not overloaded. This is because a unicast-mode client is involved in a periodic but ongoing long-term bidirectional conversation with each of the upstream servers, and it is able to gather the maximum amount of information possible regarding which time server currently has the "best" time, etc.

Unfortunately, even with good hardware and good configurations on both sides, the simple version of this type of configuration (without cryptographic server authentication) may tend to start having problems when handling more than about 500–1,000 clients per NTP server. Of course, if you have configured your clients to each use multiple upstream servers, then you magnify the problem of how many clients hit how many upstream servers. Overloaded servers start dropping too many queries, too many retransmissions are required, and the servers are providing reduced quality of service to all clients.

Unicast was the first mode invented oh-so-many years ago, and it should be supported by all NTP clients. In the official documentation on this mode, the terminology may differ somewhat from what I have used here but the concepts are the same. Note that you can build purely hierarchical relationships among the NTP servers themselves, or you can build them as symmetric active/passive peers, but either of these server-to-server infrastructures still operates in unicast mode [10].

Next, we have "broadcast" mode [11, 12]. The basic concept is that each client is configured to passively listen for broadcasts from the designated server(s), go through the NTP algorithms to try to select the best-quality time server, and then synchronize the clock to that.

However, in this mode the clients will actually operate in unicast mode during startup (something called the "startup dance"), and then settle down to passive listening. The startup dance allows the NTP client to determine what the "broadcast latency" is between the server and the client and to make suitable adjustments so that it can make a better determination as to which of the designated upstream servers is best, which results in the NTP client getting better-quality time service.

If there is no response from the broadcast server to the unicast packets from the client during the startup dance, the client will fill in a default value for the broadcast latency. Alternatively, the server administrator can hard-code its own choice for broadcast latency. Clients can also be configured to avoid the startup dance altogether, through the "authdelay" command—in effect, making them broadcast-only clients.

Unfortunately, broadcasts don't cross MAC-layer segments, which means you end up needing at least one broadcast NTP server on every subnet. This naturally leads to the concept of running a broadcast NTP server on your network gear, which is generally a bad idea and discussed below.

Broadcast mode should be supported even by older NTPv3 clients.

Up next, we have the multicast variant of broadcast mode, the only difference being that the UDP packets sent by the servers are addressed to a specific multicast address (defined by IANA to be 224.0.1.1) to which all multicast clients listen. However, other than the address being different, the operations are otherwise the same as broadcast mode.

Also note that this requires support at the network level. By default, most network devices are not set up to support routing multicast traffic, so they would need to have their configurations updated. Moreover, each multicast server will see packets from all multicast clients during their respective startup dances, and all clients will see packets from all servers, and again this will tend to bring problems with congestion, drops, retransmits, etc. As with broadcast mode, multicast mode should be supported even by older NTPv3 clients.

With NTPv4, there is a new mode based on multicast networking called "manycast" [13]. This is an automatic discovery mechanism used by clients to find their closest server(s). The client sends UDP packets to the configured multicast address, but it starts with a packet TTL of zero, to see whether there are any manycast servers on the local segment. If the client doesn't get a response in a given period of time, it retransmits with a TTL of one, to see whether there are any manycast servers that are just one hop away. This process continues until either the TTL is set to the maximum and no servers ever respond or the client finally discovers and sets up one or more relationships with servers somewhere on the network.

Manycast mode allows clients to automatically detect their nearest NTP servers and then set up unicast associations with them. The load will automatically be distributed throughout the infrastructure as you put multicast servers in strategic places. You will minimize as much as possible the number of servers that see traffic from too many clients, the number of clients that will see traffic from too many servers, and the number of router hops traffic has to cross in order to get from the starting point to the destination.

Of course, manycast has the same problem as multicast, in that it needs support at the network layer. However, manycast is new with NTPv4, and support for it will most likely not be found in older NTPv3 clients.

I won't discuss "pool" mode, since it is intended to allow sites to make better use of the NTP Pool Project [14] as opposed to helping you set up your own infrastructure (pool-style or not), and it's a new enough addition to the system that I'm not sure it will be covered by the upcoming NTPv4 RFCs that the IETF NTP Working Group [4] is putting together.

But assuming your version of the ntpd code is new enough to include this option, there is official documentation on how you could potentially configure your NTP clients to use the NTP pool [15].

For example, this might be a useful configuration option to use on your NTP servers to help ensure that they get an adequate number of upstream servers and a better chance at good-quality time service, even if you don't configure any of your own internal NTP clients to make use of the pool.

Running NTP Servers on Network Devices

The concept of running in broadcast mode naturally leads to the idea of people running NTP servers right on the networking gear itself. Unfortunately, although most networking gear has specialized hardware for performing the important switching and routing functions, noncore functions (such as NTP) end up getting shunted over to a "general-purpose" processor. Of course, there's lots of things that this general-purpose processor is supposed to be doing in addition to NTP, so you get a competition: the other stuff suffers, or NTP suffers, or— the most likely outcome—both suffer.

In addition, most network devices have the cheapest possible clock circuits—even on the really expensive routers where a single line card can cost a hundred thousand dollars or more. The design of NTP is such that it can only compensate for a certain amount of error in the underlying clock circuits before it just gives up. Most network devices tend to have clock circuits that have so much error inherent in them that they usually run right on the ragged edge of the amount of error that can be compensated for within the NTP protocol. As such, you should always configure them to be NTP clients: they tend to make pretty poor NTP servers even if all the clients are configured to only passively listen to broadcasts.

Moreover, network devices used as broadcast-mode NTP servers probably won't support the cryptographic server authentication methods, which would make them triply poor choices for NTP servers. This issue is also discussed in the Community Supported Documentation [16].

Lost Clock Interrupts

With NTP, you don't ever want to see clock interrupts get lost. This is one of the fastest ways to kill your NTP accuracy, and it will very likely cause the NTP daemon to quit completely. There can be many causes of lost clock interrupts, the two most common being hardware problems [17] and OS problems [18].

When the server is running on complex hardware configurations, you are likely to see excessive amounts of jitter and other statistical errors in terms of servicing clock interrupts. For NTP, the more precise and accurate the servicing of clock interrupts, the better. Typically, this means more simply configured machines are better—you'll never have a throughput issue, so you don't need to throw in really fast network cards with things such as TCP Offload Engines, and since the application is single-threaded you'll never have a CPU load problem that can be resolved by throwing more CPU cores at it.

In other words, you can probably throw out every single modern machine you've got.

Indeed, currently one of the best NTP server hardware platforms you can buy is the Soekris net4501 Single-Board Computer [19]. Poul-Henning Kamp [20] has done wonders with these boxes, and the official NTP time servers for Denmark are running on them. These machines are about as dead simple as you can get, and they can handle hundreds or thousands of clients as easily as or better than pretty much anything else on the planet [21].

There is a comparable SBC configuration that has become more common in the pool.ntp.org project. If you go to the Web site [14], you can find out more about this project, and maybe you can get the current coordinator of the project, Bjorn Hansen, to send you a link to their alternative configuration.

Note that pool.ntp.org operates in unicast mode, and by playing games with DNS-based load balancing and geographically aware names that can be chosen by the admin of the NTP client (which might also be a local NTP server), it ends up scaling to handle several million clients across the world. If the pool were able to operate in manycast mode, I have to believe that this could reduce server hardware requirements by at least one or two orders of magnitude.

Then there are the OS configuration issues. In addition to making sure that the hardware services clock interrupts in a precise and accurate manner, you need to make sure that the OS is configured to do the same.

Unfortunately, owing to the internal architecture of Windows-based OSes, the best they can do is ~50 ms accuracy, and for a good-quality NTP server you really want to get down into the single-digit millisecond ranges, if not lower.

Likewise, you will also see problems on modern versions of Linux or other freely available OSes that try to handle 1000 clock interrupts per second on lower-end hardware (e.g., with kernels configured with "Hz=1000"). On higher-end hardware that can handle these settings, or where you can tune the kernel settings to a more appropriate level, you should be able to get good-quality time service from just about any UNIX or UNIX-like OS available currently or in the past 20 years. My laptop regularly stays in the single-digit millisecond range of accuracy relative to UTC, and it's not anything particularly special in this regard.

Reference Clocks

If you're running a network of NTP servers, you may want to have one or more of your own internal "reference clocks" [22] configured so that you can provide the best quality of time to your clients. This would also give you a good measure of additional robustness in case your Internet connection goes down.

NTP actually depends on these external clocks to provide a reference of UTC against which everything else can be measured. One key measurement in NTP is your logical distance from your closest refclock: a machine directly connected to a refclock is operating at Stratum 1 (the refclock itself is Stratum 0), a machine is Stratum 2 if it is a client of a Stratum 1 server, etc. The lower your stratum number, the better the quality of time service you can potentially provide to your clients, if your refclock is good enough.

There are many different types of reference clocks, including GPS-based devices, radio-based equipment (using WWVB, DCF, or one of the other radio broadcasting stations around the world), rubidium or even cesium-based atomic clocks, and CDMA or GSM mobile telephone–based equipment. Heck, you can even use a dial-up modem to connect to a time service via POTS telephone lines.

I won't discuss any of them in detail, but as an NTP server administrator the primary thing you need to know is that they are configured in a way that is very similar to unicast mode (using the "server" keyword), but instead of listing a hostname or regular IP address, you use the appropriate pseudo-IP address in the 127.127.0.0 range. Which specific address you use will vary depending on which particular refclock you have and which driver it requires.

I will say that GPS-based refclocks are very popular, and if you're willing to build them yourself or if you can find someone willing to build one for you, they can be had for relatively small amounts of money—on the order of $100 or so. If that's too expensive, then radio refclocks can be had for as little as $20, if you're willing to put in some work or you can find someone who will do that for you.

Unfortunately, most vendors ship with their standard NTP client/server software without support compiled in for refclocks. Therefore, if you want to directly connect one or more of your NTP servers to a refclock, you will probably have to recompile and reinstall the NTP daemon, ntpd, from the source code. An alternative would be to buy an appliance that provides both refclock and NTP Stratum 1 service in a turnkey device, although that can get expensive. You'll need to decide which option is right for you.

Conclusions

So, this is what we have for the modes of operation we're concerned about:

- Unicast
- Broadcast
    - Multicast
    - Manycast

Generally speaking, multicast and manycast are the most scalable, with different issues that your infrastructure will have to support or deal with.

Multicast mode will allow a smaller number of servers to support a larger number of clients, but there may tend to be network "storms" of traffic resulting from excessive numbers of clients going through the startup dance around the same time. (Some randomization is built into the startup process, but it can only do so much to alleviate load on the server.) You will also get into issues with too many servers trying to talk on the same multicast channel at the same time.

Multicast mode also tends to encourage centralizing resources for ease of management, which will increase the number of router hops that traffic has to pass through in order to get from the server(s) to the clients and thus will reduce the quality of the time service provided to the clients. Even if multicast servers are located in close proximity to their multicast clients, the clients will not be able to get the best-quality time service, because of the asymmetric nature of the communications between them and the server, and once the startup dance is done, they may be unable to adapt to changing network conditions.

Manycast is still pretty new. The NTP community doesn't have that much experience with it yet, but it may require more server(s) to support the same number of clients. However, since it distributes these servers closer to the clients and minimizes the number of router hops, it helps to increase the quality of time service provided and the probability that clients will continue to obtain tolerable time service in the event that their subnet is temporarily disconnected from the rest of the world.

Overall, manycast mode is considered to be the most scalable and robust NTP server infrastructure. Quoting from the official documentation on manycast:

> It is possible and frequently useful to configure a host as both manycast client and manycast server. A number of hosts configured this way and sharing a common multicast group address will automatically organize themselves in an optimum configuration based on stratum and synchronization distance.

If you can't run multicast or manycast, then your next most scalable option would be broadcast.

In any event, these more scalable techniques tend to increase the exposure your time servers and their IP addresses get to the network and therefore to increase the number of clients dependent on them. That increases the probability that someone else will want to spoof packets from them, which they will probably be able to do quite easily since all communications are via UDP or similar methods.

Therefore, regardless of whether you use broadcast, multicast, or manycast, you should configure your systems to provide cryptographic server authentication to their clients, and ideally you should do the same for unicast mode as well.

RESOURCES

[1] http://support.ntp.org/bin/view/Support/NTPRelatedDefinitions.

[2] http://www.eecis.udel.edu/~mills/ntp/html/index.html.

[3] http://support.ntp.org/.

[4] http://www.ietf.org/html.charters/ntp-charter.html.

[5] http://support.ntp.org/download.

[6] http://www.ietf.org/rfc/rfc1305.txt?number=1305.

[7] http://www.ietf.org/rfc/rfc2030.txt?number=2030.

[8] http://www.eecis.udel.edu/~mills/y2k.html#utc.

[9] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#client.

[10] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#symact.

[11] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#broad.

[12] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#bcst.

[13] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#mcst.

[14] http://www.pool.ntp.org/.

[15] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#poolt.

[16] http://support.ntp.org/bin/view/Support/
DesigningYourNTPNetwork#Section_5.6.

[17] http://support.ntp.org/bin/view/Support/KnownHardwareIssues.

[18] http://support.ntp.org/bin/view/Support/KnownOsIssues.

[19] http://www.soekris.com/net4501.htm.

[20] http://people.freebsd.org/~phk/.

[21] http://phk.freebsd.dk/soekris/pps/.

[22] http://www.eecis.udel.edu/~mills/ntp/html/refclock.html.

SANDEEP SAHORE

reclaim disk space by shrinking files

Sandeep Sahore holds a Master's degree in computer science from the University of Toledo and has nearly 15 years of experience in the computing industry. He specializes in low-level and C programming, systems engineering, and system performance.

ssahore@yahoo.com

The source for cfsize.c may be found at http://www.usenix.org/publications/login/2008-10/cfsize.c.

SYSADMINS FROM TIME TO TIME ARE faced with the problem of reclaiming disk space, a problem that lurks in the shadows waiting to buzz the pager into life. The typical response is either to remove files or to compress them, or to invoke some combination of the two approaches. But there are many situations where the choice is not so cut-and-dried. Let's say there is a file filling up a storage partition that cannot be removed because its data should be kept for a rolling window of one year or because its contents are sensitive or because it is being held open by a process. In such cases it is better to shrink the file in size instead of removing or compressing it.

Having faced such scenarios a countless number of times I decided to write a program that would shrink files in size from the beginning or "head." This program is called cfsize, which is short for "change file size"; it is described in detail in the sections that follow. cfsize is written in the C language for the UNIX platform, but it works equally well on Linux.

The Need for Cfsize

So what is the need for designing and developing cfsize, when a standardized utility, csplit, already exists? Though cfsize and csplit look similar they are poles apart functionally.

Cfsize was ostensibly designed to change a file in size by deleting part of its contents from the "head." By shrinking the input file in place, it reclaims space from a file system that is at its threshold limit. Behind the scenes, cfsize siphons off the data that needs to be kept into a temporary file and when finished it replaces the input file with the temporary one. This retains the latest data in the file while the oldest data is thrown away.

In contrast, csplit makes a copy of the input file and splits the copy into smaller parts based on user-specified criteria. The smaller files thus created can be concatenated to reproduce the original file, since csplit follows "the whole is the sum of the parts" method. It does not alter or shrink the original file nor does it reclaim space. In fact, by creating a copy of the original file, it uses more space.

From the preceding discussion it should be clear that cfsize and csplit are two very different tools. Cfsize reclaims disk space by shrinking files in size, whereas csplit is mostly an intermediate step of a multi-step process.

Among its lesser-known rivals is the trunc.pl utility written in Perl. A keyword search for trunc.pl on the Internet pulls up its Web site. It is similar in functionality to cfsize but uses external UNIX utilities such as tail and system instead of Perl built-ins, thereby incurring overhead owing to the repeated invocation of the `fork()` and `exec()` system calls, not to mention from their small internal buffers. It also takes as its argument a discrete number of lines instead of the new file size, which makes the calculations for reclaiming disk space cumbersome. The abstraction provided by trunc.pl is offset by the performance penalty incurred for shrinking files, especially large ones.

Compilation and Execution

After obtaining the source code, assemble the cfsize executable using an ANSI C compiler (either cc or gcc) as:

```
# cc cfsize.c -o cfsize
```

Store the executable in /usr/local/bin or a directory of your choice and invoke it without any arguments to obtain usage:

```
# cfsize
usage: cfsize -s filesize[k|m|g] file ...
```

Cfsize takes the following options, along with a list of files that need to be reduced in size:

```
-s filesize[k|m|g]
```

That is, the new size of the file is in bytes, kilobytes, megabytes, or gigabytes, with bytes being the default, as specified, respectively, with no suffix or with the k, m, or g suffix as shown.

Program Flow and Design

Conceptually the whole cfsize program can be divided neatly into three distinct parts:

- Parse and process the options given on the command line.
- Open and read the file(s) supplied on the command line.
- Shrink the listed file(s) to the specified size.

Before diving into an in-depth explanation of its parts, let’s go over the mode of operation supported by cfsize. As already stated, cfsize chops off the “head” of the file, implying that the latest entries in the file are kept while the oldest ones are thrown away. Another way to look at this is to think of the file being *rolled* from the top down. When the desired size is reached, the *rolled-up* portion is virtually torn off.

PARSE THE COMMAND-LINE OPTIONS

The cfsize utility takes a single mandatory command-line option `-s`, which takes the new size of the file as its argument. The input file is “chopped off” from the “head” and the program checks whether the new size of the file has been passed to `-s` followed by enabling a flag and invoking the `getfsz()` routine to calculate the desired size of the file. The flag is checked to see whether the mandatory -s switch has been provided on the command line. If any of these checks evaluates to false the program errors out:

```
long getfsz(int s, char *sarg)
{
  int c;
   long n = 0;

  while (c = tolower(*sarg)) {
   switch (c)
   {
     case '0': case '1': case '2': case '3': case '4':
     case '5': case '6': case '7': case '8': case '9':
       n = 10 * n + (c - '0');
       break;
     case 'k': case 'm': case 'g':
       if (*(sarg-1) >= '0' && *(sarg-1) <= '9' && !*(sarg+1)) {
        if (c == 'k')
          kb++;
        else if (c == 'm')
          mb++;
        else if (c == 'g')
          gb++;
       }
       else
        fprintf(stderr, "%s: invalid argument to option --
           %c\n", prog, s), usage(prog);
       break;
     default:
       fprintf(stderr, "%s: invalid argument to option -- %c\n",
                 prog, s), usage(prog);

   }
  ++sarg;
  }
  return n;
}
```

FIGURE 1

The getfsz() function listed in Figure 1 ensures that the argument to the -s option is a valid number. It scans the filesize argument string one character at a time, converting it into an integer while checking for the presence of characters that are not numerical. It also figures out whether the file size reduction is specified in kilobytes, megabytes, or gigabytes. It terminates abnormally if the argument string contains any characters outside the acceptable range.

OPEN LISTED FILES AND SHRINK TO SPECIFIED SIZE

After processing the options, cfsize moves on to reading the files listed on the command line. A while loop is used to open, read, and "chop off" the files from the "head." It has a built-in safety net to terminate execution of cfsize if a user mistakenly enters a file size that is greater than the current size. The size of the file being processed currently is obtained by calling the stat() library function with the filename as its argument. If the new file size is more than the current file size, cfsize terminates abnormally. This safety net prevents the file from being "inflated" instead of being "shrunk." An error is raised if a file cannot be opened, and the program moves on to the next file in the list until the list of files is exhausted:

```
void fsplit(long fsz, char *fnam, FILE *fin)
{
   int c;
   FILE *fout;
   long neg = -fsz;

   /* end abnormally if the temp file cannot be created */
   if (!(fout = tmpfile()))
      catcherr("tmpfile()");

   /* set file pointer to "neg" bytes from end of current file */
   if (fseek(fin, neg, SEEK_END))
      catcherr("fseek()");

   /* go to end of the line to avoid inline file breakage */
   while ((c = getc(fin)) != '\n')
      ;

   /* move the data that needs to be retained to the temp file */
   while ((c = getc(fin)) != EOF)
      putc(c, fout);

   /* truncate and prepare the current file for writing */
   if (!(fin = fopen(fnam, "w")))
      catcherr("Cannot open %s", fnam);

   /* set file pointer to the beginning of the temp file */
   if (fseek(fout, 0L, SEEK_SET))
      catcherr("fseek()");

   /* move the contents of the temp file to the current file */
   while ((c = getc(fout)) != EOF)
      putc(c, fin);

   /* flush buffered writes to the current file by closing it */
   if (fclose(fin))
      catcherr("Cannot close %s", fnam);
}
```

FIGURE 2

Figure 2 shows `fsplit()`, the function that is at the heart of cfsize and which is responsible for shrinking files. It starts by creating a temporary file, using the `tmpfile()` function, for storing the data that needs to be retained. Next it moves the file offset backward from the end of the file, stopping after exactly `filesize` bytes. The file offset is then advanced to the end of the line it currently rests in order to prevent in-line file breakage. This implies that the new size may be the same or less than the file size specified on the command line. With the file offset poised at the beginning of the line, the data to be retained is moved to a temporary file. After the data migration is complete, the temporary file replaces the input file. If any one of the system calls or library functions invoked by `fsplit()` fails, the program ends abnormally.

Examples of Usage

A good way to get familiar with any tool quickly is to understand how it is used in common scenarios, and that's the focus of this section. For example, the command to "chop off" a file ~25 MB in size from the "head" down to 10 kB would be:

```
# cfsize -s 10240 file.txt
```

Alternatively, one can use the `k` (kilobytes) suffix for the file size instead of bytes:

```
# cfsize -s 10k file.txt
```

Not just one but many files can be specified on the command line as long as you do not exceed the maximum allowable number of command-line arguments for the shell. The following command reduces all logfiles in the current directory to 2 MB:

```
# cfsize -s 2m *.log
```

The cfsize utility works on files only. Standard input (`STDIN`) has no meaning to cfsize and commands like the following should not be used because the program abends:

```
# cat file.txt | cfsize -s 10k
# cfsize -s 10k < file.txt
```

Standard output (`STDOUT`) also has no meaning to cfsize, since the input files provided on the command line are modified *in situ*. Here's an example of what not to do:

```
# cfsize -s 50k input.txt > output.txt
```

This command would reduce input.txt from 250 MB to 50 kB and create a zero-length output.txt file, which is not what is intended.

Let's wrap up this section by going over the application of the "end-of-options" switch. The command to reduce `-file` from 2 GB to 1 MB would be:

```
# cfsize -s 1m -file
cfsize: illegal option -- f
usage: cfsize -s filesize[k|m|g] file ...
```

However, cfsize thinks that it is being passed option `-f` and it terminates abnormally, as it recognizes `-s` as the only valid option. This ambiguity is the reason why the end-of-options switch `--` has been built into cfsize. To correct this pernicious situation, insert the end-of-options switch into the command line right before the processing of any files:

```
# cfsize -s 1m -- -file
```

Bugs and Shortcomings

Chopping a file from the "head" needs one important caveat, because of the way file truncation works. The data that needs to be kept is moved to a temporary file and when the desired file size is reached the original file is replaced with the temporary one. This means that the target directory, that is, the one containing the temporary file, should have enough space to hold the intermediate data; otherwise the whole operation will fail. Note that permissions on the target directory come into play when cfsize is executed without superuser privileges. Another point to keep in mind about cfsize is the fact that it does not support large files, that is, files that are greater than 2 GB. If it were used on a file bigger than 2 GB, cfsize would terminate.

Cfsize has been tried and tested on many UNIX and Linux platforms. It is designed almost exclusively with sysadmins in mind, so while using cfsize, if anyone comes across a bug or feels that redesigning the algorithm, implementing coding shortcuts, or efficiently using system resources can improve the program, please contact me via email. Please do the same if any one of you comes across a tool, besides those mentioned here, that can rival the claims of cfsize.

Conclusions

Cfsize was designed for doing a simple task, that is, reducing the sizes of the files given on the command line. It provides a much-needed respite from storage space woes to sysadmins who up to now had to either compress files or remove them. A third option in the form of a C program named cfsize is now readily available to all system admins. Future plans for cfsize may include revising it to support large files.

JASON DUSEK

concurrent patterns: parallels in system and language design

Jason Dusek is a systems architect with a self-funded startup in the Bay Area. His technical interests include distributed computing, functional programming, and serialization.

jason.dusek@gmail.com

CONCURRENCY AND PARALLELISM ARE usually discussed together, and they are both important in reorganizing a program for execution on multiple cores or multiple computers. Parallelism asks, "How can we break a program up into independently executing parts?" It is very much in the domain of compiler writers and library providers, as the necessary program transformations are tedious at best. Concurrency asks, "How can independent programs share a value once in a while?" It bedevils program designers at every level—providers of parallelism toolkits, operating systems designers, application programmers, system administrators providing distributed network services, and even folks just trying to sell a few books.

Sharing

For programs to share a value, they may both possess a copy or they may both possess a handle.

If they both have a handle, that is cheapest and allows them to communicate most efficiently. For example, with file handles, two applications opening the same file are using less memory than two applications with one sending a copy to the other. They also communicate more quickly. However, handles can result in confusion and danger—for example, memmapped files. If two programs memmap the same file, and the "receiver" is not careful to close it and reopen it at the right time, one can very likely crash the receiver by writing into the memmapped file. The less extreme example of two database clients overwriting parts of one another's changes, resulting in a mutually confusing (but readable) record, is a more common pitfall of these handle-based (we'll call them "shared-memory") approaches to concurrency. Down this path lay transactions and locking. There are numerous forms of locking, the most dreadful being spin-lock, deadlock, and live-lock.

When we pass around copies, each process can pretty much go on its merry way from the point of copy transfer. As mentioned earlier, this can result in a great waste of memory. These "message-passing" strategies are inevitable, though, in network services—an LDAP server cannot share memory with its clients; it has to just send them the data. If we structure all our applications this way, without regard to locality, then it is at least consistent, if not efficient. Garbage collection is very important in message-passing architectures, because you create a lot of garbage by copying things all over the place. However, a program is safe from interruption from other programs as long as it does not get (or request) any messages. The problem of data synchronization can be resolved through frequent polling or publish-subscribe (possible in LDAP, if you are brave)—but at least there is some kind of warning, some kind of formal indicator, that a resource has changed.

At this point you may wonder how the two database clients are sharing memory while the LDAP server and its clients are passing messages? It is really a matter of where I choose to point the camera. Between the servers and clients there is message passing—clients make a request and servers send back a message full of data. Any changes in the server's data do not affect the clients at all as long as they don't ask for updates. Between the two database clients, there is shared state—each application writes to the database and reads from the database to communicate with the other processes. The difference shows up in where we put the locks—a database locks its tables, not its socket.

We Don't Need Concurrency In Programming Languages . . .

It stands to reason (although I cannot prove it) that all models of concurrency are explored in systems first and in languages later. Network services and operating systems offer a kind of parallelism—operating system process, multiple clients, servers running on distinct machines—and thus invite all the problems we usually associate with threading and message passing.

Within the *NIX filesystem, as in other shared-memory systems, we need to protect processes from mutual confusion—we need transactions, locks, and semaphores. The *NIX filesystem models all three. Through file locking, multiple POSIX programs protect one another from contradictory updates. A *shared lock* allows multiple shared locks to coexist with it, whereas an *exclusive lock* can only be taken when there are no other locks. When the locks are advisory—the default for *NIX—the lock is merely a baton, which processes can only possess under certain circumstances. These advisory locks are really semaphores, tokens of resource ownership without enforcement [1, 2]. In contrast, mandatory locks are true locks—processes block if they don't have the right lock. No reads can proceed without some kind of lock, and writes cannot proceed without a write lock. Often, an operating system will provide special signals and subscriptions to allow a process to know that a file is again open for reading, that another process wants to write a file, and other things. We can see how message passing and shared memory complement one another [3].

A transaction is a collection of operations on different resources that is performed in an all-or-nothing manner. To perform a transactions in a *NIX filesystem we must:

- Lock the greatest parent of all the files in the transaction and every subordinate file. (We'll be waiting a long time to get all those locks in an active system.)
- Recursively copy the greatest parent into a new directory.
- Perform our changes.
- Use `mv` to atomically overwrite the old greatest parent with the now updated copy.

Our rather overly generous lock scope has prevented anything from changing in the meantime.

Why must we make such a gratuitous copy? Because the filesystem does not offer a means to atomically mv several files. What if we got halfway through the mvs and the user sent SIGKILL? Then we'd have left the system in an inconsistent state, with no means of recovery. So we have to find a way to mv all the changes at once, and that means we have to do the copy. If we had local version control we could be more flexible, simply reverting changes if there were a failure, while keeping everything exclusively locked so no other processes could read the corrupt data until the "rollback" is complete. (Can we be sure reverts never fail?) Although rollbacks are not required for transactions, we see how they can protect us from over-locking.

As mentioned earlier, *NIX provides "signals," a form of message passing. Sockets, a more general and flexible mechanism, allow processes to communicate with one another by establishing a connection and sending messages back and forth, which corresponds to the *channels* of Hoare's CSP [4, 5]. A one-way channel is a *pipe*, both in Hoare's terminology and in *NIX. Unfortunately, *NIX does not allow us to open multiple pipes to a process, so this is a point where *NIX and Hoare diverge. A named pipe—a FIFO—is like a mailbox, a well-known place to leave a package for another process. Unfortunately, FIFOs allow IO interleaving and thus cannot really be used as mailboxes. But I think you get the idea. So what are signals? Signals are mailboxes—the kernel knows where to put messages for every process. We look up its PID to send it a message. Channels, pipes, and mailboxes can fake one another:

- To get a mailbox from channels, the receiver should simply aggregate all messages from every sender regardless of the channel they came in on. To get channels from mailboxes, we must always send a "return address" as part of the message. The receiver, when reading the messages, uses the return address to know which connection it is handling and where to return results.
- To get a pipe from a channel, only send messages in one direction. To get a channel from pipes, we need two pipes, one in either direction. The sender and the receiver must have the sense to keep the pipes together.
- To get pipes from mailboxes, we can *multiplex* the mailbox. Once again, we use the "return address," but this time, we only use the return address to tell us where the message came from, not how to send it back.

Insofar as they work, *NIX concurrency features support a strategy of composing concurrent systems from operating system processes. This is nice in so many ways—the file system handles garbage collection and caching, the scheduler distributes processes over the hardware, and programs in multiple languages can share resources and concurrency model. However, *NIX file locking is brittle, and the only out-of-the-box message-passing model is channels, by way of sockets. *NIX turns out not to be the best platform for an application that is looking for operating-system-level concurrency—but a search for an alternative leads us far afield.

Bell Labs' Plan 9, an evolution of UNIX with some of the original UNIX team on board, offers pipes in the sense of Hoare's CSP. Through resource multiplexing, the same name in the filesystem refers to different resources for different processes, converting a FIFO to a pipe to avert the IO interleaving that bedevils *NIX FIFOs [6]. We could probably emulate this system on any other *NIX, using bind mounts, union mounts, and pipe polling, but it would not be pretty.

At just about the time of Plan 9's emergence, Tandem's NonStop platform was in decline. NonStop SQL ran on top of the Guardian cluster operating system. In Guardian, every resource—every file, even—was a "process" that could receive messages [7]. Pervasive message passing is the door to easy clustering. NonStop SQL was able to run transactions across the cluster, which is a nontrivial task; and Guardian was able to fail over a process from one machine to another if the need arose [8].

There are numerous "cluster operating systems," but what they address is more a matter of resource usage than concurrent design primitives [9–11]. Plan 9 and Guardian are special because they make message-passing tools available to the application programmer and provide an environment where those message-passing tools are widely used.

Networks services model a few concurrency patterns not mentioned above.

Multi-view concurrency, which we might call transactional copy-on-write, is used in some SQL databases and is a long-tested approach to ACIDity. To read, read. To write, copy everything you wish to read or write, prepare the changeset, and the database will apply it if (only if) nothing that was read has been written since, and nothing that was written has been read or written since [12–14].

IRC is an example of a publish-subscribe message-passing service. We subscribe to the channel and receive change sets to the channel—messages—as they arrive. There is no way to pull "the" channel, though—and so we sometimes miss messages, to the amusement of all [15]. In contrast, OpenLDAP offers publish-subscribe messaging as an optimization on the shared-memory model. A client subscribes to an LDAP subtree and, as changes are made, they are forwarded [16]. However, there is one true tree—the tree on the LDAP server—and we can synchronize with it to ensure our copy is correct [17].

. . . But We Like It

Even when operating-system-level concurrency works, there's some mnemonic load in handling it. Network-service-level concurrency is in some sense simpler, but it also handles less—process management is delegated to the operating system. Concurrent programming languages specify an entire concurrent system: resources and a model for their use, as well as processes and a means of creating them.

Language-level concurrency has tended toward message passing. Early versions of Smalltalk were distinctively message-passing, and Carl Hewitt, the founder of the "Actors model," was inspired by Smalltalk [18]. More recent message-passing languages include Stackless Python, used to implement the massively multiplayer game EVE [19]; Limbo, a project by the team that worked on Plan 9 [20]; and Erlang [20]. The former two are channel languages, whereas Erlang is a mailbox language. MPI, a message-passing library and runtime for C, Java, C++, O'Caml, and Fortran, brings message-passing to languages that do not have it natively [21, 22]. Shared memory is an unusual paradigm at the language level.

Erlang, a Message-Passing Language

Erlang, superficially similar to Prolog, makes RPC a language primitive. Processes (function calls) can send messages, listen for messages, and access their present PID. They perform message sending and nonconcurrent things (e.g., math and string handling and ASN.1 parsing) until they hit a receive statement, which is rather like a case statement, only with no argument. The argument is implicit—the next message in the mailbox. After a message is handled, the executing function may call itself or another function recursively, or it may do nothing, which ends the process.

Erlang processes run within instances of the Erlang virtual machine, called nodes. Nodes are clustered to form applications that run on several machines at once. Although Erlang is often called a functional programming language, this is missing the point. Erlang offers easy IPC, a notion of process hierarchy and identity, rapid process spawning, and excellent libraries for process control and monitoring. Erlang is what the shell could have been.

Processes register with a cross-node name server so that they can offer named services to other processes. Processes are "linked" to other processes in the sense that a parent is made aware of the linkee's exit status. The combination of global name registry and linking allows us to implement reliable services in the face of "fail fast" behavior. Processes are arranged with a monitor—an error handler—linked to a worker. The worker runs a function that registers itself for the global name and then calls the main function that handles requests and recursively calls itself over and over. The monitor hangs out. If the server process dies, the monitor receives a signal to that effect and recalls the function.

The special thing about Erlang is not that we can write programs this way, but that we write *every* program this way. Programs are collections of communicating services in Erlang, not a main thread of execution with subroutines (at least, not on the surface.) The innumerable executing threads are easy to parallelize, because they share no state with one another and thus can be interrupted and resumed at any time. When we see Erlang trumpeted as the multi-core solution, that is why.

Erlang's bias toward concurrent design is perhaps too great. Although the standard libararies are rich in protocol handlers and design patterns for concurrent applications, the language is weak at string handling and arithmetic.

Haskell's Approach to Shared Memory

Programming languages that offer safe access to shared memory are rare. Haskell, a purely functional language, offers "Software Transactional Memory," which is much like multi-view concurrency in databases.

Haskell's type system allows it to make very strong guarantees about shared-memory operations. A pure function is a function that yields the same answer for the same arguments [e.g., sin(x)] whereas an impure function—hereafter a procedure—returns different things depending on the context [e.g., `gettime()`].

How do we perform input and output and get the time of day in a purely functional language? It turns out that there is a "pure view" of these impure operations. The principle is not hard to understand—values from impure functions are wrapped in a special container—so `gettime` does not return an `Int`; it returns an `IO Int`, an `Int` within the `IO` container.

A pure function, even when it shares state with another pure function, is not allowed to mutate it. It is immaterial in which order we evaluate the pure functions, so long as we evaluate calls that a function depends on before evaluating the function itself. This no-mutation property makes parallelization easy, and the Haskell compiler takes advantage of that.

So we have all these functions, and they are likely to execute at any time. They do not have process identifiers, and there is no global registry of names. How do these functions communicate? One means, an early one, was to in fact brand every concurrent function with `IO` and force it to operate on references. This brings us all the problems of shared memory and none of the solutions; it also forces the compiler to be as paranoid about code accessing references as it is about code accessing the filesystem or network. The Glasgow Haskell Compiler project later introduced a new container, STM, which is specifically for operations that work on a large global store of values [24]. A procedure that executes within STM creates a log of its reads and writes. When the procedure ends, the log is checked to ensure that none of the values have changed since the procedure read them. As long as the reads are okay, the writes in the log are committed. If they are not okay, the procedure retries until it works (including forever).

STM offers true fine-grained transactions on a runtime system that can run millions of threads, but there are no provisions for clustering or process hierarchy. This is in some sense inevitable; shared memory systems don't dovetail nicely with the network's natural separation of state across servers.

Concurrency and You

Splitting a program into concurrently running parts can simplify design and always parallelizes the program. Whether this results in a net performance benefit depends on the overhead of communication.

To take advantage of concurrent design and implicit parallelism, one must adopt a new way of thinking about program structure, data structures, and efficiency. In-place update is efficient and natural in sequential computing, but in concurrent systems it is fraught with peril and obstructs parallelism. Bolting concurrency onto a sequential language—an imperative language—leads to inconsistency at best, and so it is understandable that much work in concurrency has taken place under the declarative tent.

Concurrent Perl would find many users if it existed, and there have been numerous attempts to bring concurrent programming to C and C++. Concurrency-friendly languages are unusual languages, bringing more or less of the functional paradigm with them. Will a new language gain a foothold, or will we find a way to bring message passing or transactional memory to C?

Perhaps, like object-oriented design, concurrent programming will find wide use only after it has been integrated with C. Toolkits for parallelism in C, C++, and Java are certainly catching up, although they are used mostly by game programmers and authors of scientific visualization software. Languages used mostly in Web programming and system administration have not received that kind of attention, and consequently we see the growth of Erlang in the area of high-availability network services. Niche programming languages will always have their niche, and it's likely that mainstream programming languages will always have the mainstream.

REFERENCES

[1] fcntl man page: fcntl - manipulate file descriptor: http://linux.die.net/man/2/fcntl.

[2]Semaphores: http://www.ecst.csuchico.edu/~beej/guide/ipc/semaphores.html.

[3] Andy Walker, "Mandatory File Locking for the Linux Operating System," April 15, 1996: http://www.cs.albany.edu/~sdc/Linux/linux/Documentation/mandatory.txt.

[4] C.A.R. Hoare, "Communicating Sequential Processes: 7.3.2 Multiple Buffered Channels," June 21, 2004: http://www.usingcsp.com/.

[5] Sean Dorward, Rob Pike, David Leo Presotto, Dennis M. Ritchie, Howard Trickey, Phil Winterbottom, "The Inferno Operating System": http://doc.cat-v.org/inferno/4th_edition/inferno_OS.

[6] Rob Pike, "Rio: Design of a Concurrent Window System," February 4, 2000: http://herpolhode.com/rob/lec5.pdf.

[7] The Tandem Database Group, "NonStop SQL, a Distributed, High-Performance, High-Availability Implementation of SQL," April 1987: http://www.hpl.hp.com/techreports/tandem/TR-87.4.html.

[8] Wikipedia, Tandem Computers: http://en.wikipedia.org/wiki/Tandem_Computers#History.

[9]: C. Morin, R. Lottiaux, G. Vallee, P. Gallard, D. Margery, J.-Y. Berthou, I. Scherson, "Kerrighed and Data Parallelism: Cluster Computing on Single System Image Operating Systems," September 2004: http://www.ics.uci.edu/~schark/cluster04.ps.

[10] Wikipedia, QNX: http://en.wikipedia.org/wiki/QNX.

[11]: Nancy P. Kronenberg, Henry M. Levy, William D. Strecker, "VAXclusters: A Closely-Coupled Distributed System," May 1986: http://lazowska.cs.washington.edu/p130-kronenberg.pdf.

[12] Wikipedia, MultiView concurrency control: http://en.wikipedia.org/wiki/Multiversion_concurrency_control.

[13] David Patrick Reed, "Naming and Synchronization in a Decentralized Computer System," September 1978: http://publications.csail.mit.edu/lcs/specpub.php?id=773.

[14] Philip A. Bernstein and Nathan Goodman, "Concurrency Control in Distributed Database Systems," June 1981: http://www.sai.msu.su/ ~megera/postgres/gist/papers/concurrency/p185-bernstein.pdf.

[15] RFC 1459: IRC Concepts: http://www.irchelp.org/irchelp/rfc/chapter3.html.

[16] LDAP for Rocket Scientists: Replication refreshAndPersist (Provider Push): http://www.zytrax.com/books/ldap/ch7/#ol-syncrepl-rap.

[17] LDAP for Rocket Scientists: Replication refreshOnly (Consumer Pull): http://www.zytrax.com/books/ldap/ch7/#ol-syncrepl-ro.

[18] Alan Kay, "The Early History of Smalltalk: 1972-76—The First Real Smalltalk: http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html.

[19] About Stackless: http://www.stackless.com/.

[20] Limbo: http://www.vitanuova.com/inferno/limbo.html.

[21] Concurrent programming: http://www.erlang.org/doc/getting_started/conc_prog.html.

[22] Wikipedia, MPI: http://en.wikipedia.org/wiki/Message_Passing_Interface.

[23] MPI asics: http://www-unix.mcs.anl.gov/dbpp/text/node96.html.

[24] Simon Peyton Jones, "Beautiful Concurrency," December 22, 2006: http://research.microsoft.com/~simonpj/Papers/stm/beautiful.pdf.

DAVID N. BLANK-EDELMAN

practical Perl tools: attachments

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Perl for System Administration*. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs.

dnb@ccs.neu

"From attachment springs grief, from attachment springs fear. From him who is wholly free from attachment there is no grief, whence then fear?"

—*The Dhammapada: The Buddha's Path of Wisdom*, translated from the Pali by Acharya Buddharakkhita

THE BUDDHA WAS PROBABLY NOT talking about mail attachments in the Dhammapada. But there's enough fear and loathing around mail attachments that it would be worthwhile to see whether we can use Perl to reduce the suffering around this issue. In this column we're going to look at three different tasks related to attachments and how to work through each using some precision Perl modules.

Before we get started we first have to understand a little background information about attachments. To have an (interoperable) attachment, a mail message has to be constructed in a certain way. The blueprint for how this message is constructed is defined using the Multipurpose Internet Mail Extensions (MIME) standards. I say "standards" not simply to watch Nick Stoughton's ears perk up (hi, Nick!) but because it takes at least six documents to construct this particular snake pit (there are more):

- RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N. Freed and N. Borenstein, 1996.
- RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed and N. Borenstein, 1996.
- RFC2047: MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, K. Moore, 1996.
- RFC2077: The Model Primary Content Type for Multipurpose Internet Mail Extensions, S. Nelson and C. Parks, 1997.
- RFC4288: RFC 4288—Media Type Specifications and Registration Procedures, N. Freed and J. Klensin, 2005.
- RFC4289: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, N. Freed and J. Klensin, 2005.

Don't run screaming yet; wait for another paragraph or two. Let me give you the four-sentence summary of these documents necessary to understand the rest of the column:

MIME messages are composed of "parts" surrounded by separators in the text. Each part is labeled with a type and a subtype to help the mail client understand what kind of part it is (is it a picture? a movie? a pdf file? etc.). If a part contains information that can't be represented in ASCII (e.g., a .jpg or a .pdf file), it is encoded into a format that can be represented that way. Oh, and for extra special fun, parts of a MIME message can contain other parts (e.g., you forward someone an entire mail message as an attachment that itself had attachments).

This doesn't sound so bad until you start to read the specs and realize that there are so many edge cases and places where different people can read the specs in different ways. Add on top of that the beautiful agony of HTML messages (MIME messages with HTML parts that a mail reader is expected to display instead of plain text) and you'll quickly realize why many a mail client author has gone barking mad.

Hopefully I've scared you away with this preface so I can simply copy Lorem Ipsum text into the rest of the column instead of having to write more about it. On the off-chance you are still reading, let's take a look at the first task.

How Do You Send Email with Attachments?

Since misery loves company, it makes sense to first figure out how we can spread the MIME joy. It didn't used to be this way, but people who program in Perl are now in the (enviable?) position of having quite a few good modules for sending email messages with attachments. They fall into roughly two categories: those that let you play Vulcan and forge the bits yourself and those that make as many decisions as possible for you so you can wave an impatient "just do it" hand. We'll look at examples from both categories. I should mention that the choice of module for each category is based on my experience, but I would encourage you to look at the other possible choices. For example, Mail::Sender, MIME::Lite, MailBox, Mail::Builder, and MIME-tools all have things going for them that make them worth your consideration.

I've become fond of the mail-related modules from the Perl Email Project (http://emailproject.perl.org/), so the example from the do-it-yourself category is Email::MIME::Creator with some help from Email::Send to actually send the message. Email::MIME::Creator can be considered a bit of an add-on to the more general-purpose Email::MIME package. It is designed specifically for the creation of new MIME messages. Let's see a very simple example of it in action. The first part of the procedure is to load our modules and then create the MIME parts we'll need for the message. In this case we'll need a MIME part for the body of the message and another for the picture we're going to attach:

```
use Email::Simple;
use Email::MIME::Creator;
use File::Slurp qw(slurp);
use Email::Send;

my @mimeparts = (
   Email::MIME->create(
      attributes => {
         content_type => 'text/plain',
         charset => 'US-ASCII',
      },
      body => 'Sending mail from ;login: magazine.',
   ),
   Email::MIME->create(
      attributes => {
         filename => 'picture.gif',
         content_type => 'image/gif',
         encoding => 'base64',
            name => 'picture.gif',
        },
        body => scalar slurp('picture.gif'),
    ),
);
```

Each call to `Email::MIME->create()` creates a new MIME part. We need to specify the MIME attributes and the data for each part. In the second call above we use File::Slurp to easily set the body field to the contents of the GIF file being sent.

Once we have the parts made, we create the actual mail message, this time with an `Email::MIME->create()` call that includes the headers for that message and an anonymous array with pointers to the MIME parts created in the previous step:

```
my $message = Email::MIME->create(
    header => [
        From   => 'loginauthor@usenix.org',
        To => 'dnb@usenix.org',
        Subject => 'Email::MIME::Creator demonstration',
    ],
    parts => [@mimeparts],
);
```

Email message in hand, we can now use Email::Send to actually send it out by handing it to the MTA on our machine:

```
my $sender = Email::Send->new({mailer => 'Sendmail'});
$Email::Send::Sendmail::SENDMAIL = '/usr/lib/sendmail';
$sender->send($message) or die "Unable to send message!\n";
```

And away it goes . . .

If you don't want to bother putting together a MIME message part by part, there are modules such as Email::Stuff available. The name sounds pretty casual, and so is the module. It lets you send the same kind of message with just a single line:

```
use Email::Stuff;

# to give the underlying module a clue where to find the MTA binary
$Email::Send::Sendmail::SENDMAIL = '/usr/lib/sendmail';

Email::Stuff->from('loginauthor@usenix.org)
    ->to('dnb@usenix.org')
    ->text_body('Sending mail from ;login magazine.')
    ->attach_file('picture.gif')
        ->send;
```

And away this message goes . . . Is this easier to use? Sure. But it doesn't give you the same level of control as Email::MIME::Creator. This may or may not be important to you.

How Do You Deal with Attachments You've Received?

Now that I've shown you how to send email with attachments, all of the cool kids will want to do it too, and soon you'll be awash in a sea of messages. Your first idea might be, "Hey, these attachments must all be valuable, how do I save them all?" To indulge this naive response, we could use one of the generic MIME processing packages such as Email::MIME or MIME-tools, but here too there is at least one module that is precisely targeted to the task: Email::MIME::Attachment::Stripper. Here's some sample code:

```
use Email::MIME;
use Email::MIME::Attachment::Stripper;
use File::Slurp qw(slurp write_file);

my $m = Email::MIME->new( scalar slurp 'message.eml' );
my $s = Email::MIME::Attachment::Stripper->new( $m, \
        'force_filename' => 1 );

foreach my $attachment ( $s->attachments ) {
   write_file( $attachment->{filename},
      { buf_ref => \$attachment->{payload} } )
   or die "Can't write $attachment->{filename}: $!\n";
}
```

After loading the modules we parse the email message into an Email::MIME object. This isn't strictly necessary, because Email::MIME::Attachment::Stripper can take other formats if the right module is in place, but I prefer not to leave that to chance. The Email::MIME object is then fed to Email::MIME::Attachment::Stripper so it can do its stuff. We give that call an extra argument of `force_filename` because we want the module to either extract a filename from the message or, if the sending client was sloppy and didn't include one, to make it up. This is important because the next few lines of code expect to be able to write to a file with a specific name.

We then ask Email::MIME::Attachment::Stripper to provide a list of the attachments it has found. It returns a list of hashes, with each hash containing both the information about that attachment and the actual data sent. This payload gets handed off to `File::Slurp's write_file` method and a file is created with that information in it.

One quick related aside: The code is very trusting. It takes the name of the file from data someone else has sent us and is happy to create files with whatever name it is handed. In general you should make your code a little less sanguine and have it only allow filenames that pass some sort of vetting process.

Now, what if you quickly tired of receiving copies of cutepuppy.jpg but wanted to keep the messages sans their attachments? Email::MIME::Attachment::Stripper has a `message()` method that will hand you back an Email::MIME message object that represents the original message without any of the attachments. That's one way of going about the task, but I want to show you another, perhaps more interesting method that gets us a more sophisticated result.

If you've ever had to deal with spam from a mail administrator's perspective, it is entirely likely that you've crossed paths with the open source package Apache SpamAssassin (http://spamassassin.apache.org/). The basis of this package is a set of Perl modules called Mail::SpamAssassin(::*). There are two things that many people don't know about this module set: (1) It contains a really robust MIME parser (because it has to, since spammers throw all sorts of malformed data at it); (2) that parser and some other very handy utility methods can be used for other purposes. If you haven't read the documentation for Mail::SpamAssassin::PerMsgStatus yet, you should.

One of the utility methods related to this column is `get_decoded_stripped_body_text_array()`. This method (according to the doc) "returns the message body, with base-64 or quoted-printable encodings decoded, and non-text parts or non-inline attachments stripped" and "HTML rendered, and with whitespace normalized." If you've ever wanted the "essence" of a message (e.g., for indexing), this method can provide it. It is used like this:

```
use Mail::SpamAssassin;
use File::Slurp qw(slurp);

my $sa = Mail::SpamAssassin->new();

# check_message_text actually attempts to determine the text is spam
my $status = $sa->check_message_text( scalar slurp 'message.eml' );

# but we need its status response, with which we can:
my $body = $status->get_decoded_stripped_body_text_array();

print @{$body};
```

How Do You Know If an Attachment Is, Umm, Icky?

I suppose there is both a Buddhist and a technical answer to this question. Let me try to address the latter. Attachments get a bum rap because they are a major vector for viruses and other malware.

If you want to determine whether an attachment is unsavory, you usually need to pass it through some other package that tries through a variety of methods to determine whether it is unpleasant or not. In the open source world, one very popular package is ClamAV (www.clamav.net). There are a few Perl packages that are designed to use the ClamAV engine. I'd like to mention three of them because they each take a different approach, only one of which may be appropriate for your needs:

- ClamAV::Client talks to a running ClamAV daemon for its scans. This works great if you already are using ClamAV in some fashion or would like something higher-performing than the next module.
- Mail::ClamAV is a Perl wrapper around the ClamAV scanning library API. This is good if you want your Perl code to actually perform the scan or need fine-grained control over the options used for a scan. If your code is going to scan a file and quit each time it is run it is probably going to be less efficient than ClamAV::Client because it needs to spin up the ClamAV engine and load the virus database each time. If your code performs the ClamAV initialization work and then begins to scan lots of files at a time, it is probably a wash.
- File::VirusScan is a "unified interface for virus scanning of files/directories," according to the documentation. It is a wrapper around quite a few (13 as of this writing) commercial and open source anti-virus packages. The same Perl code can scan a file or directory using one or even several anti-virus engines (in order). This is useful if you plan to use a battery of anti-virus checkers or if you'd like to write code that isn't tied to any one of their APIs.

Here's an example of using ClamAV::Client to check the attachments we get back from Email::MIME::Attachment::Stripper:

```
use Email::MIME;
use Email::MIME::Attachment::Stripper;
use File::Slurp qw(slurp);
use ClamAV::Client;

my $msg = Email::MIME->new( scalar slurp 'message.eml' );
my $strip =
   Email::MIME::Attachment::Stripper->new( $msg, 'force_filename' => 1 );
```

```
# we can also connect to a ClamAV clamd listening over a TCP socket if we

# use different options
my $scan = ClamAV::Client->new( socket_name => '/var/run/clamd-socket' );

die "unable to talk to ClamAV daemon"
   unless defined $scan and $scan->ping();

my $ans; # the results back from ClamAV
foreach my $attach ( $strip->attachments ) {
   $ans = $scan->scan_scalar( \$attach->{payload} );
   ( defined $ans )
       ? print "$attach->{filename} is infected with $ans!\n"
       : print "$attach->{filename} is clean.\n";
}
```

The Email::MIME::Attachment::Stripper lines of this code should be familiar from the previous examples. The key new thing we added was the call to ClamAV::Client's `scan_scalar` that passes the contents of a scalar pointed to by a scalar ref off to a ClamAV daemon for processing. If the answer it gets back is anything but `undef` (clean), the code prints the name of the infection as identified by the daemon.

Now that you know a few ways to evaluate your attachments, it's time to end the column. Take care, and I'll see you next time.

PETER BAER GALVIN

Pete's all things Sun: Solaris System Analysis 102

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 coauthor of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://pbgalvin.wordpress.com and twitters as "PeterGalvin."

pbg@cptech.com

RECENTLY I WAS HELPING OUT A FRIEND, a CEO at a small business, who had her main system running without a backup. As we all know, friends don't let friends compute without backups. Given that the system was an Apple Mac, it was a trivial matter to attach an external drive and push the couple of buttons needed to execute a backup. When I was done she was rather surprised and asked if that was all there was to it. Was computer administration really that easy? After pondering a second I came up with a fundamental statement about system administration: It's easy, except when something goes wrong, and then it can be very, very challenging.

For a sysadmin, a good day can turn into a very bad day with just a few words: "The system has a problem." Such problems, especially ones of performance or reliability, can be difficult to solve. In fact they can be the most difficult task a sysadmin performs.

The goal of the previous "Pete's All Things Sun" column, "Solaris System Analysis 101," was to put a stake in the ground about the first steps that should be taken when a system has "a problem." The hope is that you, the sysadmin reader, will contribute to it, creating a consensus document. Given that we live in the time of Web 2.0, a wiki seemed like the best way to foster contributions, and that wiki is now live [1]. Please have a look and contribute your wisdom and knowledge for the betterment of sysadmin-kind.

That leads us to this column, "Solaris System Analysis 102." Once the 101 steps are taken, what can be done to determine the specific cause of the problem and fix it? The previous column was mostly operating-system-independent. Almost all of the ideas there apply to all operating systems equally. In this column that will not be the case. Here, then, are specific steps I use to analyze a Solaris system and determine the cause of the problem. Most of these commands are Solaris-specific, including DTrace code. This column will also be added to the wiki, allowing you to comment, correct, and expand. Please do so! In the future, watch for BoFs and other activities at USENIX conferences about this topic.

Phase 1: Search for the Smoking Gun

Sometimes the system has a large, easy-to-find problem. In those cases it would be a shame to spend a lot of time chasing down complex paths. Rather, the first step is to check for obvious problems with the "usual suspect" commands. The goal of this phase is to narrow the problem area to a specific aspect of the system.

Solaris System Analysis 101 ended with a list of areas to explore. Here are some more specifics:

- Scan through log files such as /var/adm/messages and via `dmesg`. Don't ignore anything odd: It could be the canary indicating the problem.
- Run `svcs -a` to check for services that have failed or are disabled.
- Check for full disks or changed mount information via `df` and `mount`.
- Run `ifconfig -a` and look for any errors; run `kstat` and read through the section of output of a given network interface (such as e1000g0) to check network parameters such as duplex and speed.
- Read through /etc/system and look for settings copied from other systems or left behind during an upgrade. /etc/system should never be copied or left intact between operating system or application upgrades; such events should cause an audit of the file for entries to remove or update. Check the *Solaris Tunable Parameters Reference Manual* [2]. This document is updated for every Solaris release. Watch out for system setting recommendations from vendor documents.
- Check /etc/projects for any resource management settings that could be affecting system or application performance.
- Check the load average of the system. `uptime` shows the 5-, 10-, and 15-minute average number of threads wanting to run on the system. If those numbers are significantly (two times or more) higher than the number of cores in the system, users will report "slowness."
- Check the stat commands and look for anomalies. Note that the first set of output is averages per second since the system booted. The following sets are averages per second since the previous set of output. As with all of these commands, understanding the output and the underlying system is key.
- Check `iostat -x 10` and check the `svc_t` column for large service times (in milliseconds). Anything above 30 ms can be of concern. Also note that dividing kilobytes written per second by writes per second produces the average write size during that period, which can help when analyzing I/O issues. The same applies to the read values (`r/s` and `kr/s`).
- Check `mpstat 10`: How was processor time spent? Per CPU (each row being a CPU's status), what percentage of time was spent in user-land (running user code) (`usr`), how much in the kernel (`sys`), and how much idle (`idl`)? Most time should be `usr`, and any more than a few percent in the kernel can indicate a problem.
- Check `vmstat 10`: How many threads are running or want to run (`kthr r`), how many are blocked waiting for something (usually I/O) (`kthr b`), and how many processes have been swapped out (`kthr s`)? Swapped out means that the system was desperately short of memory and booted entire processes out to disk. That's bad. Also check `page sr`, the scan rate, to see whether the system is short of memory. The larger this number, the more the system is hunting for memory. Anything above 0 is considered a memory shortage. Memory is orders of magnitude faster than disk, so any use of disk as virtual memory can cause a system slowdown.
- Check `vmstat -p 10`. This shows system-wide memory operations. This is the place to check whether the system is short on memory and to

determine which system aspect is using the memory [executable process pages, anonymous (heap, stack, or malloc) uses, or file system I/O].

- Check prstat. If the problem is simply processes using up CPUs, then prstat can show which processes those are. What is more difficult is figuring out what the process is doing and whether it should be doing it.
- Check prstat -Lmp <pid>. This shows detailed state information about a specific process at the current time. If the process has multiple threads it shows a row per thread. Columns 3 through 9 (USR through SLP) add up to 100%, showing the percentage of time the thread spent running in user mode (USR), in kernel mode (SYS), and so on.
- Use pmap -x <pid>to explore the memory map of a problem process.
- Use DTraceToolKit and DTrace scripts to look at specific suspect aspects of the system.

Phase 2: Finding the Owner of the Gun

With the Phase 1 rough data in hand, did you find the problem? User-level problems are relatively easy. If a process is using too much CPU or memory and you have the source code, it is now a program development and debugging problem. If the application is well written, then perhaps the only solution is adding resources to the system to allow the application to match your performance needs. For home-grown code, be sure to use the latest version of a given compiler. Also note that Sun's SunStudio development environment is now available [3] for free (without support), generates great code, and has good debugging tools built in, including the DTrace-based D-light tool and "performance analyzer" functionality. Also, at least with Solaris, each release usually brings about performance improvements. If you are running an older version of Solaris, consider the (difficult) step of upgrading. In addition, Java code is a major component in many applications, and Java can be difficult to performance-analyze and tune. Try to use the latest JVM, especially because Java 1.5 adds DTrace support and Java 1.6 automatically optimizes garbage collection.

If the problem is at the system level, then more time (and commands) may be needed to track down the problem. The good news is that Solaris 10 has many more tools than previous Solaris releases (and other operating systems in general) to find and fix these problems.

Some higher-level system areas to consider include:

- Are you running the most appropriate scheduler for each system in your environment? Solaris defaults to time-share scheduling for user processes. If your system is a server that doesn't run general user tasks, then time-sharing is overkill with more overhead. If you want all processes on the system to have the same priority (not changing as time-sharing does based on CPU used and I/O requested), then consider changing to the much lower-overhead fixed priority scheduler "FX." Such a change could buy you 5% or more CPU time. To make FX the default class execute dispadmin -d FSS. That change is persistent across reboots. To move current processes from time-sharing to FX, use priocntl -s -c FX -i class TS.
- If the problem involves some processes starving others of resources, consider implementing the fair-share scheduler and resource management. Those can be implemented either for the full system or, more easily, per-zone when zones (a.k.a. containers) are installed on a system. There is a lot to resource management and zones, as has been covered previously in *;login:*. The slides from my tutorial on Solaris 10 adminis-

tration have all the gory details and are freely available online [4]. There are links to this and other resources at my blog [5].

- If there are high-priority processes on a system, consider "pinning" them to a set of CPUs. These processes will stay on those CPUs and not be rescheduled or interrupted. A good time to use this technique is for database servers or just for the log-writing process of a database. The Solaris tools to use here are processor sets and process bindings.
- Are you using the best-fit page sizes? Having the same sizes of I/O operations from memory through to the physical disk is one key to good I/O performance. For example, OLTP databases such as Oracle's frequently perform I/O in 8-kB chunks. If you format your disks to use 8-kB sectors, I/O will be streamlined. Be sure to take into account the underlying disk structures (i.e., if you have a SAN, understand the I/O geometry within the LUNs that are provided). Note that terminology of disk structures varies, but ZFS calls its I/O chunk the "recordsize." In this Oracle example, set a ZFS recordsize to 8 kB, and, for good performance, make sure that the underlying storage array has RAID sets that are multiples of 8 kB. Jiri Schindler wrote a very in-depth analysis of matching application and device I/O patterns in his PhD thesis [6].
- Is your I/O well-balanced and spread across enough devices (e.g., disks and network ports)? In general, I/O is the most likely bottleneck, disk I/O the most likely I/O culprit, and individual disks the most limiting I/O device. Any given disk can perform 100 to 200 I/O operations per second (IOPS). If your system needs to do thousands of IOPS, then you need tens of disks, well tuned, to provide that I/O. RAID 0+1 and 1+0 are better-performing than RAID 5, so match the RAID level with the performance needed.
- Are you using the best CPU for the workload? Sun has two product categories: The first includes the "X" and "M" servers, which run a few threads very fast. The "T" servers are chip multi-threading (CMT) systems and run lots of threads, but run them rather slowly. An analogy can help sort out the best uses for these systems. Think of the "X" and "M" servers as race cars and the "T" servers as trucks. Each has its uses, so make sure you use the right system for the needed performance. Also, there are several steps that can be taken to determine whether a "T" server is right for your applications and to tune these servers. Sun's Web site [7] is the best place to start.

As always, benchmarking is the best way to test performance and performance changes, if the benchmarking is accurate and repeatable. Watch out especially for caching effects in benchmark efforts. Caching happens at all levels of computer systems, so, for example, it is safest to reboot the systems involved between each benchmark run. Consider, however, that SAN arrays also have caches, which could invalidate (or at least complicate) benchmark results.

Run Forensics on the Gun

Once the range of the problem has been narrowed, specific analysis can be done on the problem area to ferret out the source of the problem. DTrace is a fabulous tool for this analysis.

The DTraceToolkit provides over 200 prewritten (but unsupported) tools for getting detailed information about the operation of many areas of the system. Get familiar with the tools so they are in your arsenal when needed. The scripts are well documented and demonstrated online [8], so I won't repeat that information here.

Beyond the DtraceToolkit, the sky is the limit for delving into system activity details. For example, here is sample code to graph the time spent in each system call by each process:

```
syscall:::entry
/uid != 0/
{
self->tm = timestamp
}
syscall:::return
/self->tm/
{
@[execname, pid, probefunc] = quantize(timestamp - self->tm);
self->tm = 0
}
```

In another example, processes starting and exiting immediately can be difficult to spot and can greatly decrease system performance. Find them by the command line `/usr/sbin/dtrace -n 'proc:::exec{printf("%s execing %s, , uid/zone =%d/%s\n",execname,args[0],uid,zonename)}'`.

Another previously hidden performance hit is error management. Detect and fix failing system calls before moving forward, as that will change your performance picture. A DTraceToolkit tool, errinfo, displays all system call errors.

For I/O, to display files and the I/O being done to them execute `/usr/sbin/dtrace -n 'io:::start{@[execname, args[2]->fi_pathname] = count()}'`. To determine the block size execute `/usr/sbin/dtrace -n 'io:::start{@[execname, args[2]->fi_pathname] = quantize(args[0]->b_bufsize)}`.

To determine the level of multi-threading of the applications on the system execute `/usr/sbin/dtrace -n 'profile:::profile-100hz /pid/{@[pid, execname] = lquantize(cpu, 0, 512, 1);}'`.

Networking can also be a bottleneck, as even multiple 1-Gb links can be slower than other system aspects. Even with Solaris 10, network bottlenecks can be difficult to spot owing to the lack of a DTrace networking provider. That provider was included in Solaris Nevada build 93, so it should appear in a future Solaris release. For details see Sun's wiki [9]. In the meantime a good tool is nicstat, also available online [10].

If the information in this column helped determine the problem but didn't provide a solution to the problem, then it is time to drill down further into the specific problem area. The resources listed below should help with that.

Next Time

If the OpenSolaris Distribution (project Indiana) meets its release goals, then the first production release will be done before the next issue of *;login:*, and that should a rich topic for the next PATS column.

Resources

Very good information for drilling down into each Solaris area of performance tuning is available at http://www.solarisinternals.com/wiki/index.php/Solaris_Internals_and_Performance_FAQ.

A good paper about specific detailed aspects of Solaris performance problem resolution is "Performance Analysis Using DTrace," by Benoit Chaffanjon, at http://opensolaris.org/os/project/sdosug/past_meetings/Performance_Analysis_Using_DTrace.pdf.

REFERENCES

[1] http://wiki.sage.org/bin/view/Main/AllThingsSun.

[2] http://docs.sun.com/app/docs/doc/817-0404.

[3] Sun's SunStudio development environment is available at http://developers.sun.com/sunstudio/.

[4] http://www.galvin.info/2006-11.s10admin.zip.

[5] http://pbgalvin.wordpress.com.

[6] http://www.pdl.cmu.edu/PDL-FTP/Database/CMU-PDL-03-109.pdf.

[7] http://www.sun.com/bigadmin/topics/coolthreads/.

[8] The best starting point for the toolkit is http://www.brendangregg.com/dtrace.html.

[9] http://wikis.sun.com/display/DTrace/ip+Provider.

[10] http://www.brendangregg.com/K9Toolkit/nicstat.

DAVE JOSEPHSEN

iVoyeur: *you* should write an NEB module.

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

dave-usenix@skeptech.org

The Nagios source code can be downloaded from http://www.usenix.org/publications/login/2008-10/nagfs.c.

FOUR YEARS AGO, I ATTENDED THE Nagios BoF at LISA '04 in San Diego. It was being thrown by a few employees from Groundwork, including Taylor Donditch, the author of fruity. Despite the fact that the BoF was a day before the tech sessions started and therefore not on the official BoF schedule, it was standing room only. For me this was an amazing contrast from 2001, where I mentioned Nagios in a network monitoring BoF and was met by blank stares.

In 2004, Nagios 2 was a fairly new beast, and Taylor was excitedly waxing prolific about the Event Broker interface. That night, all questions to Taylor led back to the Nagios Event Broker. Improved passive checks? NEB. Scalability problems? NEB. Goldfish dead? NEB. This was for good reason: The NEB put a lot of power in the hands of the sysadmin and promised to eliminate or at least reduce the myriad influx of Nagios-related Perl kludges on Nagios Exchange. If you had a problem with Nagios, there was now a correct way to fix it, and that was to write an NEB module. There was no doubt in my mind that everyone in that room would hurry off straightaway and create all sorts of interesting and useful event broker modules. I knew that by morning a wiki would have appeared somewhere, with 30 or so of them a template for making your own and a Web comic making fun of people who used them. For my own part, inspired, I immediately dove into the NEB headers (well, three or so days later, once I sobered up).

Four years later, the Perl kludges have only grown in number, whereas the NEB modules are nowhere to be found. I find this surprising and unfortunate, because the NEB is an elegant solution, and it has the potential to help far more sysadmins per line of code than any script you'll likely find on Nagios Exchange today. For once, the folks who wrote the application recognized our need to customize their program and actually engineered their app in such a way that it can be fairly easily modified to suit our needs. Further, the mechanism they've created is as portable as the app itself, and it could easily do for Nagios what the distros did for Linux. Today, I can internally modify Nagios to customize for a particular problem domain and distribute my customized Nagios in the form of a small piece of shared-object code that can be switched on or off by anyone who uses Nagios. That's cool.

So since Rik tells me this issue will be available at LISA, I'd like to honor the 2004 Nagios BoF by taking some time to explain how the NEB works and hopefully inspiring some of you to sober up and scratch some of your Nagios itches by writing NEB modules. First, I'll give a short description of the architecture, and then we'll walk through a working NEB module I wrote called nagfs, which implements a filesystem interface to Nagios. Since I'll be using my own code as an example, I'll be stuck talking about Nagios 2.x in this article. That's a bit of a bummer because several very empowering changes have been made to the architecture in 3.x; perhaps I'll cover those in a follow-up next time.

The Event Broker itself is a software layer between Nagios and small, user-written shared object files called event broker modules. The Event Broker initializes all of the modules when Nagios first starts, so it knows what events the modules are interested in. Then it sits around waiting for interesting events to occur, passing out memory handles for each interesting event to the module that is interested.

NEB modules are shared libraries written in either C or C++. The NEB module registers for the types of events it is interested in and provides function pointers to functions that presumably do things with the events they receive. Each NEB module is required to have an entry and exit function and, beyond that, can do pretty much anything it wants. The interesting thing about this architecture is that Nagios globally scopes just about everything (by design), so from the perspective of the NEB module, the sky is the limit.

That is to say, because pretty much all the interesting functions and structs are globally scoped, as long as Nagios's execution pointer is in the module's address space the module has the power to change anything it wants. It can, for example, insert and remove events from the scheduling queue or turn on or off notifications or do things such as preempt given check commands or postprocess returned data from service checks. In a nutshell, anything that can be changed at runtime can be changed by the module. Strictly speaking, the module need not even register to receive events; upon initialization the module could schedule its own call-back routines in a timed fashion and do its job using nothing other than Nagios's scheduling engine. It could, for example, wake up every morning and change the value of the day-pager's email address, or wake up every 5 seconds and provide state information to a visualization front-end.

So what sorts of events can a module subscribe to? In Nagios 2.3.1, the version of Nagios I'm using as I write this, there are 31 total call-back types, although some of them are reserved for future use. These constants are defined in nebcallbacks.h, in the "includes" directory of the tarball. Listing 1, on the next page, contains some of the call-back type constants.

The available callbacks cover every type of event that can happen in Nagios. An NEB module may register to receive information about any or all of these event types. Once it initializes all the modules, the Event Broker waits for events matching the type subscribed to by the module and, upon receiving one, gives the module information about the event, as well as a handle to the relevant data structures.

For example, if the module registered for EXTERNAL_COMMAND_DATA, the Event Broker would notify it every time an external command was inserted into the command file. A handle to a struct that defined the command would accompany the notification. The module could inspect and optionally change any of the information in the command struct or even delete it altogether. But enough talk about the architecture; the best way to learn about the NEB is to see how these modules work in practice.

```
NEBCALLBACK_FLAPPING_DATA
NEBCALLBACK_PROGRAM_STATUS_DATA
NEBCALLBACK_HOST_STATUS_DATA
NEBCALLBACK_PROCESS_DATA
NEBCALLBACK_TIMED_EVENT_DATA
NEBCALLBACK_LOG_DATA
NEBCALLBACK_SYSTEM_COMMAND_DATA
NEBCALLBACK_EVENT_HANDLER_DATA
NEBCALLBACK_NOTIFICATION_DATA
NEBCALLBACK_SERVICE_CHECK_DATA
NEBCALLBACK_HOST_CHECK_DATA
NEBCALLBACK_COMMENT_DATA
NEBCALLBACK_SERVICE_STATUS_DATA
NEBCALLBACK_ADAPTIVE_PROGRAM_DATA
NEBCALLBACK_ADAPTIVE_HOST_DATA
NEBCALLBACK_ADAPTIVE_SERVICE_DATA
NEBCALLBACK_EXTERNAL_COMMAND_DATA
NEBCALLBACK_CONTACT_NOTIFICATION_DATA
NEBCALLBACK_ACKNOWLEDGEMENT_DATA
NEBCALLBACK_STATE_CHANGE_DATA
```

LISTING 1: SOME NEB CALLBACK TYPES

Nagfs is a filesystem interface that represents the state of a running Nagios daemon. Each host monitored by Nagios has a directory in the filesystem, and each service on that host has a file. The contents of the file match the Nagios service state for that service. For example, if the httpd service on box1 was down, then /usr/share/nagios/status/local/box1/httpd would contain a '2'. Most people scrape HTML from the Web interface to get this kind of info, so you can imagine how handy it is to just be able to do `grep -rl 2 /usr/share/nagios/status/local` to find all the services in a critical state instead. Nagfs keeps the filesystem up to date by subscribing to state change events. Every time a service changes state, the event broker tells nagfs, and nagfs updates the filesystem immediately. No waiting for an external event_handler to fire; if Nagios knows about it, so does nagfs.

The complete source code for nagfs is a bit long to print here. If you'd like to compile it yourself, grab a copy of the source off my blog (www.skeptech.org/?p=35), from http://www.usenix.org/publications/login/2008-10/nagfs.c, or from NagiosExchange along with the Nagios source code from nagios.org, and follow the instructions therein. What we can do is examine some key portions of the source. Let's start with the function declarations:

```
/* Nagfs Functions */
void nagfs_reminder_message(char *);
int nagfs_handle_data(int,void *);
int nagfs_write_service_status(char *, char *, int, int);
int nagfs_write_host_status(char *, int, int);
int nagfs_check_for_softfiles(char *);
```

An event broker module is only required to have two functions, `nebmodule_init` and `nebmodule_deinit`. The function `init` gets called when our module is first initialized, and `deinit` gets called when Nagios quits and we get unloaded. The functions I've declared above are all optional, and I mostly declare them up front so that the program follows a more linear progression and is therefore easier to write about. The basic strategy is that our `init` function will register for event callbacks and will call `nagfs_handle_data` to handle the data we receive from the broker; `nagfs_handle_data` will in turn call the other functions as needed to update the filesystem. For example, it will call write_service_status when a service status change has occurred and it needs to update the file that corresponds to the service in the filesystem.

Next is our init function line:

```
int nebmodule_init(int flags, char *args, nebmodule *handle){
```

It takes three arguments. The first argument is meant to give you some context about how the module is currently being initialized. I don't use it in nagfs. The second argument is a string pointer called args. You may pass arguments to the module using ones found after the module name in the broker_module directive in your nagios.cfg. If you do so, they will be available in this args string. The third argument is a handle to the struct that defines our module. We can use this to refer to ourselves, if, for example, we call a function that requires a pointer back to us. Actually, this happens right off the bat when we register with the broker to get some data:

```
neb_register_callback(NEBCALLBACK_SERVICE_STATUS_DATA,nagfs
_module_handle,0,nagfs_handle_data);
neb_register_callback(NEBCALLBACK_HOST_STATUS_DATA,nagfs
_module_handle,0,nagfs_handle_data);
```

Ask the broker for events with the neb_register_callback function, which takes four arguments. The first is a constant that defines what type of events we're interested in. These are the same constants as in Listing 1. The second is our handle, so that the broker can find out what it needs to find out about us. The third is a priority number. In general, when more than one module registers for the same type of event, they are executed in the order they are loaded by the broker on startup. You can override this behavior by specifying a priority number. The last argument is a function pointer to our data handler function. Our data handler will be the function that actually gets the event struct and does stuff with it.

There's not much more interesting here, so let's skip down to the declaration line for the handler function:

```
int nagfs_handle_data(int event_type, void *data){
```

The data handler must return an exit code in the form of an int and accept two arguments. The first of these is a constant specifying the event type: yes, once again, one of the constants specified in Listing 1. The second is a void pointer, which I'll get to in a moment.

So why would our event handler need to be passed the event type? The event handler function should be able to infer the event type, by virtue of the fact that we specified it when we defined the handler. But notice that we actually use the same handler function for both event types we are registering for. Thus, when the broker spawns the data handler, it passes the event type along, just in case the handler has more than one job (as ours does).

The void pointer is a data struct that is passed from the event broker. It's our magic smoke—the instantiation of the data we're actually looking for. It will be a different type of struct depending on the type of event data the broker passes us. It's up to you to typedef the struct into the correct type. You can find the various types in the broker.c file in the Nagios tarball. Our event handler uses a switch-case on the value of the constant to decide what kind of event we're dealing with. Then it typedefs the data struct accordingly, as you can see here:

```
switch(event_type){

    case NEBCALLBACK_SERVICE_STATUS_DATA:
        ssdata=(nebstruct_service_status_data *)data;
```

In this case, we've gotten service status data, so we've typedef'd the struct into type `service_status_data`. Now I can dereference information about the service from the struct. The broker.c file is also handy for finding out what sorts of data we can dereference from the structs we get from the broker: stuff like `svc->host_name`, which I pass to the write functions I found out about by reading the structs in broker.c.

The rest of the program is pretty self-explanatory. If we get service data, we pass it to the `nagfs_write_service_status` function. Host state data goes to the `nagfs_write_host_status` function. These functions primarily deal with directory and file access and error detection (the "boring stuff").

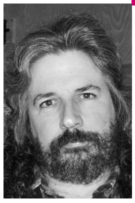
There are a slew of changes in 3.0 that make the Event Broker even more powerful. My personal favorite is the addition of custom external commands. Basically, these are commands that you make up and pass in to the external command file. They are not processed by Nagios (obviously, since Nagios won't know what you're talking about), but they can be detected and parsed by an event broker module that knows about them, so they're a great way to get external (non-Nagios) data to your module.

The moral of the story is that you should totally write an event broker module. They're fun to write (more fun than writing event handlers in Perl anyway, heh), they'll help other people out (real people, who haven't made exactly the same architectural assumptions you have), and they're a great excuse to dig around in the Nagios source, which I promise you, is some of the most elegant, well-engineered C that you'll come across in a project of this size.

Take it easy.

ROBERT G. FERRELL

/dev/random



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

rgferrell@gmail.com

WHEN I WAS BUT A WEE LAD DURING the cosmic groovy '60s, there were relatively few tomes flitting about the corner bookseller that could be considered "self-help" books: Dale Carnegie's unforgettable literary blockbuster *How to Win Friends and Influence People,* already a venerable classic by this time, leaps nimbly to mind. The rebirth of rampant materialism in the '80s provided an ideal breeding ground for a vast weed patch of personal improvement guides, and the "me first" '90s were no less fecund in this respect. I didn't mention the '70s because I was too busy coping with hormones, college, and the horror that was disco to pay much attention to what people were reading.

Even though there's (thankfully) not a lot left of the Decade with No Good Name (the oughts? the zeros?), the self-help pathogen hasn't really abated much, although it has embraced additional infection vectors such as blogs, podcasts, and Webzines. The bacillus has mutated over the years in response to economic conditions, however, and now primarily targets victims who are unemployed or just desperately want to be differently employed. There have been many millions of words written on how to find, curry favor with, retain, and slough off unwanted jobs, employers, and employees.

The only libations I have quaffed from this virtually bottomless well of wisdom concern the interview process. I happen to be rather well versed in the fine art of screwing up interviews, and a fair portion of that acumen is the direct result of having taken some of this pabulum too seriously.

Fortunately, I don't have to worry about readers taking what I say here seriously; anyone who does probably has much greater issues with life than those that might arise after putting my ersatz advice into practice. To encourage more people to seek jobs in the UNIX system administration field, then, and thereby increase my potential reading audience, I have painstakingly prepared this list of potential interview questions, based on my own experiences, stuff I stole from the Intertubes, and single-malt Islay–induced hallucinatory meditations. Study well and be prepared for the inexplicable, grasshopper.

1. If you could choose to install any operating system ever made by Microsoft, why would you?
2. Apple's OS X is based on which of the following?
   - a. *The Cat in the Hat Comes Back*
   - b. BSD
   - c. LSD
   - d. Iron Maiden's "2 Minutes to Midnight"
   - e. *Nosferatu: Director's Cut*
3. What is a "shell" and why is there always one in your omelette?
4. The power goes out unexpectedly and you have thirty seconds of reserve from the UPS you bought on eBay to shut down your UNIX machine gracefully. Which command would be your best choice?
   - a. shutdown -y now
   - b. shutdown now | imeanit
   - c. perl -p -e "exec(ps | awk '{print $1}' | grep -v PID | kill -9;
   - d. boot | halt
5. One of your users sends you an email request to increase her disk quota. What should you do?
   - a. Ignore her and go back to playing Adventure.
   - b. Ignore her and go back to playing Nethack.
   - c. Pipe her outgoing SMTP traffic to text2voice over the intercom.
   - d. SMS a meatlover's pizza and eat it while you dismantle her workstation.
   - e. Tell her to fill out a user account modification request in triplicate and email it.
6. Your boss tells you to do something you don't think is right. How do you respond?
   - a. Pretend you don't speak the same language.
   - b. Suddenly run out of the room holding your stomach.
   - c. Tell him you're on break and offer him a beer from behind the server rack.
   - d. Hold your breath until you pass out.
   - e. Ignore him and go back to sending out resumes.
7. The VP of operations calls you into her office to set up iTunes on her new PC. Which course of action should you take?
   - a. Offer to enable IPv6 on her digital picture frame while you're there.
   - b. Subscribe her to alt.binaries.pictures.erotica.walruses and wallpaper it.
   - c. Set the screensaver to activate after three seconds of idle time (with a password).
   - d. Explain to her that iTunes songs can only be downloaded on a Mac.

8. The Human Resources Director complains that his computer is running slow. You examine it and discover that a spyware application seems to be using up all the CPU cycles. Choose the best response:

    a. "I think you're hosed, dude. Better buy a typewriter."

    b. "Your operating system has rabies. Hope you had all your shots."

    c. "What time does that hot HR receptionist get off?"

    d. "See those little black lump things on the motherboard? They're PC barnacles."

    e. "Does the term 'autoimmunity' mean anything to you?"

9. Why are there "man" pages but no "woman" pages? Answer in pantomime.

10. RC scripts are used during what process?

    a. Flying your 1:25 scale V-22 in the park

    b. Burping the Poincaré conjecture

    c. Starting services at boot time

    d. Convincing the demons of darkness to pass over your simple hovel

And, last but not least, the question you're most likely to hear on any given UNIX sysadmin interview:

How did you learn UNIX?

IRTFMP.

NICK STOUGHTON

update on standards: the USENIX Standards Project

Nick is the USENIX Standards Liaison, representing USENIX in the POSIX and Programming Language Standards Committees of ISO and ANSI. When he is not busy with that, he is a consultant, with over 25 years of experience in developing portable systems and applications, as well as conformance testing.

nick@usenix.org

USENIX HAS BEEN FUNDING ACTIVITIES in standards for around 20 years. The organization has been involved with POSIX since its inception, as well as with the C and C++ programming languages, the Single UNIX Specification, the Linux Standard Base, and several other projects.

Over those 20-odd years, USENIX has gained a reputation and an esteemed position in the standards community. We are held up as the primary champion of free and open source software within the particular niche of programming language and operating system standardization.

USENIX holds numerous senior positions on the various standards committees: secretary to the IEEE Portable Applications Standards Committee, secretary to the Austin Group, International Representative for the USA on all POSIX and Linux matters, co-editor of the C standard, editor of a technical report for the C committee, editor of the Linux Standard Base, inter-working group liaison for all programming languages and POSIX, and chair of the LSB specification authority, to name just some of them. And let’s be shameless: although USENIX holds the positions, since it funds the activities, the person actually doing all of these jobs is me!

The annual expenditure for this work represents a sizable proportion of the organization’s Good Works budget (about 34% of the total, a sizable proportion of which goes to international travel expenses). Quite reasonably, the board periodically reviews where it is spending the organization’s money, and at a recent Open Board Meeting those present were asked if USENIX’s support of standards activities was really benefiting the membership.

I believe it is true that the standards that we are involved in affect in some way or other every single member of the organization, every single day they work. Maybe you never notice, but every keystroke you type has been touched by code written in C. If you are a UNIX user . . . well, the Single UNIX Specification is one of those standards. It is a superset of the POSIX standard. The fact that the same command does the same thing across all versions of Linux, *BSD, HP/UX, and several other systems is because there’s a standard. Maybe you use a GUI . . . probably some C++ there. I regularly receive questions and comments from members of various open source communities about POSIX and

C interpretations that lead me to believe that many of you are not unconscious of the standards you use.

Standards are what make open source software successful. They allow that software to be ported from environment to environment with ease, freeing us from having to reinvent the wheel every time we work on a new project. The Open Source Initiative (OSI) (www.opensource.org) has an "Open Standards Requirement" (OSR). I'd like to quote some parts of their rationale for the OSR:

> If interoperability is a grand goal as it relates to software, then standards are the critical tools for achieving this goal. . . . At this point in time, it has become largely intuitive across the industry and among users that broad and widely accepted standards are a Good Thing. . . . The purpose of an open standard is to increase the market for a technology by enabling potential consumers or suppliers of that technology to invest in it without having to either pay monopoly rent or fear litigation on trade secret, copyright, patent, or trademark causes of action. No standard can properly be described as "open" except to the extent it achieves these goals.
>
> The industry has learned by experience that the only software-related standards to fully achieve these goals are those which not only permit but encourage open-source implementations. Open-source implementations are a quality and honesty check for any open standard that might be implemented in software, whether an application programming interface, a hardware interface, a file format, a communication protocol, a specification of user interactions, or any other form of data interchange and program control.

The standards that USENIX is currently helping to develop and maintain are all Open Standards by the definitions used by the OSI.

Let's consider some current projects in which USENIX is involved. POSIX has just completed a revision (only the third since 1988) that has added a number of new APIs coming from the open source world (and, in particular, from glibc). Many hundreds of other issues were addressed at the same time, and several previously optional features have now been mandated. (An aside: Optional behavior is a real nuisance to the end-application developer. If you cannot rely on a particular interface being available, then you have to code around the possibility that the interface is absent, which bloats your code and makes maintenance harder. So, getting rid of options is definitely a Good Thing in my mind.) The POSIX working group (known as the Austin Group, after their first meeting in that city) is staunchly opposed to invention. Everything that goes into the standard must be based on widespread existing practice. Most of the issues that arise are because there is widespread existing practice that implements an API in a different way from that described in the standard. Much of the new revision has been aimed at removing some of the differences between Linux and UNIX.

The C committee is preparing for its second revision since 1989. Like the POSIX committee, there is a very strong resistance to invention by the majority of members. This revision will likely include features such as attributes for functions, variables, and possibly types, better support for concurrency (probably including a thread API that is at worst a thin layer over POSIX pthreads), and other features already supported by most if not all C compilers in one form or another.

The C++ group is struggling to finish its revision, which will make radical changes to the language. Unlike C and POSIX, in C++ invention is not feared. USENIX has been one of the leading voices in this effort, trying to ensure implementability and usefulness over cool and sexy with regard to new features! Again, like C, concurrency support is a major driver.

As a spin-off of the C++ concurrency work, and trying to fill in where the necessarily platform-neutral aspects of the programming language itself fall short, a new POSIX/C++ binding working group has just been formed. This group will be producing an IEEE standard that encapsulates POSIX threads, file system, and networking (to name just some of the larger features) in C++ objects.

I hope this report helps you to answer for yourself the “Is this a worthwhile effort?” question raised earlier this year. I’d be interested to hear your opinion.

book reviews

ELIZABETH ZWICKY, WITH BRAD KNOWLES, SAM F. STOVER, AND RIK FARROW

BETTER: A SURGEON'S NOTES ON PERFORMANCE

Atul Gawande

Picador, 2007. 257 pages.
ISBN 978-0-312-42765-8

This is obviously not even vaguely a book about system administration. I didn't pick it up intending to review it. I'm not quite sure why I did pick it up. However, I ended up mulling over the similarities between the intractable medical problems it describes and system administration.

This book is about how medicine improves, and it points out that the main problems are not technical, but human: How do you get people to do things that they should do to prevent long-term problems but that give them nothing but annoyance in the short term? This is clearly just as applicable to getting your users to comply with security requirements as it is to getting your doctors to wash their hands to prevent spreading infection.

I also found a useful moral in the tale of Dr. Semmelweiss, who famously figured out that doctors were spreading infection between women giving birth, and then didn't manage to get them to stop. This turns out not to be a simple story of an unheard genius, but the story of somebody who was technically right, but so annoying and unable to explain himself that nobody listened. We've all worked with that guy, right?

If you're looking for some fascinating and useful concepts to wrap your mind around, and are willing to stretch a little to apply them, I think you'll find a good deal of usefulness here.

GEEKONOMICS: THE REAL COST OF INSECURE SOFTWARE

David Rice

Addison-Wesley, 2008. 339 pages.
ISBN 978-0-321-47789-7

Software runs the world and it doesn't work reliably. This is clearly a bad thing. So why doesn't somebody fix it?

This book lays out the forces that keep software unreliable, explains why open source software isn't the cure, and suggests some solutions. There's an interesting discussion of licensing for software engineers, which system administrators should find eerily familiar.

I am reasonably convinced that liability for software manufacturers would improve the world. Every time I think software licensing can't get any more absurd, I discover that I am not yet adequately cynical. (Imagine my surprise when I bought a cookbook with an included CD and discovered that the software license for the CD was printed on the back side of the book's paper slipcover, where there is normally nothing at all.) I also appreciated the discussion of the forces that drive open source toward bloat.

Although I agree with the author, I think he overstates the case in several places. Software isn't the only source of unexpected interactions, and a serious case can be made that better error tolerance is important entirely separate from software problems.

YOUR BRAIN: THE MISSING MANUAL

Matthew MacDonald

O'Reilly, 2008. 247 pages.
ISBN 978-0-596-51778-6

Popular books on the brain are often minefields of attractive but inaccurate information. This one manages to avoid most of the hype and easy faulty generalizations while providing easy to read and digest information about the brain. It has useful tricks without the breathless hype of many popular books.

In particular, it has a nice clear explanation of what is known about gender differences and the brain (none of which is really all that exciting). This is a nice antidote to a lot of the nuttier stuff going around. Unfortunately, there aren't any references, so you're pretty much stuck taking the author's word for it. Comparing what this says to sources I trust, that works out OK. But if you don't happen to know any neuroscientists to ask, you would have a hard time figuring out whether this was in fact more accurate than some very popular books with numerous irrelevant footnotes.

EATING THE IT ELEPHANT: MOVING FROM GREENFIELD DEVELOPMENT TO BROWNFIELD

Richard Hopkins and Kevin Jenkins

IBM Press, 2008. 213 pages.
ISBN: 978-0-13-713012-2

On the face of it, this is the single most relevant book I'm reviewing this issue. It's about building big projects inside an existing IT organization, where you are having to interface with all the existing, crufty systems. The analogy with building on contaminated ground will seem quite apt to anybody who's tried to add anything to a mature IT infrastructure.

Clearly, the authors have worked on many projects like this and learned painfully. They have a grand theory of how to deal with the situation, which they lay out with gripping metaphors. They compare it to other development methodologies and provide some implementation advice.

However, their theory involves a grand unifying program, and implementing that program is going to be a major stumbling block for anybody who wishes to use the process. Furthermore, the grand unifying program needs to contain a knowledge representation of all the parts of the IT infrastructure. They're quite correct that having a working knowledge representation is extremely powerful and enables all sorts of fun stuff, but they imply that it's pretty straightforward for a business analyst to go from an existing program to an appropriate representation of the objects, relationships, and constraints (e.g., every account has one and only one username, every file is owned by one account, and so on and so forth). It's not at all a straightforward matter of discovery and analysis.

This book has interesting ideas for people doing large development projects that interoperate with existing systems. For most sites it's going to be much the same as better—not an immediate resource you can implement, but an intriguing starting place for adaptation.

THE BOOK OF IMAP: BUILDING A MAIL SERVER WITH COURIER AND CYRUS

Peer Heinlein and Peer Hartleben

No Starch Press, 2008. 368 pages.
ISBN 978-1593271770

REVIEWED BY BRAD KNOWLES

For us mail server administrators, there aren't many books in print that discuss installing, configuring, and operating IMAP mail systems. If you do a search at your local library or on bookseller Web sites such as Amazon, you will discover that there's only one other book available that is devoted to the subject—*Managing IMAP* by Dianna and Kevin Mullet—published way back in the dark ages of 2000 and now very long in the tooth. There are a few other books that might give us a single chapter, and at least one or two other books on programming as it relates to Internet email, but that's about it. This is the gap that *The Book of IMAP* is intended to fill, specifically for Courier-IMAP and Cyrus.

The book is separated into three standalone parts. Part One, "How to Set Up and Maintain IMAP Servers," discusses topics that are generally applicable to most IMAP servers. Part Two is devoted to Courier-IMAP, and Part Three is about Cyrus. There are also three appendix chapters providing an IMAP command reference, a POP3 command reference, and a guide to installing from source code as opposed to binary packages. The last 20 or so pages are devoted to a fairly extensive index.

The authors clearly work pretty much exclusively with Linux. Everywhere that they talk about differences among specific platforms, they discuss choices such as SuSE, Red Hat, and Debian/Ubuntu, and that's about it. If you can look past their obviously Linux-oriented nature, you should be fine.

One thing that really annoyed me about this book is the occasional misspellings and improper grammar. It is clear that English is not the first language of the authors, although it is likewise clear that the authors care a great deal about proper word usage and sentence structure. This is what makes it doubly annoying when you run across phrases such as "However, these combinations cannot be combined with additional permissions...." There are also typesetting issues, with paths to files broken in the middle of a directory name, when they should be broken at directory separators (e.g., "/"), or where options to "./configure" should not be broken across two lines at all.

Chapter 1 starts with a review of protocols and terms, and Chapter 2 is a step-by-step review of the POP3 and IMAP protocols. Chapter 3 launches into the much weightier topic of load distribution and reliability, starting with load-balancing technologies (including DNS round-robin, round-robin via iptables, and Linux Virtual Server), but where is the discussion of load-balancing switches?

Chapter 3 also covers the subject of IMAP proxies and mentions one reason why you might want to run IMAP proxies even if you have only one IMAP server—certain Webmail IMAP clients are very poorly behaved and open a separate IMAP connection for virtually every single user-visible element on the screen, and caching IMAP proxies help offload much of that work from the back-end IMAP server. However, this ignores the fact that certain other IMAP clients are also equally poorly behaved, a fact we know all too well here at my current employer. Therefore, anyone anywhere who is running an IMAP server of any size may potentially benefit from having a system with a caching IMAP proxy daemon.

Chapter 4 takes us into the subject of choosing a filesystem and filesystem tuning for maximum performance, as well as selecting benchmarking and stress-testing tools to help you during this process. You guessed it; they only tested ext3fs, ReiserFS, and XFS, although they do actually mention that OpenSolaris uses something called ZFS. Fortunately, they at least show the difference that high-performance disk drives can make, and they talk about important things such as RAID and NFS, highlighting various options you may want to look at to help improve your performance.

Chapter 5 is about potentially useful Webmail clients, including Squirrelmail and Horde/IMP, and it goes into more detail about some of the problems that such clients can cause and why you might want to use a caching IMAP proxy to help solve those. The subject of Chapter 6 is migrating IMAP servers, using tools such as imapsync, pop2imap, imap_migrate, imapcopy, and imap_tools. This chapter also talks about converting mailbox formats, changing IMAP folder names, and determining cleartext passwords—all things that you might need to worry about if you're migrating from one type of IMAP server to another.

With Chapter 7, we get into the second section of the book, where Peer Heinlein discusses the Courier-IMAP server itself. He starts by covering the basics of binary package installation, what gets put where in the filesystem, initial startup, integrating Courier with MTAs (postfix, qmail, and Exim), optimizing the configuration, and what configuration parameters do and where they go. Chapter 8 is about Maildir as an email storage format, how the IMAP namespace impacts that, filenames of email messages, and what flags can be attached to the files.

Chapter 9 focuses on user data and authentication and the myriad different ways that can be achieved, whether through internal-only methods or tying into external systems such as MySQL, PostgreSQL, or LDAP, and whether that's done directly or via PAM, etc. In this long chapter the authors try to make sense of all the hash and organize the information in a reasonable fashion. Courier-IMAP does not support the full SASL standard, but it does support enough of it that you can implement a complete IMAP mail server system. The advantage here is that the authors have left out much of the complexity that Cyrus includes with the SASL Reference Implementation, which has been a large part of why so many mail admins go screaming into the night when they hear the term.

Chapter 10 is for Courier administrators and discusses setting up various different types of shared folders, setting up quotas, using Courier to build an IMAP proxy, configuring systems for "push" IMAP email, and sending emails via IMAP.

Chapter 11 starts the third section of the book, where Peer Hartleben talks about Cyrus. He begins with structure and basic configuration, including installation (binary packages again), optional additional tools that are intended to make it much easier to administrate Cyrus via a WebUI, the Cyrus hierarchy in the filesystem and permissions, features and functions, and authentication (SASL), and then provides a "Quickstart Guide," which includes integration with postix.

Chapter 12 takes us into the Cyrus configuration file, and although it is short, it covers many different configuration options that could have a huge impact on the performance of your server. Chapter 13, on authentication and safeguards, starts with SSL and TLS encryption, SSL certificates, and SASL. For SASL, this chapter gets into detail about how various different modules interact with user authentication schemes, including /etc/passwd logins, Berkeley DB files, and integration with external database systems such as MySQL or LDAP and through intermediary systems such as PAM, and with Kerberos.

Chapter 14 covers advanced Cyrus configuration, including quotas, shared folders and ACLs, virtual domains, sorting email messages into subdirectories and the use of hashed directory schemes for enhanced performance, setting up multiple different partitions for users, the Sieve server-side message filtering language for IMAP servers, integrating Cyrus with other MTAs, backing up and restoring user mailbox data, and performance tuning.

Chapter 15 delves into internal structure and modules, and although this probably isn't strictly necessary, you learn a lot that may turn out to be extremely useful regarding which internal modules do what, what tools are available to do analysis, maintenance, and repairs, the function of a multitude of other lesser-known Cyrus tools, and using the cyradm administration tool.

Chapter 16 details Cyrus at the filesystem level, focusing on the email directory and the administration directory, and we see the function for each of the main subdirectories in these trees. Finally Chapter 17 is all about using Cyrus in a cluster, starting with the cyrus aggregator (a.k.a. "murder," as in "a murder of crows"), the front-end servers, the back-end servers, and the mupdate server, and ending with a brief discussion of replication.

This book is not perfect, but it's much better than what we've had to date. On the whole, I agreed with most of the things the authors wrote, and where I disagreed with them it was more a matter of feeling that they didn't go far enough on a given topic, as opposed to being flat-out wrong. At least the authors introduce the reader to a variety of topics in the field that you just don't find in any other book having to do with mail system administration, and they get the reader started down a good path.

However, this is not a book that does hand-holding for novices. It will be useful to experienced mail system administrators, but if you're not already in this business and you don't already know something of the subject, then you're going to have a steeper learning curve.

At the end of the day, my benchmark is whether or not I would buy the book for myself, if I wasn't given a free copy to review. My answer to that question is most definitely "Yes!"

I've had the opportunity over the years to be a principal person doing the architecture and design of two fairly large-scale email systems, one using commercial products based on Cyrus for a large national ISP and one using purely open-source software and built around Courier-IMAP for a medium-to-large multinational corporation. However, I feel that this book has definitely helped me achieve a better and deeper understanding of these packages.

I will be taking my heavily marked up copy and sharing it with my colleagues with whom I help administer the primary campus mail system for ~50,000 students and ~20,000 faculty and staff here at the University of Texas at Austin. I'm sure this will have an impact on our day-to-day administration of our existing Courier-IMAP based mail system, as well as influencing our future choices for whatever our next-generation campus-wide IMAP server will look like.

And yes, I'm also going to contact the authors and see whether they're interested in getting some help for the second edition.

GOOGLE HACKING FOR PENETRATION TESTERS, VOLUME 2

Johnny Long

Syngress, 2007. 448 pp.
ISBN 978-1-59749-176-1

REVIEWED BY SAM F. STOVER

I don't know what it is about Johnny Long's books, but I just love 'em, and this one is no exception. Good content, good style, and good humor: what more could I ask for? Since I hadn't read Volume 1 (released in 2006), I wasn't sure what to expect, but I was definitely pleased with the end product. Also, I don't want to detract from the other authors; it's apparent that this was a group effort and it was well done.

The first chapter starts off with Google basics, followed by Advanced Operators in Chapter 2. Much of these two chapters could be familiar to the tech-savvy, since we all use Google on a daily basis anyway, right? Chapter 3 is where things start to get interesting, and it just goes on from there. Not that each chapter is that much better than the previous, but each is cool in its own way. Chapter 3 starts diving into basic Google-hacking methods, with some solid guesses on what is actually happening on the back end. Chapter 4 describes how to conduct searches to find data inside of different types of documents, such as databases, log files, and config files.

Chapter 5 was probably my favorite chapter. As a whole it addresses data mining using Google search terms, but it starts with email, phone number, and domain searches—exactly the kind of thing you'd want to be able to do if you find yourself pen-testing a company and need to track down valuable information about a particular individual during said engagement. The obvious next step to this is automation, and although Google does frown slightly on some types of automation, Chapter 5 walks you around the edges and lets you put your computer to work without angering the gods. You'll see a couple of different scrapers and also be introduced to Evolution (now called Maltego), an open source "intelligence"-gathering application.

Chapters 6 and 7 deal with googling for exploits and "simple security" searches, Chapter 8 shows some techniques for finding different kinds of Web servers, portals, and also networking devices, and Chapter 9 focuses on searching for usernames and passwords. Chapter 10 goes back to the automation drawing board and discusses the AJAX Search API provided by Google. The book rounds out with about 50 pages of "Google Hacking Showcase" items in Chapter 11 and ways to protect your assets from Google hackers in Chapter 12.

All in all, this is a very solid book. There were a few more grammatical, spelling, and editorial errors than I'd like to see, but the content was good enough to distract me from the errors and omissions. I've been spending a fair amount of time lately doing engagements in which pen-testing skills and tools are needed, and I think this book, even as big as it is, will be a permanent part of the repertoire.

CRIMEWARE

Markus Jakobsson and Zulfikar Ramzan

Addison-Wesley Professional, 608 pp.
ISBN 978-0-321-50195-0

REVIEWED BY SAM F. STOVER

My biggest gripe with this book, and it's a big one, is the word "crimeware." I just can't buy into that term, and that makes this a hard book to read at times. I'm not saying that it isn't a valid or descriptive term, because it's both, but it just doesn't read or sound right. That said, I think a lot of the content of the book is really spot on, which is why I resolved to get over my anti-crimeware attitude, and I encourage you to do the same if you find that you're turned off by the title.

Chapter 1 provides an introduction to the term "crimeware" and goes into a fair bit of detail on how different types of malicious software can be grouped into what the authors have deemed crimeware, namely, malware designed for the express intent of committing criminal acts. This chapter touches on a lot of different topics and serves to show how bots, trojans, rootkits, transaction generators, proxy attacks, and dns cache poisoning can all be labeled as crimeware. There's a fair bit of technical detail present in this chapter, with indications of more in the following chapters.

The remaining chapters don't really follow any recognizable pattern or hierarchy. They were all written by different authors and could stand on their own. (In fact, I believe some of the chapters were either papers or featured articles in other publications.) I'm going to focus on the chapters that really appealed to me, but on the off-chance that what appealed to me might not be what you are looking for, I definitely recommend checking this book out and seeing whether the topics that interest you are addressed.

Even though I'm not a software engineer by any stretch, I really enjoyed Chapter 2 by Gary McGraw, which deals with "A Taxonomy of Coding Errors." He has built a whole nomenclature around coding errors and how they contribute to malware infection and propagation. Good stuff. Another chapter I really liked was Chapter 9 on "Virtual Worlds and Fraud." I see this as a ripe market for malicious entrepreneurs, and despite the brevity of this chapter, it was pretty engaging. Another good read was Chapter 11, "Online Advertising Fraud," which dove into several different mechanisms developed by bad guys to make money from folks like Google (and, incidentally, the Google Ad Traffic Quality Team co-wrote this chapter). The last chapter that really grabbed my attention was "Crimeware Business Models," which discusses how the bad guys are making money from all this criminally focused malware.

I think this book takes a reasonably good look at a very diverse and complex topic. There were definitely times when I wished there was more detail, but I guess that's saying that the topics were interesting enough that I wanted more. As with any book that collects from a large author pool, there was a little bit of overlap between certain chapters, but nothing that I couldn't overlook. The topics are interesting, the spelling and grammar are top-notch, and you can easily bounce around the book as your fancy takes you. It gets high marks from me, and I'd definitely be interested if a second edition were to come out.

RUNNING XEN: A HANDS-ON GUIDE TO THE ART OF VIRTUALIZATION

Jeanna N. Matthews, Eli M. Dow, Todd Deshane, Wenjin Hu, Jeremy Bongio, Patrick F. Wilbur, and Brendan Johnson

Prentice Hall, 2008. 586 pages.
ISBN 978-0132349666

REVIEWED BY RIK FARROW

If you read the article in this issue about virtualization in Solaris, you will have a good feeling for the depth of information found in this book, written by some of the same authors. When I first encountered Xen, I installed the right kernel, ran preconfigured guest images, and things just worked. But as soon as I needed to go beyond the basics, I discovered that running Xen is a complex topic. And that is where this book comes in.

The authors start with chapters on the basics of virtualization and the use of a Xen LiveCD, and they quickly move on to configuration options for xend, the interface to the hypervisor that runs in Domain 0. The book continues to dive deeply into setting up and running Xen, from prebuilt guest images to setting up devices that can be accessed only by a guest domain. I particularly appreciated the chapter on Xen networking, as the authors do a good job at explaining the differences among bridging, routing, and NAT-based networking, as well as about having completely virtualized networking among guests.

There is a lot of information in this book that is hard to find online, and it is also clear that the Clarkson University team that wrote this book is intimately familiar with Xen. I found much to like about this book. As an editor, I also have some problems with this book, as there are numerous little editing mistakes—e.g., sentences repeated twice, things that are poorly explained, missing explanations such as how to use a preconfigured guest saved with an .xva file suffix) and other rough edges. It is as if the authors are so familiar with Xen that they sometimes fail to communicate with others who haven't been breathing and living Xen for the past several years. Still, I can recommend this book as a useful resource to anyone tasked with managing Xen systems.

THE HEAD TRIP: ADVENTURES ON THE WHEEL OF CONSCIOUSNESS

Jeff Warren

Random House, 2007. 390 pages.
ISBN 978-1-4000-6484-7

REVIEWED BY RIK FARROW

When I learned that Elizabeth was reviewing *Your Brain: The Missing Manual*, I immediately decided that we needed to contrast that book with another, less pretentiously titled book. Whereas MacDonald's book has no references, Warren's book has 30 pages of them, all neatly listed right before the index. And although Warren's book is also about the brain, its focus is completely different.

Jeff Warren is a freelance producer for the Canadian Broadcasting Company, as well as a freelance science writer. This combination of avocations leads to a delightfully yet rigorously written romp through what he terms "The Wheel of Consciousness." Warren starts with sleep, offering himself as an experimental subject to sleep researchers. He isn't just doing this for the purpose of writing this book, but because he is genuinely curious about his own self. Besides learning the difference between hypnogogic and hypnopompic dreaming (one is quite hallucinogenic in quality, whereas the other is familiar to anyone who uses a snooze alarm), I also learned about The Watch, a reverie-like state that only occurs when you routinely get a chance to be in bed over eight hours.

Warren also examines waking states, wanting to enhance his own ability to be in "the zone" as well as to achieve better focus through the use of biofeedback training. And then there is the lucid dreaming workshop, along with the use of a special device (a topic I don't want to spoil, as it makes for good reading).

Much less of a manual, and with much greater depth, *The Head Trip* teaches you a lot about yourself while never failing to entertain.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

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

DISCOUNTS on registration fees for all USENIX conferences.

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

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

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

USACO REPORT

Rob Kolstad, USACO Head Coach

USENIX is one of the principal sponsors of the USA Computing Olympiad, the USACO. USACO fosters computing for students before they enter universities, a function of ever-growing importance as the percentage of college students choosing computer science as a major hovers at its 20-year low of 2.2%.

Through a series of six contests, 14 USA students and a single international representative earned berths at the 2008 USAICO, the USA Invitational Computing Olympiad, a grueling seven-day on-site set of six competitions that culminates in the announcement of the traveling team that will represent the United States at the International Olympiad on Informatics (IOI). This year's USAICO took place on the grounds of the University of Wisconsin–Parkside south of Kenosha, Wisconsin, near the home of director Dr. Don Piele.

While the name USACO may suggest that the competitions are strictly for USA students, this is by no means true. USA students generally compose a quarter or fewer of the competitors. In this season's largest Internet-based competition (March 2008), 984 students vied for a gold medal. Of those, 304 (30.9%) were from China, and 134 (13.6%) were from the USA. Students from Belarus (58), Bulgaria (46), Romania (44), the country of Georgia (35), Indonesia (33), Poland (25), Iran (25), and India (25) rounded out the top 10 of the 61 countries represented at the contest. Contests are translated into a number of languages, including Chinese, German, Farsi, Georgian, Indonesian, Polish, Russian, Spanish, and Turkish.

The three-hour USACO competitions feature three divisions (Gold, Silver, and Bronze), which ask competitors to write algorithmic programs in C, C++, Pascal, and/or Java. The Bronze contests set simple tasks that involve sorting, array manipulation, string manipulation, and the like. Once a student has mastered "flood fill" (e.g., given a map of elevations, how big is a lake that includes square [33, 25]?), he or she moves to the Silver division. Solutions for Silver division tasks require algorithmic thinking and programming: Dijkstra's algorithm, graph manipulation, graphic algorithms, and challenging ad hoc problems that each have a unique algorithm for solution.

Silver problems often have time constraints that become quite challenging unless the proper programming technique or algorithm is chosen. Consider a task involving a triangle of numbers with five (later, N) levels:

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

The challenge is to maximize the sum of numbers chosen by starting at the top of the triangle and traveling down the triangle to the bottom row, moving slightly right or left each time you descend one row. In the sample above, the route from 7 to 3 to 8 (on the third row) to 7 to 5 produces the highest sum: 30. Almost any first-year computer programmer will sneer at such a problem and quickly—and correctly—aver that a recursive solution will be but a few lines long.

Of course, there's always a twist. In this case, the triangle has as many as 100 rows and the time limit is 1.0 seconds in a 2.4 GHz Pentium-based computer. The simple recursive solution requires 2^100 iterations—that's 1.27 x 10^30. Even a superfast computer that can perform 10^9 iterations per second would require 10^21 seconds—40 quadrillion years—a number that dramatically exceeds the one-second time limit. The trick is to solve the problem backwards, starting at the bottom row, enumerating the best set of N – 1 solutions, and working your way back up to the top. This easily implementable solution requires O(N^2) time and way fewer than 10,000 operations for the 100-row triangle.

Silver programmers who demonstrate the "dynamic programming" algorithm (named for a math technique, not for programming) move to the extremely challenging Gold level. Only 276 folks entered the Gold competition in the huge March contest. Only 25 (less than 10%) achieved 800 or more points out of 1000. Here's the easiest task from the March Gold division:

> At Bessie's recent birthday party, she received N (2 <= N <= 100,000; N%2 == 0) pearls, each painted one of C different colors (1 <= C <= N).
>
> Upon observing that the number of pearls N is always even, her creative juices flowed, and she decided to pair the pearls so that each pair of pearls has two different colors.
>
> Knowing that such a set of pairings is always possible for the supplied testcases, help Bessie perform such a pairing. If there are multiple ways of creating a pairing, any solution suffices.

Doesn't that seem easy? Give it a try!

The June USAICO camp this year assembled our best finishers in the half-dozen Internet contests held since November (an additional qualifying exam is held in October).

Camp attendees were partitioned into two sets: those vying to make the IOI team for the August trip to Egypt and those training for next year's team. Attendees hailed from states across the continent (along with our international representative, imported specially to keep our top competitors challenged). Thomas Jefferson High School for Science and Technology in Virginia supplied a large number of students, as did their rival, Montgomery Blair High School in Maryland. Attendees included: seniors David Benjamin from Indiana, Artur Dmowski from New York, Kevin Lee from New Jersey, Spencer Liang from California, Haitao Mao from Virginia, and Jacob Steinhardt from Virginia; juniors Shravas Rao from Ohio and Goran Zuzic from Croatia; sophomores Michael Cohen from Maryland, Brian Hamrick from Virginia, Jacob Hurwitz from Maryland, Neal Wu from Louisiana, and Scott Zimmermann from Maryland; and two younger students, freshman Wenyu Cao from New Jersey and talented seventh-grader Daniel Ziegler from California, who gave the high schoolers a run for their money. All the seniors are attending Harvard or MIT this fall.

Coaches included former IOI champions Brian Dean from Clemson University, Alex Schwendner from MIT, and Percy Liang from UC Berkeley, along with perennial administrators Don Piele and Rob Kolstad. Coaches supervised, taught, deconstructed problems and solutions, and created the three problems for each of the ten contests (different contests for different divisions) of the camp.

Almost every day of the USAICO competition and training camp opened

IOI representatives Brian Hamrick, David Benjamin, Neal Wu, and Jacob Steinhardt

with a three- or five-hour contest for the high-level combatants. After lunch, contest review commenced, during which unsolved tasks were vanquished. Afternoon activities included ultimate Frisbee, Frisbee golf, miniature golf, game programming (two different games this year, including one from IBM used at the international ACM contest), movie night, and swimming. Attendees are busy from breakfast at 8 a.m. until about 10 p.m.

When the dust settled, four IOI representatives were announced, two sophomores and two seniors: Brian Hamrick, David Benjamin, Neal Wu, and Jacob Steinhardt. The level of competition was so high that team membership was up for grabs until the final five-hour contest. These four elite students represented the USA at the 20th IOI in Mubarrak City, Egypt, August 16–23, 2008. Look for the report in the December issue of *;login:*.

You don't have to look far to hear moaning about the lack of performance by today's high school students (a complaint documented since the dawn of history). The students at the USAICO include math champions, physics aficionados, musicians (coach Percy Liang also continues to perform in the world of competitive piano), and science fair winners (Jacob Steinhardt was a Silver Medal winner at the prestigious 2007–08 Siemens Competition in Math, Science & Technology for his project entitled "Cayley graphs formed by conjugate generating sets of S_n"). Croatian competitor Goran Zuzic not only achieved the highest scores at our USAICO competition (of course he can't represent the USA at the IOI) but also won his country's math and physics competitions as a high school junior.

USACO competitors grow into technical community citizens of fine repute: for example, former competitor and coach Russ Cox just won his second Best Paper award at a USENIX conference. USACO appreciates the support of USENIX and continues to strive to build strong students and future technical leaders and contributors.

UPDATE ON SAGE

Jane-Ellen Long,
SAGE Programs Director

LISA '08, Nov. 9–14

It's LISA time again. Join us in San Diego, CA, November 9–14, 2008, at the 22nd Large Installation System Administration Conference. You'll find all the information and activities you've come to expect, and more. This year we're offering six days of focused training at a special rate. Choose either virtualization—and who doesn't need to know more about that, these days?—or Solaris training from the experts. Find out more at www.usenix.org/lisa08/.

SAGE Short Topics Booklets Get Longer

Two new titles, each of nearly 100 pages, have joined the ever-expanding collection of SAGE Short Topics in System Administration, carrying on the tradition of Information from the Source. The first, *LCFG: A Practical Tool for System Configuration,* was written by—who better?—Paul Anderson, author of the LCFG tool and also author of the SAGE booklet on *System Configuration. LCFG* tells you everything you need to know to determine whether this is the right tool for your site and, if so, how best to deploy it there.

Don't forget to compare Cfengine, laid out for you by Cfengine tool author Mark Burgess in the SAGE title *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine,* and the upcoming SAGE booklet on BCFG, written by, you guessed it, BCFG tool authors Narayan Desai and Cory Lueninghoener.

The second of the recent titles, *Deploying the VMware Infrastructure,* is a collaborative effort by John Arrasjid, Karthik Balachandran, and Daniel Conde of VMware and Gary Lamb and Steve Kaplan of INX. (What were we saying about the importance of understanding virtualization?)

Everyone can preview and order the booklets online at www.sage.org/pubs/short_topics.html. SAGE members can also read and download the full booklet PDFs.

SAGE Talk

Have you joined *;login:* columnist Peter Baer Galvin's Wiki on Solaris System Analysis 101 at wiki.sage.org/bin/view/Main/AllThingsSun? (What were we saying about Solaris training?)

Not into Solaris? The SAGE blog is back, with a series on open source enterprise monitoring by *;login:* author Matthew Sacks. Comments welcome!

Speaking of joining, visit us on the SAGE Facebook group: www.sage.org/facebook. For a more formal group, join the LinkedIn SAGE group at www.sage.org/linkedin.

If you're not subscribed to the sage-members mailing list, you should be. To find out why, search the archives at www.sage.org/lists/mailarchive.html.

Don't forget to troll around the SAGE Web site from time to time. Check out the Recommended Reading and the Toolbox, and see what else is new online and what SAGE groups meet near you. (And please let us know if your group's not listed.)

Jobs, Jobs, Jobs

In these uncertain times, the SAGE Jobs Board offers you new opportunities daily, as well as an ideal site for posting your own resume. Subscribe to sage-jobs-offered to learn about postings the moment they appear. See www.sage.org/lists/lists.html for details.

From You to SAGE

Remember, SAGE is not just for you, it's *by* you, the system administration community. Have you recently had to figure something out? Write a white paper and save your fellow sysadmins some pain. Read a *really* useful book? Let us add it to the Recommended Reading list. Other ideas? Contact suggestions@sage.org.

Save the Date!

OSDI 08

8TH USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION

December 8–10, 2008, San Diego, CA

The 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08) brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software.

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

Fourth Workshop on Hot Topics in System Dependability (HotDep '08),
December 7
http://www.usenix.org/hotdep08

First USENIX Workshop on the Analysis of System Logs (WASL '08),
December 7
http://www.usenix.org/wasl08

Workshop on Power Aware Computing and Systems (HotPower '08),
December 7
http://www.usenix.org/hotpower08

Workshop on Supporting Diversity in Systems Research (Diversity '08),
December 7
http://www.usenix.org/diversity08

First Workshop on I/O Virtualization (WIOV '08),
December 10–11
http://www.usenix.org/wiov08

Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML08),
December 11
http://www.usenix.org/sysml08

Sponsored by USENIX in cooperation with ACM SIGOPS

www.usenix.org/osdi08/lo

conference reports

THANKS TO OUR SUMMARIZERS

2008 USENIX Annual Technical Conference

Boston, MA
June 22–27, 2008

VIRTUALIZATION

Summarized by John Krautheim (kraut1@umbc.edu)

- ***Decoupling Dynamic Program Analysis from Execution in Virtual Environments***
  Jim Chow, Tal Garfinkel, and Peter M. Chen, VMware

  Awarded Best Paper!

Jim Chow described a novel method for software testing and debugging using a virtual machine (VM) as recording and replay device. The concept is not new, but the technique presented provides a new tool for the arsenal of software developers and testers.

Jim points out that one of the main reasons for developing such a tool is the lack of automated methods in software development. The team at VMware wanted to make Dynamic Program Analysis (DPA) more accessible. DPA is ability to take a running computer program, stop it, and inspect its state. This technique is very useful for the programmer; however, existing tools for DPA have a very high overhead from context swapping, instrumentation, and analysis, which results in a slowdown on the order of one hundred times. Therefore, the VMware team looked for a way to improve this analysis technique without the slowdown from overhead. The solution the team came up with was to decouple the analysis and execution by parallelizing the problem with virtual machines. This allows the target system to run freely in one VM while the analysis system records and regenerates events on a separate VM. The hypervisor is used to record all inputs to the VM under analysis and can start the analysis machine from the same state and replay instructions. The analysis system regenerates all the data needed, removing the overhead from the target system. Since the overhead of recording is very efficient with virtual machines, the target system can run at roughly native speed.

To demonstrate the technique, the team developed the Aftersight system. Aftersight is built on the VMware virtual machine monitor and thus it inherits many properties of VMs that can be leveraged to solve the problem. The Aftersight system provides isolation of the target and analysis system so that the analysis is self-contained and communication bottlenecks are eliminated. Through parallelism, the analysis and the target can run separately, on multiple cores if available. This allows analysis to go faster, and multiple analyses may be performed at the same time. An added side benefit of this parallel playback and recording is that ex-post-facto analysis can be performed on behavior not known at the time of recording, providing the ability to examine events not foreseen at execution time. The team has used Aftersight to debug VMware's own ESX Server, the Linux kernel, and the Putty secure shell client, finding previously undiscovered bugs in all three.

The Aftersight system relies on the concepts of heterogeneous replay and parallel analysis. Heterogeneous replay is the ability to record and replay events at the same time, thus increasing the speed and timeliness of analysis. Parallel analysis allows analysis and system execution simultaneously, further increasing the timeliness of the results. Implementing heterogeneous replay and parallel analysis presents several technical challenges. First, keeping target and analysis systems in sync with each other without slowing the target system down is difficult. The analysis is typically slower than the target system, and there are times when the target system must be blocked because of resource allocation issues. This limitation can be overcome by additional buffering in the target system and further refinement and tuning of the analysis system to speed it up. However, there are situations where the analysis system just cannot keep up, so additional techniques such as forward caching and buffering can be applied. Also, the addition of more processing cores in the system can help offload the analysis task through further parallelization.

This talk gave several compelling reasons for using dynamic program analysis and showed how decoupling the execution and analysis environments can significantly improve productivity and effectiveness. The Aftersight system appears to have many useful applications in the development, test, and security worlds. The audience was greatly intrigued and several questions arose on the difference between the decoupled approach and existing parallel environments. The difference is at what level the recording and playing occur. Jim stated that recording at the OS level incurs more overhead than recording at the hypervisor level.

- ***Protection Strategies for Direct Access to Virtualized I/O Devices***
  Paul Willmann, Scott Rixner, and Alan L. Cox, Rice University

Paul Willmann, now with VMware, presented performance and safety measures of several strategies for access control to I/O devices from within virtualized environments. Direct access to I/O devices is required in many datacenter applications where high throughput performance is needed; however, access to these devices needs to be controlled to protect from an untrusted virtual machine (VM) tampering with or using devices it does not have permission or privilege to use.

Wallmann presented implementations of protection strategies in both hardware and software, with surprising results. Hardware implementations utilize an Input Output Memory Map Unit (IOMMU) to implement single-use mappings, shared mappings, persistent mappings, and direct mapping strategies. The software strategy is an implementation of the single-use mapping that requires the guest OS's drivers to register with the virtual machine monitor (VMM) before access is granted to the device. Both hardware and software implementations have advantages and disadvantages that are evaluated in the paper.

The different strategies were evaluated based on performance and protection capability in inter-guest and intra-guest protection categories. Three types of invalid accesses were evaluated for each strategy and for each category: bad address, invalid use, and bad device. The results showed that hardware implementations worked very well and efficiently for intra-guest protections, but it did not perform well for inter-guest protection. The software implementation performed well in all inter-guest protections except for the bad device case. Additionally, the software method provides additional protection in the intra-guest invalid use case.

With all the strategies showing very good overall protection, the biggest differentiator among the various strategies becomes performance-related. Several benchmarks were run against the strategies, including a TCP stream, a VoIP server, and a Web server. The benchmark also tested against various levels of mapping reuse. The results showed that the single-use strategy had the highest inter-guest overhead, at 6%–26% of CPU workload; however, significant mapping reuse can greatly reduce that overhead. Persistent mappings showed the highest performance, at only 2%–13% overhead with nearly 100% reuse. The software implementation showed better performance than two of the hardware strategies (single-use and shared), with 3%–15% CPU overhead. The direct-mapped hardware strategy was the best performer, although it had limited intra-guest protection capability.

The surprising result is that the software protection strategies utilized in this paper provide performance comparable to or better than the hardware IOMMU results while still maintaining strict inter-guest and intra-guest protection.

- ***Bridging the Gap between Software and Hardware Techniques for I/O Virtualization***
  Jose Renato Santos, Yoshio Turner, and G. (John) Janakiraman, HP Labs; Ian Pratt, University of Cambridge

Jose Renato Santos's talk was on improving I/O performance in virtual machines through combining hardware and software techniques. In a virtualized environment, physical devices need to be multiplexed so that each guest virtual machine (VM) can use the device. This multiplexing can be handled in software and hardware, each with its advantages and disadvantages. Software incurs a significant overhead in managing the device; however, the driver is simplified by providing a transparent interface as I/O access is handled by the host OS using a device-specific driver and the guest can use a standard virtual device driver independent of the hardware. The hardware approach is more complicated since the transparency is reduced, requiring each guest VM to have a device-specific driver; however, the performance is usually much better.

The HP Labs research team wanted to reduce the performance gap between driver domain model and direct I/O while maintaining transparency. To do so, they analyzed the Xen device driver model, focusing on the networking receive path, and compared the same workload with a direct I/O approach. They focused on several areas to improve the performance of the Xen driver. First, they reduced the data copy cost by keeping all copies between guest and driver domains on the same CPU to increase cache hits. Next, they avoided extra data copies by using dedicated NIC receive queues. Finally, they reduced the cost of the grant mechanisms, the second highest cost in Xen, by maintaining grants and mappings across multiple accesses. The team was able to reduce the receive path execution costs for a conventional NIC by 56%. For devices with multiple hardware receive costs, they were able to achieve performance near direct hardware I/O while maintaining the benefits of the Xen driver model. This is a significant improvement in performance over the original driver domain model in Xen.

By keeping the new multi-queue completely hidden from the guest and encapsulated in the driver domain, migration to the new driver is completely transparent to the guest. The team has stated that the new mechanisms will be updated in the Xen Netchannel2 in approximately 2–3 months. The next improvement they plan to make will be to look at high-bandwidth (i.e., 10 GigE and multiple guests) improvement in the Xen driver.

INVITED TALK

■ ***Free and Open Source as Viewed by a Processor Developer***
Peter Kronowitt, Intel

Summarized by Ward Vandewege (ward@gnu.org)

Peter Kronowitt's talk grew from an internal Intel presentation. He works in the Software Solutions Group, which optimizes software—all sorts of software, ranging from embedded to server. The purpose of the optimization is to ensure that when the product reaches the marketplace, there is a complete hardware and software solution.

The traditional software-enabling model at Intel goes something like this. Intel works with over 12,000 software companies. Most of these are proprietary, so Intel has to sign nondisclosure agreements (NDAs). Then engineers are assigned; they need time to get the work done, and then Intel has to wait for the market to generate demand in order to get to a mutually beneficial state for Intel and its partners.

Open source development is very different. Intel feeds software into the kernel. That software then gets picked up by community distributions such as Debian, Fedora, and OpenSuse, and those in turn feed into the products of Linux companies such as Canonical, RedHat, and Novell. This is a much more efficient model.

Intel has learned to work more effectively with kernel developers: In 2001, Alan Cox, a core kernel developer, gave direct feedback that Intel required many NDAs and was secretive about its hardware, making it very difficult to work with. Fast forward to 2007 when Alan Cox said that Intel is one of the most cooperative hardware vendors, providing good docs, errata, and software such as graphics drivers. In those six years, Intel has learned and relearned a lot of stuff.

Linux is estimated to be one-third of the market based on server shipments today. But tracking open source software (OSS) is very difficult. This is a problem—if Intel can't tell what software customers are using, it cannot put its resources in the right place to make sure the hardware works perfectly. Intel needs to know what software customers are using and deploying in order to be able to offer a "complete solution." Also, OSS is growing three times as fast as proprietary software.

Intel has been growing its open source involvement over the years, starting in 1990 when Linus Torvalds booted Linux on Intel Architecture for the first time. He was able to do that because Intel had released detailed specifications for the Intel Architecture. Since 2003, Intel has become more visibly active as a contributor to OSS. The following paragraphs highlight some examples of how Intel has been working with the OSS community over the years.

The PC BIOS had not changed for over 20 years. Intel launched the Tiano project to replace it. This was done in partnership with CollabNet, establishing the extensible framework interface (EFI) dev kit. From this, Intel learned how open source can drive industry change.

In 2003 Intel joined other vendors in a virtualization research project called Xen at Cambridge University in the UK. In 2004 Intel started contributing a large amount of code to the open-source project. Today a large ecosystem exists around virtualization, and Intel has been contributing to many projects in that space. Xen helped catalyze Intel feature adoption by vendors of virtualization products.

The telecom industry was a highly proprietary, vertically integrated industry that overinvested during the dot-com era. Intel was a founding partner of the Open Source Development Labs (OSDL), contributing to the kernel and the Carrier Grade Linux (CGL) specification. When the dot-com bubble burst, the carriers needed to cut costs, and Intel's involvement with CGL helped the Intel Architecture break into the telco industry.

In the late 1990s, Merced, the Itanium platform, solidified numerous operating system porting commitments. Intel worked with many OS vendors and indirectly contributed to the Linux kernel. Linux and Itanium helped Intel gain access to the RISC market.

Initially, Intel made Linux kernel contributions via proxy. This meant that Intel was not very visible as a community member. After long, difficult internal negotiations on open sourcing drivers, Intel started contributing code directly to the kernel. This direct participation in the community has accelerated Intel technology adoption.

Influencing Java was . . . challenging. Intel, like numerous other industry players, requested that Sun open source Java. Eventually, Intel participated in the launch of the Harmony project with other industry players, including IBM. Harmony was a clean-room OSS Java implementation. Eventually, this encouraged Sun to release an OpenJDK.

More recently, Intel has been working on Moblin, an optimized software stack for Atom-based clients. This software stack is aimed at mobile Internet devices, netbooks, cars, etc. See http://moblin.org for more information. Intel also launched LessWatts.org, an Intel open source project to make Linux greener.

DISK STORAGE

Summarized by Christopher Stewart (stewart@cs.rochester.edu)

■ ***Idle Read After Write—IRAW***
Alma Riska and Erik Riedel, Seagate Research

When users issue writes to a disk, they assume their exact data has been stored. However, mechanical anomalies can cause the data actually stored on disk to deviate from the user's original data (a.k.a. data corruption). Worse, such corruption can be silent, causing the user to wrongly believe their data was correctly written to the disk. Alma Riska presented Idle Read After Write (IRAW), a low-overhead approach to detecting silent data corruption. IRAW issues a disk read for recently written data during periods when the disk is idle. The data returned by the read is compared to a cached copy of the actual data the user intended to write to the disk; if the two differ, appropriate recovery actions are taken (e.g., retry).

Compared to a standard disk, IRAW improves reliability by validating writes soon after they occur. An alternative is to validate each write immediately after it happens (RAW). RAW improves reliability, but it degrades performance by placing an additional disk operation on the critical path of every write. In comparison, IRAW delays the validation until the disk is idle, and therefore it hides the cost of the additional read from the end user. IRAW therefore requires enough idle time for the additional disk operations to complete. Alma presented empirical evidence from five disk traces, all of which had more than enough idle time to perform the delayed reads.

Empirical results using IRAW show that it indeed has low overhead. One experiment showed that the performance of an IRAW-enabled disk almost matched that of a standard disk for a Web server application. Further, IRAW may not degrade other performance-enhancing disk operations. For instance, many applications can benefit by enabling IRAW and idle wait simultaneously. Finally, Alma showed that the footprint of IRAW in the disk cache was not too large for today's disks.

Adam Leventhal from Sun Microsystems asked whether IRAW could be applied at the filesystem level. Alma said that it is possible, but the file system will probably be less effective at identifying true idle time on the disk. Geoph Keuning from Harvey Mudd College asked whether it was even important to be concerned with the amount of cache space dedicated to IRAW, since volatile memory is getting cheaper. Alma said that one design goal was to make IRAW practical for today's disks, which meant keeping the footprint below 4–6 MB.

■ ***Design Tradeoffs for SSD Performance***
Nitin Agrawal, University of Wisconsin—Madison; Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy, Microsoft Research, Silicon Valley

Solid-state disks (SSDs) can perform certain I/O operations an order of magnitude faster than rotating disks. They have the potential to revolutionize storage systems. However, little is known about the limitations of their architecture. Nitin Agrawal discussed several inherent challenges for SSD devices and proposes solutions. The analysis is based on a detailed understanding of the architecture of SSD devices. For instance, a write to an SSD block requires that the block's contents be erased and rewritten. Further, SSD blocks can only be erased a certain number of times. Such architectural properties affect the performance and reliability of SSDs.

The granularity of writes affects the performance of SSDs. Specifically, workloads that perform writes to random locations on disk perform orders of magnitude worse than those that perform random reads. Empirical evidence showed a difference of 130 random writes per second compared to almost 20,000 random reads per second. Nitin demonstrated that properly mapping logical pages to physical blocks can improve the performance of random writes. A second performance challenge faced by SSDs is bandwidth bottlenecks. Striping and interleaving are good solutions to mitigate the bandwidth bottleneck by distributing I/O for logical blocks across multiple channels. Intuitively, this solution exploits the potential for parallelism in storage access patterns. Finally, SSD blocks wear down after a certain number of erasures and rewrites. To maximize the lifetime of the device, Nitin proposed a novel wear-leveling algorithm that increases the usable lifetime of an SSD by delaying expiry of any single block.

Jason Flinn from the University of Michigan asked Nitin about the benefit of wear-leveling, given that the whole device will wear out eventually anyway. Nitin said that wear-leveling reduces the long-term cost of SSDs, since the failure of individual blocks could force a company to purchase a new device when only a small portion of its capacity is unusable. Sean Rhea noted that wear-leveling is most beneficial for applications that frequently write to only a few pages but access many pages for reading (i.e., small hot set and large cold set). Nitin agreed.

■ ***Context-Aware Mechanisms for Reducing Interactive Delays of Energy Management in Disks***
Igor Crk and Chris Gniady, University of Arizona

Igor began by saying that most disks now support different power modes for energy conservation. The disk can be powered down during idle times to consume less energy and then spun up to an operational power mode when I/O requests arrive. In today's interactive systems, the additional latency for I/O requests that interrupt idle periods (i.e., the time for a disk spin-up) is typically seen by the end user (who then gets miffed and maligns the system as slow and unresponsive). Igor presented a mechanism to hide spin-up latency from end users by preemptively changing the disk to full-power mode before I/O requests happen. The key is to identify end-user GUI events that signal that a disk I/O is imminent. When such events happen during an idle period, the disk can be moved to an operational power mode in anticipation of the impending request. For instance, a mouse click on a "file open" button may be a good signal that an I/O request is imminent and the disk should preemptively be spun up.

Compared to today's default policy (no preemptive spin-up), the proposed solution can hide spin-up delays from the end user. Further, by considering the context of the GUI event, the proposed solution can achieve better energy conservation than a naive solution that preemptively spins up after every mouse click. Context information was collected by intercepting calls to the X windows server. Specifically, each X windows event updated a table that tracked the number of times that the event occurred in a particular context and the number of times it was followed by I/O. After data was collected for a long period of time, the event contexts that were most likely to be followed by disk I/O were tagged as good predictors. Empirical results show that preemptive action based on the identified predictors does hide the latency of disk spin-up from end users, while conserving more energy than a naive approach that does not consider the context of the event. Further, Igor mentioned that the proposed system allows users to trade off the latency they see for more energy conservation by adjusting the threshold at which an event qualifies as a predictor.

Yu Chen from Fermilab asked whether they were able to accurately predict disk requests for systems that had a large file system cache. Igor noted that the difference between file system requests and disk I/O was a significant challenge. In the current implementation, they identify the GUI events likely to cause file system requests and apply a heuristic to predict disk requests. Christopher Stewart from the University of Rochester noted that user satisfaction, as measured by Mallik et al. at ASPLOS 2008, could guide the setting of the threshold that determines when an event qualifies as a predictor. Igor agreed that the combination of the two works could be beneficial. However, he noted that GUI events can predict I/O well in advance, so a combination of the techniques may significantly affect performance.

NETWORK

Summarized by Matthew Sacks (matthew@matthewsacks.com)

■ ***Optimizing TCP Receive Performance***
Aravind Menon and Willy Zwaenepoel, EPFL

Aravind Menon demonstrated the ability to improve TCP performance by focusing on the receive side of the TCP protocol. Menon argued that receive-side optimizations are missing, contributing to lesser performance of the TCP protocol. Linux was used as the demonstration OS for his concepts, although the same principles can be applied to any operating system. Menon shows that there are two types of overhead: per-byte optimizations and per-packet optimizations. Per-packet overhead costs are the primary overhead contributor on newer CPUs, whereas on older processors the issue was with per-byte overhead.

Menon presented two types of performance improvements in his talk: receive aggregation and TCP acknowledgment offload. Receive aggregation aggregates multiple incoming network packets into a single host packet accounting for a 45%–86% increase in performance. Receive aggregation requires that the packet must be the same TCP connection, must be in sequence, and must have identical flags and options. Receive aggregation works best when receiving at a high rate of transfer. To implement this method in Linux the network driver must allocate raw packets rather than sk_buffs.

For acknowledgment offloading, the normal method of generating ACK packets, by a one-to-one mapping, is replaced by a template to generate the ACK packets, which in turn avoids buffer management costs. This must be done at the device-driver layer. Nonprotocol overhead has the greatest impact on TCP performance such as buffer management, and, specifically for the Linux driver, it also processes MAC-level analysis of each packet.

■ ***ConfiDNS: Leveraging Scale and History to Detect Compromise***
Lindsey Poole and Vivek S. Pai, Princeton University

Lindsey Poole presented a new project called ConfiDNS, which is based on the CoDNS cooperative DNS resolver system. CoDNS is a wrapper for local DNS resolution that allows faster lookups and high availability for DNS lookups. CoDNS utilizes PlanetLab for ensuring high availability as a distributed service.

ConfiDNS takes the CoDNS project and addresses the security vulnerabilities in CoDNS, which is susceptible to contamination from a single resolver being propagated throughout the entire system. The way ConfiDNS works is that when the local resolver fails, it forwards the request to peer nodes on the PlanetLab network (a feature that was present in CoDNS). ConfiDNS preserves a history of lookups and the client can specify policies for DNS lookups.

Another problem encountered with CoDNS is DNS lookups served by global content distribution networks, which may return multiple IPs from different locations for the same hostname. ConfiDNS addresses this problem by implementing a peer agreement algorithm that compares results from multiple resolutions from different geographic locations and then returns a result.

ConfiDNS proves that you can improve DNS resolution performance without compromising security. DNS attacks on the local system are much easier to carry out. ConfiDNS protects against attacks such as cache poisoning or spoofing, and it improves performance at the same time.

- ***Large-scale Virtualization in the Emulab Network Testbed***
  Mike Hibler, Robert Ricci, Leigh Stoller, and Jonathon Duerig, University of Utah; Shashi Guruprasad, Cisco Systems; Tim Stack, VMware; Kirk Webb, Morgan Stanley; Jay Lepreau, University of Utah

Emulab, a network testbed at the University of Utah, allows researchers and engineers the ability to specify a network topology including server systems to which you have root access. One of the difficulties that the Emulab maintainers experienced was a limitation in the amount of hardware available to them; therefore, a virtual solution for the network and systems was needed to power the Emulab testbed. One of the requirements of the virtual solution was that the virtual environment needed to retain the same fidelity of experiments running on the testbed so that the results would not be affected. At first FreeBSD jails were used to address this; however, jails alone were found to fall short in addressing the issue of network virtualization, so the Emulab team designed a more robust virtualization platform that expanded on the FreeBSD jail's limitations.

The team at Emulab implemented a robust network virtualization solution by developing a virtual network interface device, which is a hybrid encapsulating device and bridging device. The "veth" interface allows creation of unbound numbers of Ethernet interfaces, which then communicate transparently through the switch fabric. Veth devices can be bridged together or with physical interfaces to create intra-node and inter-node topologies. In addition to virtual network interfaces, the Emulab team also had to implement virtual routing tables that are bound to each jail and virtual interface based on the Scendaratio and Risso implementation, which implements multiple IP routing tables to support multiple VPNs. Also, for the virtual nodes themselves, the Emulab team designed a resource-packing methodology called "assign" which "packs" virtual hosts, routers, and links into as few physical nodes as possible without overloading the physical nodes. This method allows up to a 74:1 compression ration of virtual nodes/networks to physical hosts.

The research done on the Emulab testbed in addressing these scaling issues with virtual networks and nodes has enabled the team to scale efficiently while keeping the same fidelity as strictly physical hardware by using virtual interfaces and resource packing. The efficiencies achieved in the Emulab virtualization implementation now allow experiments to be executed on up to 1000 nodes, permitting powerful simulations without impact onm the fidelity of the experiments.

INVITED TALK

- ***Millicomputing: The Future in Your Pocket and Your Datacenter***
  Adrian Cockcroft, Netflix, Inc., and Homebrew Mobile Club

  Summarized by Tom Clegg (tom@tomclegg.net)

Low-power computing devices such as mobile phones—which Adrian Cockcroft calls "millicomputers," because their power requirements are measured in milliwatts rather than watts—are increasing in capacity faster than their hundred-watt datacenter counterparts. In this talk, Cockcroft gave an overview of the current state of low-power technology and cheap open hardware in particular, considered some of the applications that become possible as mobile devices approach the capacity of personal computers, and outlined a speculative "enterprise millicomputer architecture" employing thousands of low-cost nodes per rack.

In 2007, the iPhone was notable for running a full Mac OS rather than a cut-down embedded operating system—it ships with 700 MB of system software. Clearly, portable millicomputers such as the iPhone provide a real application platform. Cockcroft showed photos of a prototype "myPhone"—a Linux-based GSM/EDGE phone with many built-in features and connectivity options, and CPU and RAM specifications similar to the iPhone. In 2008, the emergence of Google Android as an open source alternative to the iPhone platform has generated a lot of developer interest. The highest-performance smart phone hardware will raise the bar further with 256 MB RAM, 16–64 GB storage, twice the CPU speed, and faster networking. AT&T plans to implement HSPA release 7 in 2009, which will deliver speed "exceeding 20 Mbps" and has a "clear and logical path" to 700-MHz 4G access in the 2010 timeframe, which should increase speed to nearly 100 Mbps. Meanwhile, short-range low-power networking is reaching 480 Mbps as Ultra-Wideband Wireless USB starts to roll out. Nonvolatile storage is steadily becoming cheaper, and emerging storage technologies promise dramatic speed increases in a few years. In the CPU market, we can expect 1-GHz quad-core processors to arrive in 2010.

Given the pace of mobile technology advances, the time is coming into view when pocket devices with wireless docking can replace laptop computers, just as laptops replaced desktop computers for many users. Combining workstation computing power with mobile connectivity, we might see "life-sharing" applications such as full-time video conferencing and virtual world integration. Integrating accessories such as an accelerometer, compass, and brainwave reader, we have a system with many possible uses such as computer-assisted telepathy, ambient presence, immersive personal relationships, and better ways to monitor and care for physically disabled people.

In addition to mobile applications, these tiny low-power computers have potential applications in the datacenter. They could help reduce power consumption, which is already a limiting factor in many situations. Cockcroft presented one possible architecture to demonstrate how a computing cluster might be constructed using low-cost mobile device boards. Modules packed onto a 1U enterprise motherboard yield a fully distributed heat model that is much easier to cool than a typical server board. Two groups of seven modules are connected via USB switches to each of eight gateway/load balancer nodes, each having two gigabit network interfaces. Thus, each rack unit has a total of 112 CPUs and 28 GB of RAM, consuming 24 W when idle and 160 W at peak power. Adding 8 GB microSDHC cards with 20 MB/s I/O each, we have 896 GB per rack unit of storage with 2240 MB/s I/O. The $14,000 cost of this system is comparable to a 1U Sun server with similar specifications—but the millicomputer offers much faster storage I/O with zero seek time and more network bandwidth, using little more than half the power.

Software implications of this platform include a small application memory limit (256 MB) on par with mainstream systems from 2001. Management implications include the need for lightweight monitoring, aggregation tools, and load balancing. This platform would be well suited to horizontally scalable applications such as Web content delivery, legacy applications that could run on five-year-old machines, storage I/O-intensive applications, and graphical video walls.

One participant pointed out that the low bandwidth between nodes could be a serious limitation. Cockcroft explained that the USB approach was taken to minimize power consumption, and the CPU power is not enough to saturate a gigabit network interface in any case. This feature of the design makes it more suitable for applications with low IPC demands, such as Web servers. It would also be possible to use other low-power interconnects, perhaps based on FPGA technology, which would give better interconnect bandwidth. It should also become less of an issue as RAM size increases. Another participant suggested a heads-up display with facial recognition software as an interesting mobile application. Cockcroft added that, although signal processing chips can be a big power drain, many processing tasks can be postponed until nighttime, when the device is plugged into a charger and it's acceptable for it to get a bit hotter than comfortable pocket temperature. Another participant brought up the possible impacts of mobile technology on the way we interact with services; Cockcroft referred to "taking the friction out of interactions" with always-on networking and services such as continuously updated status tracking. Another participant wondered whether this mobile power could simply do away with the role of the data center; Cockcroft offered that, although there tends to be a pendulum alternating between client and server focus, there will likely always be a place for centralized services, but certainly more can happen in the pocket.

Current information on millicomputing can be found at http://millicomputing.blogspot.com/.

FILE AND STORAGE SYSTEMS

Summarized by Zoe Sebepou (sebepou@ics.forth.gr)

- ***FlexVol: Flexible, Efficient File Volume Virtualization in WAFL***
  John K. Edwards, Daniel Ellard, Craig Everhart, Robert Fair, Eric Hamilton, Andy Kahn, Arkady Kanevsky, James Lentini, Ashish Prakash, Keith A. Smith, and Edward Zayas, NetApp, Inc.

John Edwards presented their work on a new level of indirection between physical storage containers (aggregates) and logical volumes (FlexVol volumes). An aggregate consists of one or more RAID groups, and its structure resembles that of a simple file system, keeping the changes made on the individual FlexVol volumes. The main goal of FlexVol was to provide new functionality by decoupling the physical device management from the data management. The decoupling strategy gives administrators the flexibility to enforce different policies on different volumes and to dynamically grow or shrink the volumes.

The mapping between the virtual block addresses of FlexVol and the physical addresses used by aggregates requires extra processing and disk I/O to deal with the address translation of each indirect block. This challenge is addressed with two main optimizations: dual block numbers and delayed block freeing. Block pointers in a FlexVol volume have two parts: the logical location of the block in the container and its physical location. In delayed block freeing, free space is held by the aggregate, not the volumes, so one counts the number of delayed free blocks and performs a background cleaning after a specific threshold. These optimizations help to reduce the overhead and result in at most a small degradation in the system's overall performance compared to traditional volume approaches.

The evaluation of FlexVol was made through the use of micro-benchmarks, including the comparison of read and write in sequential and random access patterns. Their results indicate that FlexVol performance is almost identical to that of the traditional volumes, and in the worst cases the performance difference is from 4% to 14% (mostly in random cases involving metadata overhead in write operations). Finally, Edwards provided some insight into the current use of FlexVol and its services, showing the growing adoption of FlexVol by their customers.

■ ***Fast, Inexpensive Content-Addressed Storage in Foundation***

Sean Rhea, Meraki, Inc.; Russ Cox and Alex Pesterev, MIT CSAIL

Sean Rhea presented Foundation, a preservation system based on content-addressed storage (CAS) aimed at providing permanent storage of users' personal digital artifacts. Sean pointed out that the increasing use of computers to store our personal data would lead to the undesired situation that this data would be unavailable in the future. Indeed, as software and hardware components depend on each other to make an application operate and provide the desired functionality, a user in the future would need to replicate an entire hardware/software stack in order to view the old data as it once existed. To overcome this problem, the authors, inspired by Venti, designed and developed Foundation. Foundation differs from Venti mostly in that instead of using an expensive RAID array and high-speed disks, it only uses an inexpensive USB hard drive, making the deployment of this system easy and possible for consumer use.

Foundation permanently archives nightly snapshots of a user's entire hard disk containing the complete software stack needed to view the data (with user data and application and configuration state of the current system captured as a single consistent unit). To eliminate the hardware dependencies, Foundation confines the user environment to a virtual machine. As in Venti, the use of content-address storage allows Foundation to have limited storage cost, actually proportional to the amount of new data, and to eliminate duplicates through the use of a bloom filter; other filesystem-based approaches miss this benefit.

The major components of Foundation include the Virtual Machine Monitor (VMM), the filesystem Snapshot Server (SMB), the virtual machine archiver, and the CAS layer, whose main use is to store the archived data on the inexpensive external disk and/or replicate it using a remote FTP server. The users operate on an active virtual machine which runs on top of the VMM. The VMM stores the state of the virtual machine in the local filesystem and every night the virtual machine archiver takes a real-time snapshot of the active VM's state and stores the snapshot in the CAS layer. The SMB server is used to interpret the archived disk images and present the snapshots in a synthetic file tree, accessible by the active VM over the server.

To eliminate several of the problems that appear in similar systems such as Venti, their proposed solution to reduce disk seeks is to reduce as much as possible the hash table lookups. In the case of writing, lookups occur when the system needs to update a block index and when determining whether a block has been accessed before. In these cases Foundation uses a write-back index cache that is flushed to disk sequentially in large batches. During read operations, lookups are required in order to map hashes to disk locations. In this case they start with the list of the original block's hashes, they look up each block in the index, and they read blocks from the data log and restore them to disk. Moreover, with the use of CAS they take advantage of the fact that, given a block, CAS gives back an opaque ID. This allows block locations to be used as IDs, completely eliminating read-indexing lookups and thus still allowing for potential duplicate finding using hashing.

For the evaluation of the Foundation system, the authors focused on the performance of saving and restoring VM snapshots. The important metrics taken into consideration were how long it takes for Foundation to save the VM disk image and how long it takes to boot old system images and recover old files from the directory tree. Foundation's algorithm in its two modes, by-hash and by-value, was compared against Venti's algorithm. The results indicate that Foundation operates efficiently and gives higher read and write throughput in the majority of the tested cases compared to Venti. Sean Rhea concluded that Foundation is a consumer-grade CAS system that requires only a USB drive and can be used not only as a preservation system but also as an inexpensive household backup server. Moreover, it can automatically coalesce duplicate media collections and operates efficiently without requiring a collision-free hash function.

■ ***Adaptive File Transfers for Diverse Environments***

Himabindu Pucha, Carnegie Mellon University; Michael Kaminsky, Intel Research Pittsburgh; David G. Andersen, Carnegie Mellon University; Michael A. Kozuch, Intel Research Pittsburgh

Himabindu Pucha described dsync, a file transfer system that can correctly and efficiently transfer files in a wide range of scenarios. By choosing to use all the available resources (the sender, the network peers, and the receiver's local disk) and by constantly monitoring recourse usage, dsync overcomes performance limitations present in other similar systems such as rsync and peer-to-peer systems such as BitTorrrent. Although the primary resource used by dsync is the network, dsync dynamically chooses, if necessary, to spend CPU cycles and disk bandwidth to locate any relevant data on the receiver's local file system in order to enhance performance.

dsync retrieves chunks over the network either from the sender or from any available peer that has downloaded the same or similar data. It also makes the optimization to look at the receiver's local disk for similar data by spending some of the system's CPU resources to compute the hash of data from the local disk and for scheduling purposes. Specifically, dsync source divides each file (or file tree) in equal-sized chunks and by hashing the chunks computes for each chunk a unique ID. A tree descriptor is then created describing the file layout in the file tree, the metadata, and the chunks that belong to each file. So, given a tree descriptor, dsync attempts to fetch the file chunks from several resources in parallel using the optimal resource at any given time.

The evaluation of dsync was done for several transfer scenarios; results for single receiver (one source—one receiver) and multiple receivers (in homogeneous/heterogeneous environments—PlanetLab nodes) indicate that dsync can effectively use the available resources in any environment. Moreover, the back-pressure mechanism allows for optimal resource selection and the heuristics used quickly and efficiently locate similar files in real file systems.

One questioner asked whether they have attempted to find a solution that is globally good, given that resources are to be shared among several receivers. The answer was that currently each receiver greedily uses the resources to minimize its download time, but they would like to look at strategies that enable cooperation among receivers to improve their performance.

KEYNOTE ADDRESS: THE PARALLEL REVOLUTION HAS STARTED: ARE YOU PART OF THE SOLUTION OR PART OF THE PROBLEM?

David Patterson, Director, U.C. Berkeley Parallel Computing Laboratory

Summarized by Christopher Stewart (stewart@cs.rochester.edu)

Patterson began by saying that his speech was motivated by the revolution under way in computer architecture: Microprocessors are out; parallel architectures are in. Patterson argued that the design shift from microprocessors is inevitable, so the systems community would do best by embracing parallel architectures and finding solutions to the new challenges they present. "I wake up every day and can't believe what is happening in hardware design," Patterson said. "We are in a parallel revolution, ready or not, and it is the end of the way we built microprocessors for the past 40 years."

Although the end of the microprocessor is inevitable, Patterson noted that the current movement toward parallel architectures could fail without ever achieving success. In particular, past companies based on parallel architectures have all failed. But this time, he argued, the consequences of failure would likely be more severe and widespread. Despite the history, Patterson said that he is optimistic that the parallel revolution could succeed this time, for several reasons. First, there will not be fast microprocessor alternatives to parallel architectures. Second, the open-source community will build software that takes advantage of parallel architectures. Third, emerging software trends (especially software as a service) are well suited for parallel architectures. Fourth, FPGA chips will decrease the time necessary to prototype new designs. Finally, necessity is the mother of innovation.

Of course, Patterson's optimism was restrained, since many obstacles must be overcome before the parallel revolution can be realized. In the remainder of his talk, Patterson described several challenges, or research themes, as they relate to the systems community and the approaches being taken by the Parallel Computing Lab to solve them. The challenge that he mentioned first is that there is not yet a "killer app" for parallel architectures. Patterson argued for an application-centric solution in which researchers take cues from domain experts. So far, his research group has identified potential applications such as the re-creation of 3-D sound in ear buds, accelerators for hearing aids, image-based search, modeling of coronary heart disease, face recognition, and a parallel Web browser. Adapting single-threaded applications written in old languages was the next challenge addressed. Patterson argued that such applications can be transparently improved by identifying common design patterns that can be parallelized. Following the lead of Christopher Alexander's book *A Pattern Language*, Patterson argued for 13 design patterns, which he called motifs, that if properly researched could improve performance for a range of applications.

Patterson's third discussion point was about the difficulty of developing parallel software. He advocated a two-layer approach. The first layer is the efficiency layer, which would be developed by 10% of the programming population. Software at this level consists of smart and lightweight operating systems, hypervisors, and compilers that automatically compose and optimize applications. The second layer is the productivity layer, where novice programmers encode domain-specific logic in high-level languages.

The fourth challenge was to develop a scalable lightweight operating system for parallel architectures. Current virtual machine monitors are a good step in this direction.

Finally, power conservation remains an important issue, even for parallel architectures. Patterson's group is using runtime data on power consumption and performance to inform compiler-level autotuners, the OS scheduler, and adaptable software components. This challenge is especially important for datacenters and handheld devices.

Patterson concluded by urging the systems community to seize this opportunity to reinvent "the whole hardware/software stack." His parting words were, "Failure is not the sin; the sin is not trying."

Andrew Tannenbaum noted that a crash every two months is not acceptable to most people, yet it seems to be the best that we can do with sequential programming. Since parallel programming is harder by at least an order of magnitude, how will we create software that satisfies user demands for reliability? Patterson agreed that reliability is an important problem for parallel software. He suggested revisiting software solutions that were proposed for previous parallel architectures and emphasized that a solution is critical for the parallel revolution to be successful.

Rik Farrow complimented Patterson's research agenda and broad vision. He suggested that the systems community should also consider redesigning basic primitives, such as the operating system's trapping mechanism and methods for inter-processor communication. Patterson agreed and noted the need for cooperation between the systems and architecture community in optimizing such primitives.

Jeff Mogul wondered whether Patterson's approach would fit the needs of the common developer. In particular, Patterson's motifs seemed to reflect the patterns in scientific computing and not necessarily everyday applications. Patterson argued that the motifs do cover a wide range of applications. But he noted that motif-based research is just underway, and the real benefit will be evident as more applications are developed for parallel architectures.

WEB AND INTERNET SERVICES

Summarized by Tom Clegg (tom@tomclegg.net)

- ***Handling Flash Crowds from Your Garage***
  Jeremy Elson and Jon Howell, Microsoft Research

Jon Howell began by observing that a single server in your garage can provide enough power to deploy a cool new Web application and make some money with minimal startup costs. However, if your service gets popular too suddenly, the burst of traffic can easily bring down your garage server completely. Utility computing services make it possible to accommodate flash crowds cheaply by adding servers on short notice and turning them off when they're no longer needed. Howell presented a survey of techniques for using utility computing to achieve load balancing and fault tolerance for Web services.

The survey covered four basic approaches: storage delivery networks, HTTP redirection, middlebox load balancing, and DNS load balancing. Each technique was evaluated using five criteria: applicability to different types of applications, limits of scalability, implications for application development, response to front-end failure, and response to back-end failure.

Storage delivery networks are easy to use and are suitable for serving idle content such as video files. HTTP redirection works by assigning each client to a single back-end server. This client-server affinity makes application development easier, but it is possible for clients to be bound to a broken back-end server, and a front-end failure prevents any new sessions from starting. An experiment with 150 clients and 12 back-end servers resulted in only 2% load on a single front-end server, suggesting that a single redirector could handle 7,500 clients. A middlebox load balancer associates clients with back-end servers by looking at layer 4 (TCP source port number) or layer 7 (HTTP cookie). An advantage to this technique is that it does not involve the client's participation. However, a front-end server failure is fatal to all sessions. DNS load balancing assigns clients to back-end servers by selecting and reordering a list of IP addresses when responding to queries. DNS load balancing scales very well, but it is complicated by DNS caches, resolvers, and client software. Experiments showed a huge variance in failover time on different operating systems, with the Mac OS X resolver library taking up to 75 seconds to failover to a second IP address. Also, a significant portion of clients sort the list of IP addresses and contact the lowest-numbered server first, thereby defeating the load balancing system. A hybrid approach might use a static delivery network for static content and a load-balanced cluster for active content or use DNS to balance load among several fault-tolerant middlebox load balancers, which can compensate for the sluggishness of DNS failover.

Howell shared some lessons learned from a CAPTCHA service (Asirra) and a password reminder service (Inkblot-Password), both of which handled flash crowds reasonably well. The CAPTCHA service used DNS load balancing to select a back-end server, which provides a session ID so that misdirected queries can be identified and forwarded to the correct back-end server. Occasional misdirected requests were forwarded to the correct server. Some requests failed because of utility computing back-end failures, but users could simply retry. An attempted denial-of-service attack was apparently abandoned after it failed to bring down the service.

One attendee observed that the middlebox and DNS techniques have complementary characteristics; Howell agreed that it would be worthwhile to evaluate a hybrid approach using those two techniques. Another question was why DNS address list sorting didn't prevent the DNS load balancing from being effective; Howell noted that Linux accounts for a relatively small portion of clients and that the DNS servers could help work around the behavior by returning only a subset of the full back-end server list to each query. In response to another audience question, Howell said he would be able to make the survey data available to the public.

- ***Remote Profiling of Resource Constraints of Web Servers Using Mini-Flash Crowds***
  Pratap Ramamurthy, University of Wisconsin—Madison; Vyas Sekar, Carnegie Mellon University; Aditya Akella, University of Wisconsin—Madison; Balachander Krishnamurthy, AT&T Labs—Research; Anees Shaikh, IBM Research

Most Web servers rely on overprovisioning to handle flash crowds, because it is difficult to obtain data about server resource limitations. Administrators are reluctant to perform stress tests on production servers, and testbed environments are often configured so differently that test results would not be a good indicator of the production Web server's performance. Pratap Ramamurthy presented a technique for measuring resource limitations of a production Web server without adversely affecting regular usage.

The "mini-flash crowd" service employs a distributed set of clients, synchronized by a controller, to simulate flash crowds. The controller conducts a number of experiments, each designed to test the limitations of a specific resource; for example, to test network bandwidth, the clients download large static files from the target server. Each experiment begins by launching a small number of simultaneous requests and measuring the service's response time, then performing further tests with increasing numbers of simultaneous clients. The experiment stops when the response time has increased by a user-configured threshold. This prevents the experiment from having a detrimental effect on the real users of the target service.

Before conducting a series of experiments, the controller crawls the target server and classifies objects by size and type in order to select appropriate requests for the different resource tests. It also measures the round-trip response time for each client; when conducting tests, it compensates for the difference between clients so that the target server receives all of the requests within the shortest possible time interval. The service was used to test some "cooperating" target sites, whose administrators were aware of the tests and made their server logs available to the testers. These tests were conducted with a 250-ms response time threshold and the results were provided to the service operators; in some cases the results exposed some unexpected limitations and helped to diagnose known problems. Tests with a lower response time threshold (100 ms) were conducted on a number of other public Web sites in the wild. The results of these tests were categorized according to Quantcast popularity rank, which showed that the more popular sites tend to be better provisioned and accommodate bigger client loads but that even unpopular servers often have well-provisioned network connectivity. A survey of phishing sites showed that their request handling capabilities are similar to low-end Web sites (ranked 100,000–1,000,000).

In response to a questioner, Ramamurthy said that the MFC source code will be made available. Another attendee expressed curiosity about the response time curve beyond the 100-ms threshold. Ramamurthy offered that the relevance of larger response times depends on the type of application; for example, longer response times are more important for a search index than for a binary download site. Another attendee suggested that the tests cannot be considered "nonintrusive" if they affect the target service's response time enough to be worth measuring. Ramamurthy replied that the response time increases only for the short time that the test is being conducted and that 100 ms is a relatively small impact for testing servers in the wild; in effect, the choice of response time threshold is a compromise between nonintrusiveness and the likelihood that the results will be indicative of critical resource constraints. Another questioner addressed the problem of treating Web servers as "black boxes": The profiler might be measuring the performance of a load balancer more than that of the back-end servers. Ramamurthy agreed and mentioned that different types of tests can be developed to make more fine-grained inferences in the case of a "cooperating" server.

- ***A Dollar from 15 Cents: Cross-Platform Management for Internet Services***
  Christopher Stewart, University of Rochester; Terence Kelly and Alex Zhang, Hewlett-Packard Labs; Kai Shen, University of Rochester

Internet services are becoming more popular, and the datacenters that support them are becoming more complex. The use of multiple hardware and software platforms in a datacenter is commonplace. Multi-platform management can allow high performance at low cost, but choices tend to be made on an ad hoc basis because there are too many permutations of configurations to test exhaustively. Christopher Stewart presented an approach to optimizing performance using a predictive model which can be calibrated with readily available data and used to guide server purchasing decisions and make the best use of multiple platforms in a heterogeneous environment.

Often, management recommendations must be made without modifying production systems in any way; it is impossible to obtain profiling information using source code instrumentation and controlled benchmarking. Therefore, Stewart's approach relies only on data that is readily available without touching production systems. It uses trait models derived from empirical observations of production systems, together with expert knowledge of the structure of processors and Internet services. The key principle is to derive trait models from production data for hard-to-characterize platform parameters and to use expert knowledge to compose traits for performance prediction. A trait model characterizes only one aspect of a complex system: For example, a processor metric such as cache misses can be predicted from a system configuration variable such as cache size.

The effectiveness of Stewart's method was demonstrated by calibrating a trait model on one processor and using it to predict application performance characteristics on a system with a different processor. The calibrations and predictions were made for three different applications. The model offered superior accuracy over a wide range of request mixes, compared to commonly used predictors such as benchmarks and processor clock speed. As well as service time, it was able to make accurate predictions of total response time, using a previously developed queueing model which can be calibrated in production environments. Stewart discussed potential management applications, including platform-aware load balancing, in which distributing requests to the platform best configured to their architectural demands may yield better performance than the typical weighted round-robin approach.

One attendee asked whether the model's predictions were accurate for future performance as well as past performance. Stewart explained that his method was to use the first half of a month's data to calibrate a model, then compare the resulting prediction against the data from the second half of the month. He also mentioned that the predictions were tested against the first half of the month, with favorable results, although that data was not included in the paper. Another attendee wondered whether the method would suffer from the introduction of new architectures, because of the need to develop new empirical observations and not having suitable models on hand. Stewart observed that the trait models are attractive because they can be constructed cheaply; developing new models for new platforms can be done quickly enough. Stewart also clarified that the queue model refers to the application-level queue—users waiting for responses—not the operating system's run queue.

INVITED TALK

- ***Xen and the Art of Virtualization Revisited***
  Ian Pratt, University of Cambridge Computer Laboratory

  Summarized by Ward Vandewege (ward@gnu.org)

The Xen project mission is to build the industry standard open source hypervisor. To maintain Xen's industry-leading performance, Xen tries to be first to exploit new hardware acceleration features and helps operating system vendors to paravirtualize their operating systems. Security is paramount to maintaining Xen's reputation for stability and quality. Xen supports multiple CPU types (e.g., x86, ia64, PowerPC, and ARM, with more to come). With its roots as a university project, Xen wants to foster innovation and drive interoperability between Xen and other hypervisors.

Virtualization is hot for a number of reasons. Virtualization allows clearing up the mess created by the success of "scale-out" caused by moving applications from big iron to x86: the so-called server sprawl with one application per commodity x86 server, leading to 5%–15% typical CPU utilization. This is a result of the failure of popular OSes to provide full configuration isolation, temporal isolation for performance predictability, strong spatial isolation for security and reliability, and true backward application compatibility. With virtualization, old applications can be run on old OSes instead of relying on less than perfect OS backwards compatibility.

The first virtualization benefits are server consolidation, manageability, ease of deployment, and virtual machine (VM) image portability. Second-generation benefits include avoiding planned downtime with VM relocation, dynamically rebalancing workloads to meet application SLAs or to save power, automated systems that monitor hosts and VMs to keep apps running, and "hardware fault tolerance" with deterministic replay or checkpointing.

Security of the hypervisor code is obviously very important, but hypervisors can also improve security in a number of ways. Hypervisors allow administrative policy enforcement from outside the OS—for instance: firewalls, IDS, malware scanning, all running outside of the Xen domU. OSes can also be hardened with immutable memory. The hypervisor also shields the OS from hardware complexity by abstracting away the complicated real world with multi-path IO, high availability, etc. Breaking the bond between the OS and hardware simplifies application-stack certification: Application-on-OS, OS-on-hypervisor, and hypervisor-on-hardware can all be certified more easily, which enables virtual appliances. Virtual hardware also greatly reduces the effort to modify or create new OSes. This opens the door to application-specific OSes, the slimming down and optimizing of existing OSes, and native execution of applications. Finally, hypervisors enable hardware vendors to "light up" new features more rapidly.

Paravirtualization means extending the OS so it is aware that it is running in a virtualized environment. This is important for performance, and it can work alongside hardware enhancements found in modern CPUs.

Memory management unit (MMU) virtualization is critical for performance. It is challenging to make it fast, though, especially on SMP. Xen supports three MMU virtualization modes: direct pagetables, virtual pagetables, and hardware-assisted paging. OS paravirtualization is compulsory for direct pagetables and is optional but very beneficial for virtual and hardware-assisted paging.

Network interface virtualization is tough to achieve. In addition to the high packet rate with small batches, data must typically be copied to the virtual machine when received, and some applications are latency-sensitive. Xen's network IO virtualization has evolved over time to take advantage of new NIC features. Xen categorizes smart NICs in levels 0 through 3. Level 0 NICs are conventional server NICs, whereas level 3 ones are more exotic, with very advanced features. Smarter NICs reduce CPU overhead substantially, but care must be taken that by using smarter NICs the benefits of VM portability and live relocation are not lost.

Xen Client is a frontier for virtualization: a hypervisor for client devices. Hypervisors on small computer systems will allow "embedded IT" virtual appliances that could run intrusion detection systems, malware detection, remote access, backups, etc., independent of the user-facing operating system.

To conclude: open source software is a great way to get impact from university research projects. Hypervisors will become ubiquitous, offering near-zero overhead and being built into the hardware. Virtualization may enable a new "golden age" of OS diversity, and it is a really fun area to be working in!

WORKLOADS AND BENCHMARKS

Summarized by Kiran-Kumar Muniswamy-Reddy (kiran@eecs.harvard.edu)

■ ***Measurement and Analysis of Large-Scale Network File System Workloads***
Andrew W. Leung, University of California, Santa Cruz; Shankar Pasupathy and Garth Goodson, NetApp, Inc.; Ethan L. Miller, University of California, Santa Cruz

Andrew Leung presented results from a three-month study of two large-scale CIFS servers at NetApp, the first trace study that analyzes CIFS servers. One server had a total storage of 3 TB, with most of it used, and it was deployed in a corporate datacenter. The other server had a total storage of 28 TB, with 19 TB used, and it was deployed in an engineering datacenter.

Andrew highlighted some of the interesting findings in the study. They found that more than 90% of active data is untouched during the three-month period. The read/write byte ratio was 2:1, whereas it was 4:1 in past studies. The number of requests is high during day and low during the night (as expected). Read/write access patterns have increased (as workloads have become more write-oriented). Some 64% of all files are opened only once and 94% of files are opened fewer than five times, with 50% of reopens happening within 200 ms of the previous open. Files are infrequently accessed by more than one client. Even when they are accessed by more than one client, file sharing is rarely concurrent and they are mostly read-only.

One member from the audience asked whether they analyzed how file sizes grew over time. Andrew replied that they did not analyze this but a significant amount of the data came in single open/close sets. He then asked about the average data transfer rate. Andrew replied that the access patterns varied a lot from one day to the next and it is hard to put down a number. In response to a question about the size of the system they studied, Andrew replied that he would call it a medium system. The Q&A session ended with a member of the audience commenting that the results should be fed back to the spec benchmarks.

■ ***Evaluating Distributed Systems: Does Background Traffic Matter?***
Kashi Venkatesh Vishwanath and Amin Vahdat, University of California, San Diego

Kashi Vishwanath posed the question, "What sort of background traffic should be used while evaluating distributed systems?" To answer this, they performed a literature survey of 35 papers from SIGCOMM, SOSP/OSDI, and NSDI from 2004 to 2007. They found that 25% of the papers did not use any background traffic to evaluate their system, 15% used simple models (constant bit rate or Poisson models) to model their background traffic, 33% employed live deployments for their measurements, and 25% used complex models for their measurements. Using their test setup, they first compared simple models for generating background traffic and swing to ascertain whether their traffic generator was responsive and realistic. They concluded that simple methods can result in significant inaccuracy and that you need traffic generators that are more realistic. Further, they evaluated the effect of background traffic on various classes of applications. They found that Web traffic (HTTP) is sensitive to the burstiness of background traffic, depending on the size of the objects being transferred. Multimedia apps are not very sensitive to traffic burstiness, as they are designed to tolerate some jitter. Bandwidth estimation tools are highly sensitive to bursty traffic. Based on these results, they concluded that applications should be evaluated with background traffic with a range of characteristics.

Someone from the audience asked whether they went back and tried to evaluate how their findings would affect the results from the papers in their literature survey. Kashi replied that they did do that and found that some applications changed quite a bit with the amount of background traffic.

■ ***Cutting Corners: Workbench Automation for Server Benchmarking***
Piyush Shivam, Sun Microsystems; Varun Marupadi, Jeff Chase, Thileepan Subramaniam, and Shivnath Babu, Duke University

Piyush Shivam presented this paper. Their goal was to devise a workbench controller that plans the set of experiments to be run based on some policy, acquires resources and runs the experiments, and further plans the next set of experiments to be run based on the results. The challenge is to do this efficiently (i.e., running as few experiments as possible) while achieving statistical significance. As an example they use finding the peak rate on a Linux NFS server and present various algorithms and policies for doing this (strawman linear search, search, binary search, linear, and model guided). Their results show that their automated workbench controller achieves their goals at lower cost than scripted approaches that are normally used.

A member of the audience commented that using the peak load is misleading and that the median case is more important. He then asked whether they tried varying the workload mix. Piyush replied that the peak was just an example they used in the paper and that you could try varying the workload mix. Next, Piyush was asked what happens when the parameter space explodes. Piyush replied that the response surface method lets you choose only 2% of the overall possible space.

SECURITY AND BUGS

Summarized by Kiran-Kumar Muniswamy-Reddy (kiran@eecs.harvard.edu)

- ***Vx32: Lightweight User-level Sandboxing on the x86***
  Bryan Ford and Russ Cox, Massachusetts Institute of Technology

  Awarded Best Student Paper!

Russ Cox presented Vx32, a lightweight sandbox for the x86 architecture. Vx32 is not OS- or language-specific, but it is tied to the x86 architecture. Most x86 OSes don't use all segments, and users can create their own segments. Vx32 takes advantage of this and runs the code to be sandboxed natively in its own segment. But the sandboxed code can change the segment registers. Vx32 prevents this by using dynamic instruction translation and rewriting code to a "safe" form. They evaluated Vx32 by running various benchmarks and by building four applications. For benchmarks, the overheads are low when there are no indirect branches (i.e., no instructions to be translated). The applications that they built were an archival storage system, an extensible public-key infrastructure, a port of the Plan 9 OS on top of a commodity operating system, and a Linux system call jail. The first two applications have between 30% slowdown to 30% speedup compared to native execution. Linux jail has an 80% overhead.

A member of audience asked what they planned to do about 64-bit systems as they do not have segmentation registers. Russ replied that they can switch to a 32-bit mode while running Vx32's 32-bit code segments. Next, Russ was asked whether Vx32 lives in the same segment as the code being sandboxed. If so, could self-modifying code attack it? Russ replied that the translated code lives in a different segment than Vx32. Lastly, Russ was asked how Vx32 was different from a binary instrumentation tool such as Pin. He replied that Vx32 is much faster than in Pin; you can either get performance or safety but not both.

- ***LeakSurvivor: Towards Safely Tolerating Memory Leaks for Garbage-Collected Languages***
  Yan Tang, Qi Gao, and Feng Qin, The Ohio State University

Memory leaks can occur even in garbage-collected languages such as Java and C#. One reason is that programs keep pointers to objects they don't use anymore. For long-running programs, this results in performance degradation as they take up more and more heap space and eventually crash the program. Their system, LeakSurvivor, identifies such "potentially leaked" (PL) objects and swaps them out from both virtual and physical memory. They replace references to PL with a unique kernel reserved address. Access to these addresses will result in a swap-in. They also maintain an index that keeps track of all outgoing pointers to an object. They implemented LeakSurvivor on top of Jikes RVM 2.4.2. They evaluated their system by running it with programs that had known memory leaks (Eclipse, Specjbb2000, and Jigsaw). Eclipse and Specjbb survive with good performance for much longer than they do without LeakSurvior. Jigsaw, even though it runs for much longer with LeakSurvior, eventually crashes because their leak detector could not detect "semantic leaks" present in Jigsaw. The overhead of LeakSurvivor is low (2.5%) when it is running programs that don't have leaks.

In response to whether they can meet QoS guarantees when they run LeakSurvivor on a Web server, the authors replied that they currently cannot make performance guarantees. As to whether they have to save virtual memory for a 64-bit machine, the authors explained that, in a 64-bit machine, you have infinite virtual memory and LeakSurvivor might hurt performance. When asked whether they have any plans for providing feedback to developers so that developers can fix their leaks, they admitted that they currently did not have this functionality. As to whether they had any heuristics for turning LeakSurvivor on and off, the authors replied that they currently turn it on all the time, but it is not really hard to add this function.

- ***Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing***
  Dan Wendlandt, David G. Andersen, and Adrian Perrig, Carnegie Mellon University

Dan Wendlandt presented a method to reduce the vulnerability to man-in-the-middle (MITM) attacks of some of the common protocols such as SSH and HTTPS. SSH's model of host authentication is one of "trust-on-first-use," in which the user decides whether an unauthenticated key is valid or not. This and the fact that the user must manually verify the validity of any key that conflicts with a cached key make the user very vulnerable to MITM attacks. The Perspectives approach to mitigate this is to have a bunch of notaries in the network. Instead of trusting the SSH key, a client can verify the key from the notaries. The notaries probe machines on the network and build a record of the keys used by the services over a period of time.

The notaries provide the client with spatial redundancy (observation from multiple vantage points) and temporal redundancy (observation over time). The notaries offer a better perspective to the clients and enable them to make better security decisions. Further, the client implements key-trust policies that trade off between security and availability; for example, it might accept a key even when the number of notaries that report a key is less than a quorum.

Someone asked how a client can know how many notaries are present. Dan replied that the clients can download a notary list, but he also pointed out that someone accessing a server that has just been deployed will not get temporal security. Next, Dan was asked whether Perspectives would help with the Debian bug. Dan replied that Perspectives will not help you detect bugs in the OpenSSH implementation.

- ***Spectator: Detection and Containment of JavaScript Worms***
  Benjamin Livshits and Weidong Cui, Microsoft Research

Benjamin Livshits proposed a distributed taint mechanism for detecting and containing Javascript worms. Javascript worms are hard to find and fix, as Web 2.0 technologies allow the worms to propagate themselves by generating appropriate HTTP requests. Simple signature-based solutions are insufficient, as worms are polymorphic. Their idea for detecting worms is as follows. They tag each page uploaded on the server. This tag is downloaded to clients whenever the Web page is downloaded. They also inject Javascript code so that the tags are propagated at the client side and are preserved when pages are updated. They look for worms by checking for long propagating chains. They have an approximation algorithm that is designed to scale for graphs containing thousands of nodes. They evaluated Spectator for scalability and precision by performing a large-scale simulation of MySpace and a real-life case study (on siteframe).

YuanYuan Zhou asked whether the tag should be unique or whether it can be global. Benjamin replied that it really is not an issue and that it can be global. YuanYuan then asked if the tags can be removed by the worms. Benjamin replied that they generally cannot be removed, as the tags are HTML, but there are some particular cases of worms where it can be a problem.

INVITED TALK

Summarized by Zoe Sebepou (sebepou@ics.forth.gr)

- ***Using Hadoop for Webscale Computing***
  Ajay Anand, Yahoo!

Ajay Anand described their experiences using Apache Hadoop and what led them to start developing this product. He started his talk by stating the problem Yahoo! has in collecting huge amounts of data, implying petabytes of storage capacity and a vast number of machines to deal with the processing of this data in a secure and accurate manner, while avoiding hardware outages.

Hadoop constitutes an open source implementation of a Distributed File System (HDFS) and a map-reduce programming model combined in one package. Hadoop is designed to support many different applications providing them with the required scalability and reliability, which otherwise would be extremely costly to implement in each application. Hadoop is written in Java so it does not require any specific platform. Its main components are a Distributed File System based on the architectural characteristics of the Google File System (GFS) and a Distributed Processing Framework based on the map-reduce paradigm.

Hadoop architectural characteristics include many unreliable commodity servers and one single metadata server, but it ensures reliability by replicating the data across the available data servers. Because the system was designed for the requirements of their environment and in general for Web-scale applications that make simple sequential access involving one writer at a time and as a consequence do not require strict locking features, Hadoop receives performance advantages from the simplicity of its design. Indeed, the core design principle behind Hadoop is to move the computation as close to the data as possible; processing data locally is definitely more effective than moving the data around the network.

HDFS, which is Hadoop's file system, operates using two main components: The name nodes keep information about the files (name, number of replicas, and block location); the data nodes provide the actual storage of the data. The files in HDFS are striped across the available data servers and are being replicated by a settable replication factor to avoid unavailability resulting from node failures. HDFS keeps checksums of the data for corruption detection and recovery. Every time someone requires access to a specific file, it contacts the name nodes and, after obtaining information about the exact location of the data, it directly acquires the data from the data nodes. In case of a data-node failure, the name node detects it by periodically sending heartbeats to the data nodes. After a failure, the name node chooses a new data node to store new replicas. With the use of checksums, the clients can identify data corrupted by a node outage and ask some other available data node to serve their request. However, name-node outage still remains a single point of failure.

Ajay continued his talk by analyzing the map-reduce technique used by Hadoop to enhance the system's performance by providing efficient data streaming by reducing seeks. The map-reduce mechanism follows a master-slave architecture. Specifically, the master, called Jobtracker, is responsible for accepting the map-reduce jobs submitted by users, assigns map-reduce tasks to the slaves, called Tasktrackers, and monitors the tasks and the Tasktrackers' status in order to reexecute tasks upon failure. The Tasktrackers run map-reduce tasks upon instruction from the Jobtracker and manage the storage and transmission of intermediate outputs. Ajay pointed out that some future improvements are still to be made in the map-reduce mechanism; Yahoo! is currently working on these issues. He explained that Hadoop still does not have an advanced scheduling system. The slaves of the map-reduce framework can manage one or more jobs running within a set of machines and the mechanism does work well for dedicated applications; however, in the presence of shared resources their mechanism would not be sufficient. Consequently, he described the Pig programming environment, an Apache incubator project initiated by Yahoo!. Pig is a high-level, easy-to-use dataflow language used to generate map-reduce jobs and provides extensible data processing primitives.

Ajay concluded his presentation with the current uses of Hadoop inside and outside the Yahoo! environment, also providing measurements depicting the advantages gained by using the system.

Bar Kenneth from VMware asked whether Yahoo! had considered exploring the use of virtual machines to solve problems with loss of data locality in Hadoop. The reply was that in fact virtualization is an issue they are very interested in and that they will be exploring this possibility. Moreover, their goal is to be able in the future to say that the *job* is the VM and what they actually want is to be able to replicate the jobs across the machines.

Rik Farrow wondered, if the name nodes are really critical for Hadoop, why there is no high availability for them and why they have yet to develop a mechanism to support this feature. Ajay answered that this issue is on the list of their things to do but is not at the top because most of what they are running are batch jobs and not online operations. In addition, their main priority is to enhance other things such as the scheduling mechanism to provide name-node balancing.

A second question from Rik Farrow was whether Hadoop has a shared memory architecture. The answer was that it does not. In fact, each computer node has its own memory and this memory is not shared across machines.

A questioner from Sun Microsystems asked about the algorithms used for chunking and data distribution, as well as for the fault-tolerance mechanism and the load balancing of the data placement in Hadoop. Ajay explained that the basic concept is to have three replicas, two within a rack and one outside, to spread things around as much as possible inside their environment. The same questioner asked about the communication protocol between the HDFS clients and the name nodes of Hadoop, wondering whether there is a separate path for the metadata communication and the heartbeat messages. The reply was negative; in Hadoop all the communication is taking place through the same network, without any isolated network for metadata purposes.

Another questioner asked about bottlenecks in the network bandwidth, the disk bandwidth, or the CPU utilization of their system. The speaker said that at Yahoo! they try to collect data and to do more and more profiling to identify the bottlenecks. The main bottlenecks already observed are the network and the memory in the name-node side.

In response to an additional question about how Hadoop handles a global failure and how things return to normal again, Ajay replied that Hadoop continues working in the case of node failures as long as they are not name nodes. To the final question of how many times and how often they have to upgrade their system, the answer was that in the case of upgrade everything has to come down; usually they upgrade the system once a month, with the whole process taking less than four hours.

MEMORY AND BUFFER MANAGEMENT

Summarized by Varun Marupadi

- ***A Compacting Real-Time Memory Management System***
  Silviu S. Craciunas, Christoph M. Kirsch, Hannes Payer, Ana Sokolova, Horst Stadler, and Robert Staudinger, University of Salzburg

Modern memory managers lack predictability—the time to allocate a chunk is dependent on the global memory state. In addition, fragmentation of memory is not dealt with well. To address these shortcomings, Silviu Craciunus and his co-authors have developed Compact-fit, a memory management system that responds in linear time to the size of the request and is able to trade off performance for lower levels of fragmentation.

The system works by dividing objects into differently sized "classes." Within each size class, there is allowed to be only one partially filled page. This allows quick (linear time) deallocation, since exactly one object needs to be moved per deallocation. The authors present two implementations—one actually moves data in physical memory when an object is freed (the "moving implementation") and the other manages an indirection table that allows only table information to be changed without moving the data itself. In either implementation, by increasing the number of pages that may be partially filled, some performance may be gained at the expense of more fragmentation.

Experimental evaluation shows that allocation and deallocation times show good fidelity with the theoretical predictions, but they are slower than existing memory allocators, owing to the overhead of managing fragmentation. Compact-fit is able to allocate objects effectively even with high levels of fragmentation. In response to a question, the authors said that Compact-fit will not move objects from one size class to another at the current time.

- ***Prefetching with Adaptive Cache Culling for Striped Disk Arrays***
  Sung Hoon Baek and Kyu Ho Park, Korea Advanced Institute of Science and Technology

Sung Hoon Baek and Kyu Ho Park study the neglected field of prefetching schemes for striped disk arrays. Prefetching from striped disks has several new problems, including loss of expected parallelism owing to short reads, nonsequential short reads, and the absence of cache management for prefetched data.

To manage these risks, the authors present Adaptive Stripe Prefetching (ASP), which uses new schemes for prefetching an entire stripe when a request for a block comes in, adaptive culling of the cache to preferentially evict prefetched blocks that have not been requested, and an online model to tune the ratio of prefetched to cached blocks to maximize the total hit rate.

The system was evaluated with a variety of benchmarks. It performs as well or better than any existing prefetching schemes. A question was raised regarding the performance of the system in the presence of writes. The response was that the system is primarily focused on read-heavy workloads but should work in the presence of writes as well. It was also pointed out that one of the benchmarks (Dbench) simulates a read-write workload.

- ***Context-Aware Prefetching at the Storage Server***
  Gokul Soundararajan, Madalin Mihailescu, and Cristiana Amza, University of Toronto

A problem with today's prefetching schemes is that they break down under high levels of concurrency because it is hard to detect access patterns when requests from many sources are interleaved. To address this, Gokul Soundararajan and his colleagues presented QuickMine, a system that allows application contexts to be visible to the storage server so that it can more accurately detect access patterns.

Every block request is tagged with an identifier corresponding to a higher-level application context (Web, database, or application). It is claimed that this is minimally intrusive and easy to create for any application. However, it does require minor modifications to the source code. Mining the context-tagged requests can generate block correlations for both sequential and nonsequential accesses. The system was evaluated by modifying the MySql database to pass context information and running a number of three-tier Web-based applications on it. For all benchmarks, the cache miss rate and latency were drastically reduced by using QuickMine.

In a lively question session, several attendees asked about extending the work to other contexts. In particular, file-based storage rather than block-based storage could be dealt with by using the filename+offset rather than the block number. Extension to other applications requires only localized instrumentation changes. Extension to other classes of applications would be more intrusive and is a topic of ongoing research. Schemes using fuzzy contexts or machine learning techniques to infer context could be used and are worth exploring, but Gokul believes context is still necessary, because very different queries follow the same code path through libraries.

INVITED TALK

Summarized by Matthew Sacks (matthew@matthewsacks.com)

- ***Google Hacking: Making Competitive Intelligence Work for You***
  Tom Bowers

Tom Bowers takes the ideas presented in Johnny Long's book *Google Hacking* and applies the concepts of using Google as a hacking utility for servers and other machine-related vulnerabilities to information in and of itself. The amount of information that can be gathered using the world's largest database is astounding. Tom went on to demonstrate how to gather information about a particular organization or individual by leveraging unconventional techniques for using the Google search engine.

By using Google as a utility for competitive intelligence, one can find out a wealth of information about competitors, as well as seeing what type of information is being leaked about the individuals in a company or organization and the organization itself. 80% of all competitive intelligence is done through public sources. Also, the U.S. Supreme court has ruled that information found on Google is public information.

Tom also presented the basic method for performing competitive intelligence using Google by building a competitive intelligence profile.

As an example, using Google Earth Pro (which provides more frequent updates than the standard Google Earth), Tom can map out a competitor's facility to determine where he might be able to gain easy access. From there he could use wireless scanning techniques to access the competitor's data from unsecured wireless networks. Also, using Google hacking Tom showed that a large majority of Web cams are available through the public Internet from a standard Google search!

In this talk Tom revealed the world of competitive intelligence and its primary information-gathering utility: Once a competitive profile has been built, the job of gathering additional detailed information becomes rather simple. Most of the work done in competitive intelligence can be done from one's own office or home.

WIDE-AREA SYSTEMS

Summarized by Varun Marupadi (varun@cs.duke.edu)

- ***Free Factories: Unified Infrastructure for Data Intensive Web Services***
  Alexander Wait Zaranek, Tom Clegg, Ward Vandewege, and George M. Church, Harvard University

Alexander Zaranek and his colleagues explained that this work was initiated to help process the large amounts of data needed to sequence human genomes. A free factory is a set of several 12- to 48-node clusters, some of which are co-located with data-acquisition instruments. The clusters are connected via relatively slow networks. A free factory runs *freegols*, which are application-centric virtual appliances that run within a free factory. Different users use and develop different freegols for their particular needs.

A portion of the cluster's resources is configured as warehouse instances, which provide processing, cache, and storage services. The remainder of the resources hosts Xen virtual machines for hosting freegols. The storage services within a cluster are implemented as a three-tier hierarchy:

a memory cache, a distributed block cache, and a long-term archival storage service.

More information can be found at factories.freelogy.org.

- ***Wide-Scale Data Stream Management***
  Dionysios Logothetis and Kenneth Yocum, University of California, San Diego

Dionysios Logothetis presented Mortar, a platform for building queries across federated distributed systems. Such queries are useful for remote debugging, measurement, application control, and myriad other uses. Mortar allows operators to aggregate and process data within the network itself, building multiple overlays to process data from remote sources.

Mortar builds a set of static overlay trees that overlap in order to tolerate node and network failures. By carefully building trees, it is possible to generate routes that are network-aware and resilient at the same time. Mortar avoids problems arising from static clock skew by using relative time offsets rather than absolute timestamps. By isolating data processing from data routing, it is possible to use aggregate operators that are not idempotent or duplicate-insensitive. By using multiple static overlay trees, Mortar is able to make progress when as many as 40% of the nodes have failed.

Questions were raised about how queries that require knowing the source of the data could be implemented. Dionysios replied that such queries are problematic because of the nature of aggregation itself. Other attendees wondered whether the system might fail from corner cases in the heuristics and static tree-based routing. Dionysios explained that the effect of topology on the system has not yet been fully studied, so it is hard to give a definite answer.

- ***Experiences with Client-based Speculative Remote Display***
  John R. Lange and Peter A. Dinda, Northwestern University; Samuel Rossoff, University of Victoria

John Lange presented work on speculatively executing window events on a remote display. The goal is to reduce the user-perceived latency when using a remote service. The predictability of events sent by VNC and Windows Remote Desktop was presented; VNC appeared to be much more predictable than RDP. John says that this may be primarily due to the higher level of abstraction that RDP uses, along with the much lower event rate. A Markov model was used to predict future events based on past events and user input. This also allowed control over the tradeoff between accuracy and latency.

A user study was presented for VNC prediction. Although not conclusive, the study did show that users are at least moderately accepting of display errors during misprediction. A question was asked about what constitutes an error. John explained that an error may be anything from garbage on the screen to subtle artifacts in the window. Another attendee asked about overhead. John replied that, after training, there was almost no CPU overhead but there was some memory overhead.

Third Workshop on Hot Topics in Autonomic Computing (HotAC III)

Wheeling, IL
June 2, 2008

Summarized by Alva Couch, Tufts University

The theme of this year's Hot Autonomic Computing (HotAC) was "grand challenges of autonomic computing." By contrast with two prior iterations of HotAC involving papers and panels, this year's HotAC included short presentations, working groups, and plenty of discussion.

In the morning, selected attendees were given five minutes each to describe a grand challenge problem in autonomic computing, how to solve it, and what resources would be required. Presenters were selected based upon white papers submitted to the conference organizers in advance. In the presentations, several themes emerged, including monitoring, composition, applications, and human concerns.

Autonomic systems remain difficult to monitor and the monitored data remains incomplete. Autonomic system state remains difficult to characterize and more accurate models are needed (Salim Hariri, University of Arizona). There is a need for "adaptive monitoring" that tracks changing needs (Paul Ward, University of Waterloo), as well as "experiment-based" control based upon making changes and observing results (Shivnath Babu, Duke University). The resulting monitoring infrastructure must be scalable and adaptable to a changing Internet (Fabián Bustamante, Northwestern University).

It also remains unclear how to compose different control systems to control one entity, and how to deal with openness and unexpected events. It remains difficult to compose or combine autonomic systems (Alva Couch, Tufts University) and to deal with unpredictable behavior. An ideal autonomic system might employ scalable co-ordinated cross-layer management (Vanish Talwar, HP) in which control systems are composed vertically from lower-level elements.

Several application domains for autonomic computing were explored. Empathic autonomic systems (Peter Dinda, Northwestern University) optimize for perceived end-user satisfaction. Spatial computing (Jake Beal, MIT CSAIL) requires new languages and abstractions to control a computing medium in which computing presence approximates a continuous medium. Autonomics can help us construct "Green IT" computing environments (Milan Milankovic, Intel) that exhibit reduced energy consumption, lower carbon footprint, etc. Sensor networks can be managed through a holistic strategy that treats the whole network as a single entity (Simon Dobson, UC Dublin). P2P networks can benefit from "sloppy" autonomic control mechanisms that "leave well enough alone" and only react when basic objectives are not met (Guillaume Pierre, VU Amsterdam).

Autonomic systems must also enforce human objectives (Jeffrey Kephart, IBM). Human goals can conflict and require competitive—and not simply cooperative—strategies (Ivana Dusparic, TCD). Human trust of autonomic systems remains a key problem (John Wilkes, HP).

At the end of the morning's discussion, the group voted to study three issues in detail:

- Single self-adaptive system challenges: monitoring and modeling
- Multiple self-adaptive system challenges: composition and openness
- Goals, objectives, and trust: the human side of autonomics

A working group was convened to study each problem. Each working group met in the afternoon and presented a report; these are briefly summarized next.

SINGLE SELF-ADAPTIVE SYSTEMS

Single self-adaptive systems can now be built, but systematic methods should be developed for building these systems. Systematic methods require good models for prediction, control, error detection/fault diagnosis, and optimization. Models must describe behavior at different time and detail scales, for different tasks (e.g., energy, error detection) and for different degrees of accuracy. Models can be self-learned or provided by expert human engineers. Models should describe both the system and its environment. Objectives need to be clearly defined for accountability, performance, and reliability of self-adaptive systems.

MULTIPLE SELF-ADAPTIVE SYSTEMS

Multiple self-adaptive systems might include systems composed of equipment and software from several vendors, with limited knowledge of one another, and different administrative domains and management objectives. These objectives can potentially conflict with regard to performance, availability, energy efficiency, security, reliability, resource usage, and resilience. Potential problems include independent control systems trying to control the same actuator, indirect coupling through resource shortages, conflicting policies for interacting controllers, and invalidated models resulting from unforeseen interaction. Fully understanding the problem space is in itself a research issue.

GOALS, OBJECTIVES, AND TRUST

At the root of the trust issue for autonomic systems is that users do not know what they want, nor can they write it down. Requirements come from users with differing roles, information needs, and objectives. One potential mechanism for specifying needs is for users to say what they do not like and incrementally refine policy based upon interactions. Even so, requirements are expected to be incomplete and inconsistent. Possible techniques for coping with this situation include discovering and reporting conflicts ("asking for help") and exploring "what if" scenarios with the user. To ensure trust, systems can be constrained, can actively reassure users, and can explain their actions.

For more details on the discussions and outcomes of the workshop, please see http://www.aqualab.cs.northwestern.edu/HotACIII/program.html.

Findings from the First Annual Storage and File Systems Benchmarking Workshop

University of California, Santa Cruz
May 19, 2008

Summarized by Avishay Traeger and Erez Zadok, Stony Brook University; Ethan L. Miller and Darrell D.E. Long, University of California, Santa Cruz

A growing consensus in the community of file and storage system researchers and practitioners is that the quality of benchmarking must be improved significantly. We have found that there is often too little scientific methodology or statistical rigor behind current benchmarking, which is largely done ad hoc. In response, with the goal of improving the quality of performance evaluation in the field, we held the Storage and File Systems Benchmarking Workshop on May 19, 2008, at the University of California, Santa Cruz. It was sponsored by the Storage Systems Research Center (SSRC, www.ssrc.ucsc.edu).

This workshop brought together top researchers and practitioners from industry and academia, representing all levels of the storage stack, along with statisticians and other interested parties. The main goals of the workshop were to educate everyone on the problems at hand and to discuss possible solutions. Participants presented relevant topics, and there was much interaction and discussion.

The goal of this effort is improving the scientific and statistical methodologies used. This goal requires little research in the field, but it does require educating both those who conduct performance evaluations and those who analyze results. It also requires program committees and reviewers to raise the bar on the quality of performance evaluations in accepted papers. A longer-term goal is to have computer scientists embrace the rigor of the other sciences. It is essential to be able to validate the results of others. Without it, it is meaningless to compare the performance of two systems. All presentations and slides are available at www.ssrc.ucsc.edu/wikis/ssrc/BenchmarkingWorkshop08/.

WHY FILE AND STORAGE SYSTEM BENCHMARKING IS DIFFICULT

Erez Zadok, the workshop's chair, began with an overview of the storage stack, highlighting some complexities that make benchmarking these systems a difficult task and providing some examples of poor benchmarking practices. Some of the factors contributing to the complexity are:

- Storage variety: Storage does not consist only of a single local hard drive. Other types include Logical Volume Managers (LVMs), RAID, Network-Attached Storage (NAS), Storage Area Networks (SANs), flash, object storage, and virtualization.
- File system variety: Many types of file systems exist. Those operating on a local disk can use different data structures, logging infrastructures, and other features such as encryption or compression. Network file systems behave differently from local ones because of cache effects and network latencies. They are very common today, and distributed file systems are becoming even more prevalent.
- Operating system variety: Several operating systems exist, each with different behaviors. In addition, running OSes in virtual machines is becoming more common. Finally, even the same OS will behave very differently depending on the configuration.
- The workload: User activity and access patterns are difficult to accurately characterize and recreate.
- Asynchronous activity: Other processes and kernel threads may also interact with the storage stack and change the system's behavior.
- Caches: Operating system caches at various levels, as well as disk caches, can contain recently accessed data and metadata, which can change the behavior of the workload.

THE CURRENT STATE OF FILE AND STORAGE SYSTEM BENCHMARKING

The next presentation was from Avishay Traeger (Stony Brook University), summarizing his recent article [6]. The article surveys the benchmarks and methodologies that were used in file and storage system papers from SOSP, OSDI, FAST, and USENIX between 1999 and 2007 and included 415 benchmarks from 106 papers. He also looks at how testbeds and results were presented and suggests better benchmarking practices. Some of the findings were that approximately 47% of the papers did not specify how many runs were performed, and more than 28% of the benchmarks ran for less than one minute. In addition, only about 45% of the papers had some indication of variance (standard deviation or confidence intervals).

Current Benchmarks

The benchmarks presented at the workshop were IOzone [2] and SPECsfs [5] (both presented by Don Capps of NetApp) and FileBench [3] (developed by VMware and Sun Microsystems, presented by Spencer Shepler of Sun). In contrast to benchmarks that are generally used, these benchmarks provide important improvements. SPECsfs presents new techniques for scalable workload generation. IOzone and FileBench can create a variety of user-specified workloads, which may help to reduce the number of ad hoc benchmarks that are created and used. Ad hoc benchmarks are generally small programs that are written for in-house use. Using popular tools in favor of ad hoc micro-benchmarks can aid in reproducing and comparing results, as now only the workload specifications need to be reported, rather than the source for the entire benchmark. In addition, we would expect that the more popular tools will have fewer bugs and operate more correctly.

IOzone is a portable open-source file system benchmarking tool that can produce a wide variety of I/O and, more recently, metadata workloads. It can produce single or multiple execution threads and can even run on multiple nodes. An interesting feature of IOzone is its use of telemetry files. IOzone can replicate I/O operations based on a file containing *byte offset*, *size of transfer*, *compute delay* triplets, so that it can provide benchmark results from system call traces. IOzone has been downloaded millions of times, and it is the first result on Google when searching for "file system benchmark." Surprisingly, IOzone was not used in any of the conference papers surveyed by Traeger et al. In fact, many researchers publishing in the surveyed conferences have written their own benchmarks which produce workloads that IOzone can easily produce. We can only speculate about the reason for this phenomenon at this point, as we have no hard data, but we believe that this may be another indication of poor benchmarking practices in the file and storage system community.

SPECsfs is a file server benchmark that measures both throughput and response time. SPECsfs was originally created to test NFS servers. The latest version, SPECsfs2008, supports CIFS in addition to NFS. The major changes to the NFS portion of the benchmark since version 3.0 are updated I/O size distributions, a new operation mix, and the dropping of UDP and NFSv2 support. The CIFS portion is rather different, using a Hidden Markov Model driven by traces to generate the workload, rather than a predefined operation mix. The workloads for both NFS and CIFS are now based on data from many real customers. It is important to note that SPECsfs2008 cannot be used to compare NFS and CIFS servers.

An interesting point that was brought up is that the NFSv4 protocol depends much more on the client's behavior than previous versions. To benchmark a complete NFSv4 system, the client's behavior should be taken into account. This means that the method that SPECsfs uses for benchmarking NFSv3 systems would not be applicable to NFSv4 (since the benchmark crafts its own RPC packets). Any current benchmark that uses the POSIX interface can send requests to an NFSv4 server via a real client, thereby taking the client's behavior into account. However, it is up to users to define what constitutes an appropriate file server workload for their system; for that, configurable workload generators such as IOzone and FileBench can be used. In the future, we hope the community will define one or more standard fileserver workloads that are generally applicable and revise them periodically. Of course, additional benchmarks may be used as well to provide a clear picture of the system's performance characteristics.

At times benchmarking applications can be a very difficult task. For example, properly running up a TPC-C database benchmark is very expensive and may require several months of time to set up and run. In addition, we do not currently know how to extrapolate micro-benchmark results to reflect the performance of real applications. Therefore, we need to use macro-benchmarks, which more closely represent the applications themselves, and build a portfolio of workload-specific benchmarks. FileBench was developed as a method of accurately representing more complex file-based applications, so that the performance impact of a file system or storage layer can be properly characterized for specific workload types. It uses a synthetic workload model to accurately represent the workload and application stack, including the process model, the I/O types, synchronous I/Os, and, most importantly, the interlocking between I/Os. It also provides the framework for operating on statistical hierarchies of file system trees and high-level file system objects, including create/delete, traverse directory, and read/write.

Short-term Goals for Benchmarks

We realize that creating a perfect solution will involve much research and community involvement. However, there are steps that we can take now to make benchmarks more accurate and help facilitate comparable and reproducible results. In terms of accuracy, the benchmark should use accurate timing in measuring metrics. Eric Anderson (HP Labs) also pointed out the importance of accurate timing in issuing file system and I/O requests. It should also be a simple, easy-to-understand workload. This helps ensure accuracy and also assists in understanding the results and their implications. The benchmark should also accurately depict a real-world scenario if its goal is to do so. How to measure this accuracy, however, is an open problem. Finally, open-source benchmarks promote openness and allow more people to inspect the code for correctness. Of course, the code should not be modified, so that results remain comparable.

In terms of comparable and reproducible results, the benchmarks should have three main qualities. First, they should be scalable. Benchmarks may properly exercise the system at one point in time, but as systems become faster, the benchmark may no longer be appropriate. For example, a common benchmark is measuring the time required to compile some source code (as in the Andrew benchmark). However, source code that was used for benchmarks several years ago would fit in a modern system's cache and therefore would not adequately exercise the storage subsystem. Second, benchmarks should have few dependencies on libraries and the OS. For example, the Bonnie benchmark creates a random read pattern by utilizing the system's pseudo-random number generator. This causes the read pattern to change from system to system, which can lead to different results owing to caching, read-ahead, and disk locality. Third, it should be cheap, easy to set up, and portable, so that it can be used by a large number of people to benchmark on many systems.

TRACES

Traces are logs of operations that are collected and later replayed to generate the same workload (if done correctly). Two problems associated with traces are availability and replay method.

The availability issue is being addressed by the Storage Networking Industry Association's Input/Output Traces, Tools, and Analysis Technical Work Group (SNIA IOTTA TWG). Geoff Kuenning of Harvey Mudd College presented an overview of this working group. They have set up a repository at http://iotta.snia.org which seeks to archive traces in a single place using a uniform format with tools to process them. It also helps to clear up licensing issues for the traces. The preferred trace format is DataSeries, which was presented by Eric Anderson of HP Labs. DataSeries is designed for long-term storage (built-in checksums), is self-describing, and provides substantial analysis speedups and moderate space improvements. There are tools available to convert several other formats to DataSeries, as well as tools to analyze the trace files.

The problem of replaying traces is partly addressed by Buttress [1], a high-fidelity I/O benchmark system, which was also presented by Eric Anderson. This project demonstrates the importance of accurate issue time for I/O requests and provides a method for issuing them much more accurately than before. However, the system is very fragile, and it is easy to specify open (trace) workloads that are unachievable and get poor results. This is a difficult and important problem that will require more research.

INDUSTRY EXPERIENCES

Several attendees presented their benchmarking experiences from the industry perspective. First, VMware's Richard McDougal, Devaki Kulkarni, and Irfan Ahmad presented their experiences in benchmarking Virtual Machine (VM) environments. When benchmarking inside of a virtual machines, it is important to note that time measurements and the CPU's clock cycle counter may be distorted (generally by around 100 microseconds). This is especially true when the CPU is fully utilized; it can be mitigated by using

ESX-TOP, which gathers CPU utilization information from the host, by using the hardware's clock cycle counter rather than the virtualized one, or by timing from the host rather than from inside the VM. For benchmarking ESX servers, they noted that simple workloads will not suffice, as servers see different I/O patterns to the same volume, or I/O from a single application being split among multiple volumes. In addition, virtual file systems are often specially optimized, and so standard benchmarks are not always sufficient.

Next, Daniel Ellard from NetApp presented their experiences in benchmarking flash SSDs. Their goal is to perform measurements on a single device and to extrapolate to estimate the performance of a large array of devices. These new devices have characteristics that differ from disks. For example, flash SSDs implement quasi-file systems, have a strange layout that is striped across several devices, have nondeterministic writes, and have drastic aging effects. NetApp uses what they call micro-workload benchmarks; these lie somewhere between micro-benchmarks and macro-benchmarks in terms of complexity. They have developed a workload generator called Biscuit. The user defines tasks and these are generated by Biscuit. Biscuit also supports random variables, as well as telemetry and trace files.

The next presentation was by Jeff Fuller from Microsoft, who discussed some of the benchmarking methodologies used for Windows clients and servers. Fuller's group performs client application characterization to measure metrics that end users care about, such as high-level response time. Their application-level benchmarking allows them to use the same benchmarks on different platforms and compare user experiences across platforms. In addition, application-level workloads are more portable and realistic than lower-level ones. They also take client idle time (during which much asynchronous activity happens), as well as bursts of activity.

Finally, Eric Kustarz from Sun Microsystems discussed ZFS benchmarking experiences. As ZFS is a rather complex file system, the Sun group uses a large number of workloads to obtain a clear picture of its performance. Although they mainly use FileBench, they also use an assortment of other benchmarks, including IOzone, Bonnie, SPECsfs, and many others. They utilize various OpenSolaris tools to locate performance problems, such as Dtrace, Lockstat, fsstat, kstat, and vmstat.

BENCHMARKING GUIDELINES

The workshop included much discussion about proper benchmarking and statistical methodologies, and we compiled a set of guidelines to consider when evaluating the performance of a file or storage system.

A performance evaluation should have clear goals. We recommend posing questions that should be answered by the evaluation, and then choosing the systems, configurations, and benchmarks to answer them. The benchmarking process consists of four steps: selecting appropriate benchmarks, running the benchmarks, analyzing the results, and reporting the results.

First, hypothesize on what the results should look like, decide on the appropriate initial state of the system (contents of caches, partition locations, file system aging, etc.), and create it accurately. When choosing a benchmark, you should use it for its intended scope. For example, the Andrew benchmark should not be used as an I/O benchmark, and Postmark produces an NFS mail server workload. In addition, create new benchmarks only if existing ones do not provide the needed features or workload characteristics. Prefer to extend existing benchmark tools rather than writing new ones.

When running the benchmarks, we recommend using an automated system [4, 7] to reduce the possibility of human error and to ensure that all runs are identical. As many data points as possible should be collected so that proper statistical analysis can be performed on the results. For benchmarks with nonuniform workloads (e.g., a compile benchmark), this can be done by running the benchmark multiple times. For benchmarks with uniform workloads, such as those that perform a certain number of read operations, it may be possible to take measurements at regular intervals during a single run to increase the number of data points collected. In addition, we recommend measuring the system only when it is in steady state, by discarding any start-up and cool-down effects.

For quantities that are additive (e.g., time or bytes sent), the same estimate of the mean and standard deviation should be obtained whether many short runs or just a few long runs are conducted. If a stable workload is measured by dividing it into many smaller intervals, then the central limit theorem will typically apply, and thus the distribution of the mean will be approximately normal; therefore, a confidence interval for the mean can be easily constructed from estimated standard deviations, even if the distributions of the individual runs are not themselves normally distributed. When a run cannot be broken down into multiple subunits from identical distributions, there is no guarantee about the distribution of the mean.

The results can now be analyzed. As a first check, ensure that the distribution of the results is reasonable, and see whether the results match your expectations. If not, investigate and explain why. It can be useful to examine graphical summaries, such as histograms or cumulative distribution functions.

When reporting results, be sure to describe precisely what was done, to help others to understand the experiments and allow them to reproduce your results. This includes a complete description of the platform, the benchmark and any parameters, the source code for the system being tested, and the raw benchmark results. Of course, licensing issues may restrict the distribution of some of this information, but as much as possible should be provided. In addition, most publications limit the number of pages available, so we recommend publishing the information in an online appendix. We hope that repositories will be created for the long-term storage of such information. In addition to describing what was done, explain why the evaluation was done that way. This helps others to interpret the results.

Report the number of runs performed and include statistical measurements, such as standard deviations or confidence intervals, so that others can determine the accuracy of your results. If you get high standard deviations, it could be an indication that your distribution is multi-modal (which may suggest an unstable storage system); in that case, you might plot your data as a histogram and explain the modality. Quartiles may also be helpful in describing non-normal distributions, but you should have at least 30 data points before using quartiles. In some cases box-plots may be more suitable than histograms (generally when the number of data points is large). Note that standard confidence intervals (based on a normal approximation) are not appropriate for non-normal distributions.

SUMMARY

Many interesting and important issues were discussed at this workshop, and we hope to discuss more topics next year. These include simulators, tracing technology, aging effects, and measuring power consumption. In addition, we would like to discuss how to benchmark distributed and petabyte-scale systems, as well as virtual machine technologies. We also discovered that many are not familiar with the advanced statistical methods required to properly analyze benchmark results. We hope to discuss some of these methods as well.

Longer-term research goals were also discussed. One challenge is how to accurately scale traces so that they stay relevant for longer periods of time. This is important because a trace is collected once and used for many years. However, hardware, software, and usage patterns change rapidly, making the traces outdated almost as soon as they are captured. Other challenges include how to model an application's behavior as a workload model and how to measure the accuracy of a given model. Finally, there is a question of how to compare the results from two benchmarks where the platforms were different. The answer may lie in virtual machine technology, but how to do this accurately is an open question.

This first workshop was an important step in improving the overall quality of performance evaluations in the file and storage system community. Participants raised important issues and discussed potential solutions. We hope that researchers and practitioners will educate themselves and improve the quality of their performance evaluations. Finally, we hope that reviewers will raise the standards for performance evaluations in conference and journal publications.

We have been continuing our discussions on our mailing list, and we plan to publish a more detailed set of benchmarking guidelines in the future. We have also created a file and storage system benchmarking portal at http://fsbench.filesystems.org/. It links to a Wiki containing the agenda (including slides from the talks) and a list of attendees, subscription information for the mailing list, a Web version of the benchmarking guidelines, and other resources.

ACKNOWLEDGMENTS

We would first like to thank all of the attendees of this first workshop, whose valuable input and enthusiasm helped make it a success, especially Eric Anderson, Andrew Leung, and Tim Moore for supplying us with workshop minutes and Eric Anderson, Don Capps, and Richard McDougall for their reviews. Thanks are owed to Herbie Lee for his help with the statistical aspects of this article. Thanks also go to the Storage Systems Research Center at the University of California, Santa Cruz, for hosting and sponsoring this workshop. Some workshop organizers were sponsored in part by NSF award CCF-0621463 (HECURA).

REFERENCES

[1] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan, "Buttress: A Toolkit for Flexible and High Fidelity I/O Benchmarking," in *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST '04)*, San Francisco, CA, March 31–April 2, 2004, pp. 45–58.

[2] Don Capps, IOzone filesystem benchmark, July 2008: http://www.iozone.org/.

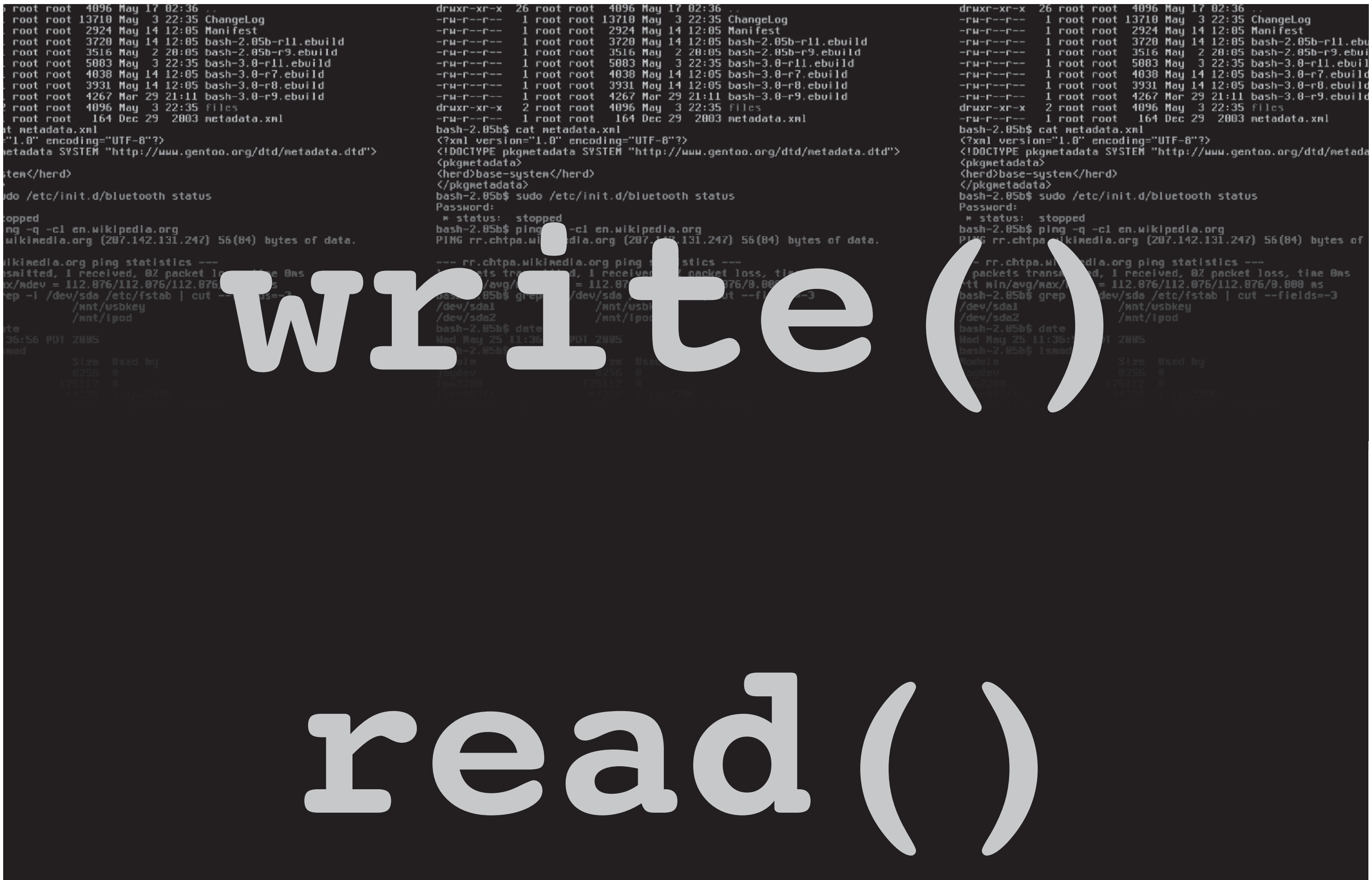
[3] FileBench, July 2008: http://www.solarisinternals.com/wiki/index.php/FileBench.

[4] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu, "Cutting Corners: Workbench Automation for Server Benchmarking," in *Proceedings of the 2008 USENIX Annual Technical Conference*, Boston, MA, pp. 241–254.

[5] SPEC, SPECsfs2008, July 2008: http://www.spec.org/sfs2008.

[6] A. Traeger, N. Joukov, C.P. Wright, and E. Zadok, "A Nine Year Study of File System and Storage Benchmarking, *ACM Transactions on Storage (TOS)*, 4(2):25-80, (2008).

[7] C.P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok, "Auto-pilot: A Platform for System Software Benchmarking," in *Proceedings of the 2005 USENIX Annual Technical Conference, FREENIX Track,* Anaheim, CA, pp. 175-187.


```
drwxr-xr-x  26 root root  4096 May 17 02:36 ..
-rw-r--r--   1 root root 13710 May  3 22:35 ChangeLog
-rw-r--r--   1 root root  2924 May 14 12:05 Manifest
-rw-r--r--   1 root root  3720 May 14 12:05 bash-2.05b-r11.ebuild
-rw-r--r--   1 root root  3516 May  2 20:05 bash-2.05b-r9.ebuild
-rw-r--r--   1 root root  5083 May  3 22:35 bash-3.0-r11.ebuild
-rw-r--r--   1 root root  4038 May 14 12:05 bash-3.0-r7.ebuild
-rw-r--r--   1 root root  3931 May 14 12:05 bash-3.0-r8.ebuild
-rw-r--r--   1 root root  4267 Mar 29 21:11 bash-3.0-r9.ebuild
drwxr-xr-x   2 root root  4096 May  3 22:35 files
-rw-r--r--   1 root root   164 Dec 29  2003 metadata.xml
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
 * status:  stopped
bash-2.05b$ ping -q -c1 en.wikipedia.org
PING rr.chtpa.wikimedia.org (207.142.131.247) 56(84) bytes of data.

--- rr.chtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 112.876/112.876/112.876/0.000 ms
bash-2.05b$ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1            /mnt/usbkey
/dev/sda2            /mnt/ipod
bash-2.05b$ date
Wed May 25 11:36:56 PDT 2005
bash-2.05b$ lsmod
```

acm queue architecting tomorrow's computing

Sign in | Create Account | Join ACM | Why Join ACM? Association for Computing Machinery

ARTICLES | INTERVIEWS | AUDIOCASTS | VIDEOS | PLANET QUEUE

HOME

DIGITAL EDITIONS
January/February 2008
Past Issues

TOPIC OF THE MONTH
Virtualization

TOPICS
Hardware
Networks
Security

COLUMNS
Curmudgeon
Geek@Home
Kode Vicious

SECTION NAME
CTO Round Table
ACM TechNews
Washington Updates
RISKS Forum

SIGN UP FOR QUEUENEWS
Our free weekly newsletter features excerpts of ACM Queue's latest articles.
your email
HTML version
Text version
SIGN UP

Get *FREE* online access to thousands of online books and courses when you join! acm

curmudgeon
Some Swans are Black
If you think that the color of swans is simply a matter of avian plumage, it could well be that there's an element missing in your programmer's toolbox
by Stan Kelly-Bootle | April 30, 2008
Tags: software, programming, networks, security, linux
3 Comments | 621 Views

article
Network Virtualization
Breaking the Performance Barrier: Shared I/O in virtualization platforms has come a long way, but performance concerns remain.
by Scott Rixner, Rice University | April 27, 2008
Tags: software, programming, networks
0 Comments | 188 Views

article
Beyond Server Consolidation
Server consolidation helps companies improve resource utilization, but virtualization can help in other ways, too.
by Werner Vogels, Amazon.com | February 16, 2008
Tags: software, programming, networks, security
7 Comments | 506 Views

interview
A Conversation with Jason Hoffman
A systems scientist looks at virtualization, scalability, and Ruby on Rails
by Bryan Cantrill | April 29, 2008
Tags: software, programming, networks
18 Comments | 3280 Views

curmudgeon
Some Swans are Black
If you think that the color of swans is simply a matter of avian plumage, it could well be that there's an element missing in your programmer's toolbox
by Stan Kelly-Bootle | April 30, 2008
Tags: software, programming, networks, security, linux
3 Comments | 621 Views

article
Network Virtualization
Breaking the Performance Barrier: Shared I/O in virtualization platforms has come a long way, but performance concerns remain.
by Scott Rixner, Rice University | April 27, 2008
Tags: software, programming, networks
0 Comments | 188 Views

article
Beyond Server Consolidation
Server consolidation helps companies improve resource utilization, but virtualization can help in other ways, too.
by Werner Vogels, Amazon.com | February 16, 2008
Tags: software, programming, networks, security
7 Comments | 506 Views

interview
A Conversation with Jason Hoffman
A systems scientist looks at virtualization, scalability, and Ruby on Rails
by Bryan Cantrill | April 29, 2008
Tags: software, programming, networks
18 Comments | 3280 Views

More Recent Content

search
browse tags

TOPIC OF THE MONTH
Virtualization
The technology isn't new, but it has matured a lot over the past 30 years. Get caught up ion the changes.
More

QUEUE FOCUSES... on the problems and challenges presented by emerging and potentially "disruptive" technologies, as well as the choices and consequences practicing professionals and their companies are expected to face as a result.

PLANET QUEUE
Your favorite Queue bloggers post about issues of interest to practitioners

dtrace.conf(8)
dtrace.conf(08) was this past Friday, and (no surprise, given the attendees), it ended up being an incredible (un)conference...
Posted on The Observation Deck | 2:23 PM, 4/18/08

Service Contracts & Global Constraints
We ran into an interesting problem with one of our web service interfaces at work today...
Posted on Rambling On Software | 1:54 PM, 4/10/08

dtrace.conf(8)
dtrace.conf(08) was this past Friday, and (no surprise, given the attendees), it ended up being an incredible (un)conference...
Posted on The Observation Deck | 2:23 PM, 4/18/08

Service Contracts & Global Constraints
We ran into an interesting problem with one of our web service interfaces at work today...
Posted on Rambling On Software | 1:54 PM, 4/10/08

More Planet Queue

About | Advisory Board | Authors | Join ACM
RSS Feeds | Site Map | Terms of Use | Privacy Policy | Contact Us
© 2008 ACM, Inc. All Rights Reserved.

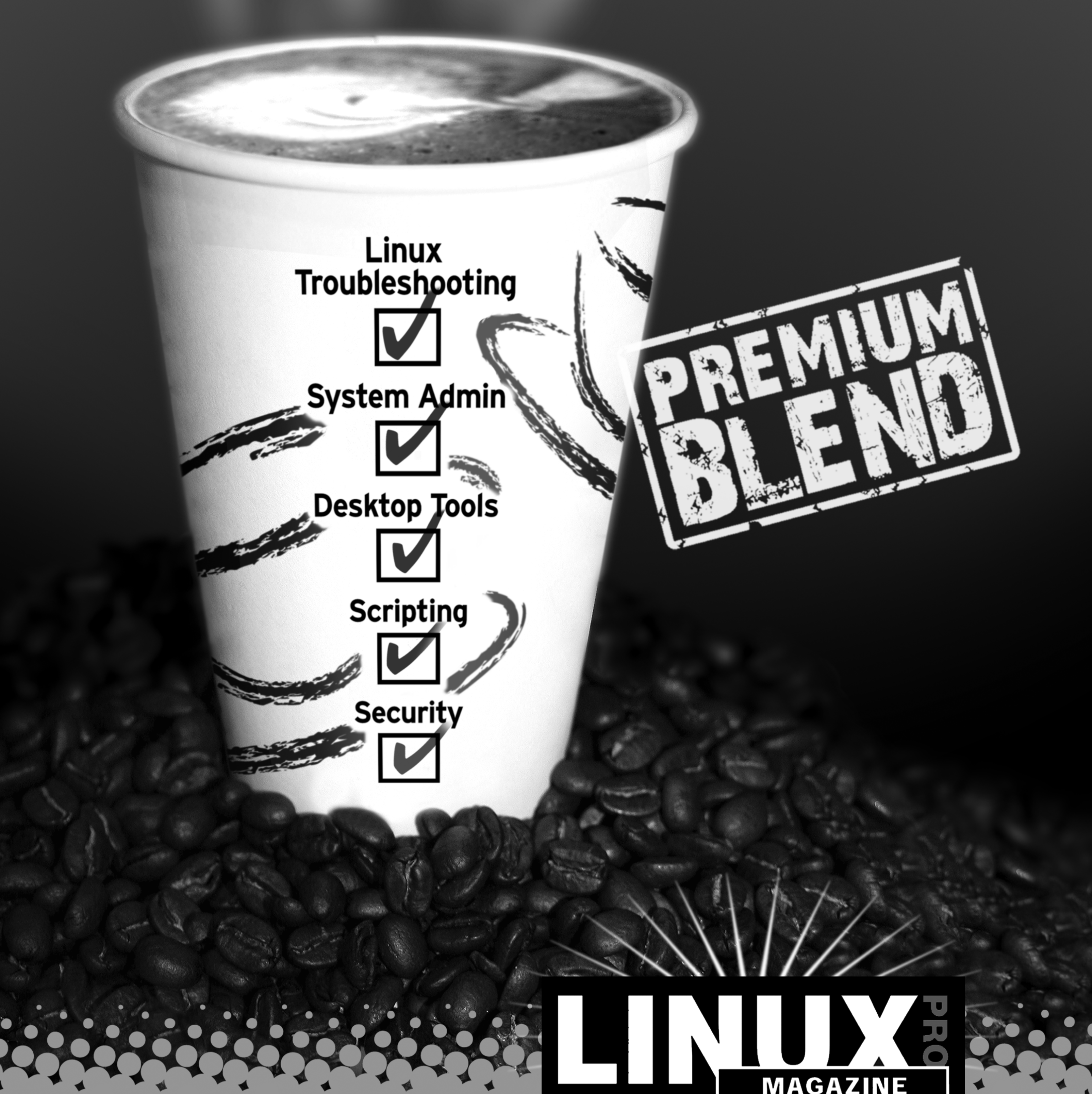
Conceived by software engineers.
Written by software engineers.
Strictly intended for software engineers.
ACM Queue examines the techniques and technologies that are just on the verge of making a disruptive difference, with a particular emphasis on the challenges that lie just ahead. ACM Queue also reviews and interprets promising research, focusing on how it might be applied to real-world systems. No hype. No nonsense. Just a hard-nosed focus on the problems and new opportunities expected to arise over the next few years.

www.acmqueue.org

- Redesigned for easy, intuitive navigation
- Free access to unlocked gems from ACM's Digital Library
- A critical concentration of substantive blogs
- Quality video and audio feeds
- Online discussions—and debates—with the experts

acm queue architecting tomorrow's computing

acm Association for Computing Machinery



LINUX PRO MAGAZINE

If you like the taste of Linux, why not treat yourself to the best?

Linux Pro Magazine delivers real-world solutions for the technical reader. In every issue, you'll find advanced techniques for configuring and securing Linux systems. Learn about the latest tools and discover the secrets of the experts in Linux Pro. Each issue includes a full Linux distribution on DVD.

www.linuxpromagazine.com

SAVE THE DATE! Mark your calendar to attend

DALLAS | Sept 23 SAN FRAN | Nov 17 WASH. DC | Dec 16

COMING TO A CITY NEAR YOU

IT ROADMAP IN '08!

What a great opportunity to network with my IT peers and the outstanding speakers on the state of IT.

FRANK CARLSON
TELEHEALTH AND CONFERENCING ENGINEER
NORTHERN COLORADO MEDICAL CENTER

Network World, helped us acquire new relationships with new vendors, as well as solidify existing relationships, with existing vendors.

SAJID QURESHI
IT PROCUREMENT AND SERVICE
DELIVERY MANAGER
HITT CONTRACTING, INC.

IT Roadmap Conference & Expo continues it's trek in 2008 with a nationwide tour including new cities, new topics, new speakers and new sponsors! That's right. You'll have a chance to attend one of the multi city events we'll be offering this year.

You won't want to miss out on 10 tracks of crucial network technology:

> VIRTUALIZATION
> ENTERPRISE MOBILITY
> NETWORK AND APPLICATION ACCELERATION
> NAC: NETWORK ACCESS CONTROL
> DATA CENTER INFRASTRUCTURE AND MANAGEMENT
> SECURITY AND COMPLIANCE
> NETWORK MANAGEMENT, AUTOMATION & CONTROL
> VOIP, VIDEO AND UNIFIED COMMUNICATIONS
> NEXT GENERATION WAN SERVICES
> SAAS AND CLOUD COMPUTING

Complete with case histories from front-line users. Data from industry researchers. Insights from IT specialists. And embedded within... a tightly-focused, solution oriented expo of top vendors.

We look forward to seeing you in 2008!

INTERESTED IN ATTENDING? INTERESTED IN SPONSORING?
www.networkworld.com/itr2008

ITRoadmap CONFERENCE & EXPO

Real world system administration training

LISA'08

22ND LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE

San Diego
CALIFORNIA
November 9–14, 2008

PLENARY SESSIONS

"Reconceptualizing Security," by Bruce Schneier, *Chief Security Technology Officer, BT*

"The State of Electronic Voting, 2008," by David Wagner, *University of California, Berkeley*

6 DAYS OF TRAINING

- **New! Virtualization Track:** Xen Hypervisor, VMware ESX 3i, and security taught by Stephen Spector, John Arrasjid, Phil Cox, and more
- **New! Solaris Track:** Debugging, administration, and DTrace taught by James Mauro, Peter Baer Galvin, Marc Staveley, and Jeff Victor
- Plus classes on Cfengine 3 by Mark Burgess, RRDtool by Tobias Oetiker, and more . . .

3-DAY TECHNICAL PROGRAM

- 2 tracks of Invited Talks by industry leaders on timely topics—live streaming available!
- Refereed papers on topics such as configuration management, parallel systems deployment, virtualization, and security
- Workshops, Guru Is In sessions, Birds-of-a-Feather sessions, Work-in-Progress reports, posters, and more!
- Vendor Exhibition: A showcase of the latest commercial innovations

SPONSORED BY
USENIX & [sage]

Register by October 17, 2008, and save! **http://www.usenix.org/lisa08/loo**

;login:
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES
RIDE ALONG ENCLOSED