;login:
OCTOBER 2014 VOL. 39, NO. 5

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Operating Systems

Columns

Conference Reports

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

UPCOMING EVENTS

OSDI '14: 11th USENIX Symposium on Operating Systems Design and Implementation

October 6–8, 2014, Broomfield, CO, USA
www.usenix.org/osdi14

Co-located with OSDI '14 and taking place October 5, 2014

Diversity '14: 2014 Workshop on Supporting Diversity in Systems Research
www.usenix.org/diversity14

HotDep '14: 10th Workshop on Hot Topics in System Dependability
www.usenix.org/hotdep14

HotPower '14: 6th Workshop on Power-Aware Computing and Systems
www.usenix.org/hotpower14

INFLOW '14: 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads
www.usenix.org/inflow14

TRIOS '14: 2014 Conference on Timely Results in Operating Systems
www.usenix.org/trios14

LISA14

November 9–14, 2014, Seattle, WA, USA
www.usenix.org/lisa14

Co-located with LISA14

URES '14 West: 2014 USENIX Release Engineering Summit West
November 10, 2014
www.usenix.org/ures14west

UCMS '14 West: 2014 USENIX Configuration Management Summit West
November 10, 2014
www.usenix.org/ucms14west

SESA '14: 2014 USENIX Summit for Educators in System Administration
November 11, 2014
USENIX Journal of Education in System Administration (JESA)
Published in conjunction with SESA
www.usenix.org/jesa

SaTCPI '15: National Science Foundation Secure and Trustworthy Cyberspace Principal Investigators' Meeting (2015)

January 5–7, 2015, Arlington, VA
Sponsored by the National Science Foundation
Presented by USENIX

FAST '15: 13th USENIX Conference on File and Storage Technologies

February 16–19, 2015, Santa Clara, CA, USA
www.usenix.org/fast15

NSDI '15: 12th USENIX Symposium on Networked Systems Design and Implementation

May 4–6, 2015, Oakland, CA
www.usenix.org/nsdi15

HotOS XV: 15th Workshop on Hot Topics in Operating Systems

May 18–20, 2015, Kartause Ittingen, Switzerland
Submissions due: January 9, 2015
www.usenix.org/hotos15

USENIX ATC '15: USENIX Annual Technical Conference

July 8–10, 2015, Santa Clara, CA, USA

Co-located with ATC '15 and taking place July 6–7, 2015

HotCloud '15: 7th USENIX Workshop on Hot Topics in Cloud Computing

HotStorage '15: 7th USENIX Workshop on Hot Topics in Storage and File Systems

USENIX Security '15: 24th USENIX Security Symposium

August 12–14, 2015, Washington, D.C., USA

LISA15

November 8–13, 2015, Washington, D.C., USA

Do you know about the USENIX Open Access Policy?

USENIX is the first computing association to offer free and open access to all of our conferences proceedings and videos. We stand by our mission to foster excellence and innovation while supporting research with a practical bias. Your membership fees play a major role in making this endeavor successful.

Please help us support open access. Renew your USENIX membership and ask your colleagues to join or renew today!

www.usenix.org/membership

Stay Connected...

twitter.com/usenix
www.usenix.org/youtube
www.usenix.org/gplus
www.usenix.org/facebook
www.usenix.org/linkedin
www.usenix.org/blog

;login:
OCTOBER 2014 VOL. 39, NO. 5

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EDITOR
Rik Farrow
rik@usenix.org

COPY EDITORS
Steve Gilmartin
Amber Ankerholz

PRODUCTION MANAGER
Michele Nelson

PRODUCTION
Arnold Gatilao
Jasmine Murcia

TYPESETTER
Star Type
startype@comcast.net

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

www.usenix.org

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

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for non-members are $90 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.

EDITORIAL

Musings

RIK FARROW

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

I've often rambled on about the future of operating systems, imagining something completely different and new. Of course, there are loads of practical issues with that path, like the inability to run any existing software on the spanking new OS. And it turns out there are still things about existing operating systems that can surprise me.

I speculated that a future operating system might not look at all like Linux, today's favorite OS for servers and for OS research projects too. Linux is large, complex, and difficult when it comes to incorporating large changes into it because of its history and design. While the BSD kernels are more modularly designed, they are less popular, and thus not as interesting, or even as well-known. And both are enormous, with many millions of lines of code. While Minix3 is much smaller and actually takes a new approach, it too suffers from the "not as popular as Linux" issue, like the BSDs.

I've watched the OS space for a while, curious to see if some of the less popular directions taken will pick up a lot of interest. And that interest generally comes from providing features that users, whether they are running servers in some cluster or researchers looking to add the next neat feature or improvement, just can't live without.

Size Is Not Everything

Today's OSes are huge. When I worked with Morrow Designs in the '80s, I actually put together a set of two, double-density, floppy disks that contained a bootable kernel and utilities you needed to recover an unbootable system. That was a total of less than 800 kilobytes of code for the equivalent of a rescue CD, which should sound ridiculous in this day and age. But is it really?

The Internet of Things (IoT) already includes inexpensive devices, and that means slower CPUs, small memories, and sometimes relatively generous amounts of flash. With small memories, these devices won't be booting a generic kernel but one trimmed down to the bare essentials. In one sense, that's easy enough: You can build a kernel without support for file systems and devices you will never use in a diskless system on chip (SoC) device. Popular examples of this include the Raspberry Pi and the BeagleBone Black.

But even these devices are overkill for many IoT applications. Another popular example is the Arduino family, which does not run *nix but may still include networking. Even simpler (and slower with less memory) are the Peripheral Interface Controllers (PICs), favored not just by hobbyists but also by device designers. These devices have really tiny amounts of RAM (really just RAM as registers), yet are more than adequate for many household and industrial devices. They do not run *nix, or even what could ever be called an operating system.

Let's head to the opposite extreme and consider IBM's Sequoia (Blue Gene/Q) that was installed at Lawrence Livermore National Labs in 2011. Like others in this series, the Sequoia's compute nodes (some 98,000 of them) run the Compute Node Kernel (CNK). The CNK operating system is just 5000 lines of C++, just enough to communicate with I/O nodes and launch applications that have been compiled just for the CNK environment. The concept behind CNK is simple: the bare minimum of memory and processing required so that

most of the CPU and memory can be devoted to computation. And it works, as the Sequoia was the world's fastest computer for a while, as well as using 37% less energy than the computer (BG/K) it replaced.

So, the Sequoia runs a more sophisticated version of what runs on Arduinos on its compute nodes. There is no memory management or thread scheduler: Applications are single threaded and run in physical rather than virtual memory.

Stripped Down

The same stripped down to bare essentials approach can be found in rump kernels. The brainchild of Antti Kantee, rump kernels provide just those portions of an operating system needed to run a single application in physical memory, with no scheduler. Kantee refactored the NetBSD kernel into a base and three modules that allow the rump kernel to support applications that can run on bare metal or on top of a hypervisor. Not that rump kernels are the only game in town: OSv, MirageOS, and Erlang-on-Xen all are designed to remove the need for a full operating system and its environment when running on top of a hypervisor.

There's yet another way to stop layering operating systems over a hypervisor operating system, and it has been around for many years. You may have heard of LXC, a project that has been used for years as a way of providing the illusion of having your own hosted system. With LXC, and related technology like Solaris Zones, there is only one operating system. LXC, or other container software, provides the illusion of being the master, root, of your own system, when what you really are running is a group of processes in a jail. Just as the BSD jail has evolved over time, so has the Linux container. James Bottomley discusses Linux containers in an article in this issue, and he and his co-author have left me feeling like real progress has been made in making containers both secure and efficient.

Still in the theme of "stripped down," but not related to operating systems, I had hoped to get Ben Treynor (Google) to write about the concept of the *error budget*. Treynor introduced this idea during his keynote at the first SREcon, and I will try to cover it concisely here. Imagine that you are running software-as-a-service (SaaS) on an immense scale, that you must do so efficiently (no operators, just skilled SREs) but do not want to violate your service level agreement of five 9s, or 99.999% uptime. At the same time, you continually need to update your client-facing software. Your error budget includes some tiny fraction of your total capacity for providing SaaS for testing. And the better job you do of testing, the further your error budget, that .001%, can stretch. Read Dan Klein's article and perhaps you will see how Google's approach to updating software fits into this concept of the error budget.

I still hope that Ben Treynor will have the time to write for us someday.

The Lineup

We begin this issue with two operating systems-related articles. When I met Kirill Korotaev (Parallels) during Linux FAST '14, I was already interested in Linux container technology. I caught up with Kirill during a break, and asked him to write about Linux containers. Kirill suggested James Bottomley, and James agreed to write, working with Pavel Emelyanov. They've produced both a history and an excellent description of Linux containers for this issue.

Greg Burd (Amazon) had suggested that I publish an article about rump kernels in 2013, but I didn't think the technology was ready. When Antti Kantee volunteered to write about rump kernels this summer, I took another look. Kantee actually wrote his PhD dissertation about refactoring the NetBSD kernel to support the concept of rump kernels: a method of supplying the parts of an OS you need for a particular application, and no more. He and Justin Cormack continue to work at making rump kernels easier to use, and their article in this issue explains the concept in detail.

I met Steve Muir during ATC '14. While Steve presented the paper, four other people from Comcast were involved in the research. Their goal was to create an in-memory database for a read-only service that could be transparently updated. Their use of Paxos as a means of managing updates between a hierarchy of servers got me interested, plus their software is open source.

A student of John Ousterhout, Diego Ongaro, presented a Best Paper at ATC '14, "In Search of an Understandable Consensus Algorithm," which the researchers offer as a replacement for Paxos. Although the subject is not covered in this issue, Raft is focused on applications like the one the Comcast people wrote, and on RAMCloud (of course). You can find the Ongaro paper on the USENIX Web site, as well as videos explaining how Raft works, by searching online.

Dan Klein, Dina Betser, and Mathew Monroe have written about the process they use within Google to push software updates. While the process is quite involved, I had heard about parts of it before Dan volunteered to write for *;login:* from various sources. And it both makes sense and realizes a cautious yet realistic approach to upgrading software without causing catastrophic failures—perhaps just small-scale ones within the error budget. Klein's article covers not only updating but also a further optimization that will make the process more efficient, involving less human interaction.

Andy Seely continues his series of columns about managing system administrators. In this contribution, Andy relates a set of three parables he uses as guides and stories he can share to motivate co-workers.

EDITORIAL

Musings

Charles Border and Kyrre Begnum introduce a workshop and a new journal. The Summit for Educators in System Administration had its first official meet at LISA '13, and will occur again at LISA14. The *Journal of Education in System Administration* (JESA) provides a mechanism for publishing research about educating system administrators year round.

Dilma Da Silva has written her second article about CRA-W, the organization devoted to helping woman PhD candidates in the fields of computer science and engineering. Dilma discusses the Grad Cohort, a yearly gathering of grad students and mentors focused on providing useful information about both completing grad school successfully and planning beyond grad school. And right now (late 2014) is the time to be making plans, and applying for support, to attend Grad Cohort 2015.

David Blank-Edelman claims that this time he is going to be totally practical about his chosen topic. I would claim David is always practical and pragmatic. David has been working on health checks for a small cluster of LDAP servers, and he takes us through both aspects of what a health check requires and Perl support for querying LDAP servers.

Dave Beazley considers Python's problems with paths. It's not so much that Python can't manipulate pathnames. It's just that the ways of doing so have been disjointed, involving multiple OS modules. Well, things have gotten more elegant with a new module, pathlib, available as part of Python 3.4.

Dave Josephsen continues on his mission of evangelizing for the proper design and use of monitoring systems. In this column, Dave rails against the arithmetic mean, showing just how badly the mean works when used to summarize/compress time series data. And, of course, Dave offers alternatives.

Dan Geer has written a concise article clearing up the confusion surrounding terms like false positive and true negative. Dan not only does this, but also provides an example for determining the most efficient ordering of tests for sensitivity and specificity.

Robert Ferrell, having recently retired from being a badge-carrying Fed (bet you didn't suspect that), has decided to poke fun at the Internet of Things. Even as people rush to connect their cars and thermostats up to the Internet, Robert points out that the security of these devices is about on par with that of the Internet—in 1994.

I've written a review of the new edition of the *Design and Implementation of the FreeBSD Operating System*. It's not the first time I've taken a look at similar volumes, as past USENIX president Kirk McKusick has been part of writing about BSD operating systems for over 20 years. This edition, the first in 10 years, contains several new chapters as well as much updated material.

Mark Lamourine, while technically a system administrator, continues to write excellent reviews of books on programming topics. This time, he covers books on when and how to use Bayesian statistics, understanding when refactoring an imperative program to use functional programming features can help, and an experimental work called the *Go Developer's Notebook*.

We have lots of summaries: ATC '14, HotCloud '14, HotStorage '14, WiAC '14, and ICAC '14. Most are incomplete, as there were too many sessions to cover and not enough volunteers—with the exception of the WiAC summary, which was thoroughly covered by Amy Yin. If you are planning on attending LISA14, and want to be certain a favorite session gets covered, contact me to volunteer.

The first time I attended the OSDI conference, I asked someone I knew why there weren't any papers about new OS designs. His answer was simple: It's hard. Designing a new OS takes many years and is also a risky endeavor. That's why I now look more closely at important but incremental changes, like unified container support for Linux, and at work like Kantee's, where he has converted a complete kernel into a more modular form. And, I continue to watch seL4, which just went open source (July 2014), Arrakis, and Minix3.

Containers

JAMES BOTTOMLEY AND PAVEL EMELYANOV

James Bottomley is CTO of server virtualization at Parallels where he works on container technology and is Linux kernel maintainer of the SCSI subsystem. He is currently a director on the Board of the Linux Foundation and chair of its Technical Advisory Board. He went to university at Cambridge for both his undergraduate and doctoral degrees after which he joined AT&T Bell Labs to work on distributed lock manager technology for clustering. In 2000 he helped found SteelEye Technology becoming vice president and CTO. He joined Novell in 2008 as a Distinguished Engineer at Novell's SUSE Labs and Parallels in 2011. jbottomley@parallels.com

Pavel Emelyanov is a principal engineer at Parallels working on the company's cloud server projects. He holds a PhD in applied mathematics from the Moscow Institute of Physics and Technology. His speaking experience includes talks about container virtualization at LinuxCon 2009, at the joint memory management, storage and file-system summit in 2011, and about checkpoint-restore on LinuxCon Europe 2012 and Linux Conf AU 2013. xemul@parallels.com

Today, thanks to a variety of converging trends, there is huge interest in container technology, but there is also widespread confusion about just what containers are and how they work. In this article, we cover the history of containers, compare their features to hypervisor-based virtualization, and explain how containers, by virtue of their granular and specific application of virtualization, can provide a superior solution in a variety of situations where traditional virtualization is deployed today.

Since everyone knows what hypervisor-based virtualization is, it would seem that comparisons with hypervisors are the place to begin.

Hypervisors and Containers

A hypervisor, in essence, is an environment virtualized at the hardware level.

In this familiar scenario, the hypervisor kernel, which is effectively a full operating system, called the host operating system, emulates a set of virtual hardware for each guest by trapping the usual operating system hardware access primitives. Since hardware descriptions are well known and well defined, emulating them is quite easy. Plus, in the modern world, CPUs now contain special virtualization instruction extensions for helping virtualize hard-to-emulate things like paging hardware and speeding up common operations. On top of this emulated hardware, another operating system, complete with unmodified kernel (we're ignoring paravirtual operating systems here for the sake of didactic simplicity), is brought up. Over the past decade, remarkable strides have been made in expanding virtualization instructions within CPUs so that most of the operations that hardware-based virtualization requires can be done quite efficiently in spite of the huge overhead of running through two operating systems to get to real hardware.

Containers, on the other hand, began life under the assumption that the operating system itself could be virtualized in such a way that, instead of starting with virtual hardware, one could start instead with virtualizing the operating system kernel API (see Figure 2).

In this view of the world, the separation of the virtual operating systems begins at the init system. Historically, the idea was to match the capabilities of hypervisor-based virtualization (full isolation, running complete operating systems) just using shared operating system virtualization techniques instead.

In simplistic terms, OS virtualization means separating static resources (like memory or network interfaces) into pools, and dynamic resources (like I/O bandwidth or CPU time) into shares that are allotted to the virtual system.

A Comparison of Approaches

The big disadvantage of the container approach is that because you have to share the kernel, you can never bring up two operating systems on the same physical box that are different at the kernel level (like Windows and Linux). However, the great advantage is that, because a single kernel sees everything that goes on inside the multiple containers, resource sharing

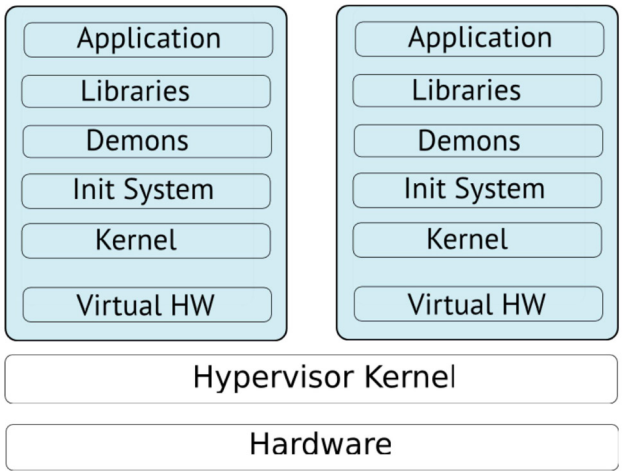


Figure 1: Hypervisor diagram

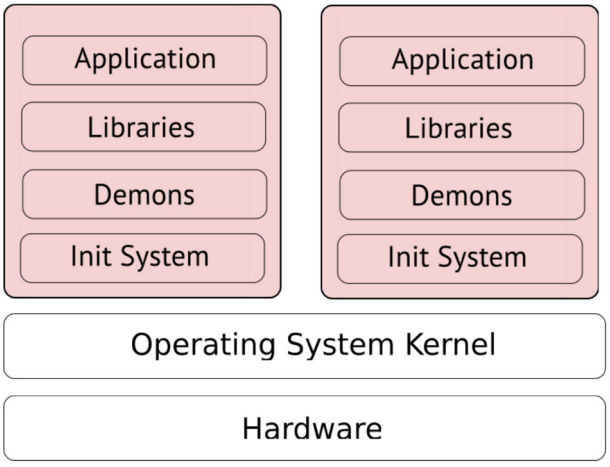


Figure 2: Container diagram

and efficiency is greatly enhanced. Indeed, although the container stack is much thinner than the hypervisor stack by virtue of not having to run two kernels, most of the container improvements in density in fact come from the greater resource efficiency (in particular, sharing the page cache of the single kernel). The big benefit, of course, is that the image of what's running in the container (even when it's a full operating system) is much smaller. This means that containers are much more elastic (faster to start, stop, migrate, and add and remove resources, like memory and CPU) than their hypervisor cousins. In many ways, this makes container technology highly suited to the cloud, where homogeneity is the norm (no running different operating systems on the same physical platform) and where elasticity is supposed to be king.

Another great improvement containers have over hypervisors is that the control systems can operate at the kernel (hence API) level instead of at the hardware level as you have to do with hypervisors. This means, for instance, that the host operating system can simply reach inside any container guest to perform any operation it desires. Conversely, achieving this within a hypervisor usually requires some type of hardware console emulating plus a special driver running inside the guest operating system. To take memory away from a container, you simply tune its memory limit down and the shared kernel will instantly act on the instruction. For a hypervisor, you have to get the cooperation of a guest driver to inflate a memory balloon inside the guest, and then you can remove the memory from within this balloon. Again, this leads to greatly increased elasticity for containers because vertical scaling (the ability of a virtual environment to take over or be scaled back from the system physical resources) is far faster in the container situation than in the hypervisor one.

The History of Containers

In many ways, the initial idea of containers goes back to Multics (the original precursor to UNIX) and the idea of a multi-user time-sharing operating system. In all time-sharing systems, the underlying operating system is supposed to pretend to every user that they're the sole owner of the resources of the machine, and even impose limits and resource sharing such that two users of a time-sharing system should not be able materially to impact one another.

The first real advance was around 1982 with the BSD `chroot()` system call leading to the Jail concept, which was founded in the idea of logically disconnecting the Jail from the rest of the system by isolating its file-system tree such that you could not get back out from the containerized file system into the host (although the host could poke about in the Jailed directory to its heart's content).

In 1999, SWsoft began the first attempts at shared operating system virtualization, culminating with the production release of Virtuozzo containers in 2001. Also in 2001, Solaris released Zones. Both Virtuozzo and Zones were fully isolating container technology based on capabilities and resource controls.

In 2005, an open source version of Virtuozzo (called OpenVZ) was released, and in 2006 an entirely new system called process containers (now CGroups) was developed for the Linux kernel. In 2007, Google saw the value of containers, hired the CGroups developers, and set about entirely containerizing the Googleplex (and making unreleased additions to their container system in the meantime), and in 2008, the first release of LXC (LinuX Containers) based wholly on upstream was made. Although OpenVZ was fully open source, it was never integrated into the Linux mainstream (meaning you always had to apply additional

patches to the Linux kernel to get these container systems), which by 2011 led to the situation in which there were three separate Linux container technologies (OpenVZ, CGroups/namespaces, and the Google enhancements). However, at the fringes of the 2011 Kernel Summit, all the container parties came together for a large meeting, which decided that every technology would integrate upstream, and every out-of-Linux source tree container provider would use it. This meant selecting the best from all the out-of-tree technologies and integrating them upstream. As of writing this article, that entire program is complete except for one missing CGroups addition: the kernel memory accounting system, which is expected to be in Linux by kernel version 3.17.

The VPS Market and the Enterprise

In Web hosting parlance, VPS stands for Virtual Private Server and means a virtual instance, sold cheaply to a customer, inside of which they can run anything. If you've ever bought hosting services, the chances are what you bought was a VPS. Most people buying a VPS tend to think they have bought a hypervisor-based virtual machine, but in more than 50% of the cases the truth is that they've actually bought a container pretending to look like a hypervisor-based virtual machine. The reason is very simple: density. The VPS business is a race to the bottom and very price sensitive (the cheapest VPSes currently go for around $10 US a month) and thus has a very low margin. The ability to pack three times as many virtual container environments on a single physical system is often the difference between profit and loss for hosters, which explains the widespread uptake of containers in this market.

Enterprises, by contrast, took to virtualization as a neat way of repurposing the excess capacity they had within datacenters as a result of mismatches between application requirements and hardware, while freeing then from the usual hardware management tasks. Indeed, this view of virtualization meant that the enterprise was never interested in density (because they could always afford more machines) and, because it built orchestration systems on varied virtual images, the container disadvantage of being unable to run operating systems that didn't share the same kernel on the same physical system looked like a killer disadvantage.

Because of this bifurcation, container technology has been quietly developing for the past decade but completely hidden from the enterprise view (which leads to a lot of misinformation in the enterprise space about what containers can and cannot do). However, in the decade where hypervisors have become the standard way of freeing the enterprise datacenter from hardware dependence, several significant problems like image sprawl (exactly how many different versions of operating systems do you have hidden away in all your running and saved hypervisor images) and the patching problem (how do you identify and add all the security fixes to all the hypervisor images in your entire organization) have lead to significant headaches and expensive tooling to solve hypervisor-image lifecycle management.

Container Security and the Root Problem

One of the fairly ingrained enterprise perceptions is that containers are insecure. This is fed by the LXC technology, which, up until very recently, was not really secure, because the Linux container security mechanisms (agreed upon in 2011) were just being implemented. However, if you think about the requirements for the VPS market, you can see that because hosting providers have to give root access to most VPS systems they sell, coping with hostile root running within a container was a bread-and-butter requirement even back in 2001.

One of the essential tenets of container security is that root (UID 0 in UNIX terms) may not exist within the container, because if it broke out, it would cause enormous damage within the host. This is analogous to the principle of privilege separation in daemon services and functions in a similar fashion. In upstream Linux, the mechanism for achieving this (called the user namespace) was not really functional until 2012 and is today only just being turned on by the Linux distributions, which means that anyone running a distribution based on a kernel older than 3.10 likely doesn't have it enabled and thus cannot benefit from root separation within the container.

Containers in Linux: Namespaces and CGroups

In this section, we delve into the Linux specifics of what we use to implement containers. In essence, though, they are extensions of existing APIs: CGroups are essentially an extension of Resource Limits (POSIX RLIMITs) applied to groups of processes instead of to single processes. Namespaces are likewise sophisticated extensions of the `chroot()` separation system applied to a set of different subsystems. The object of this section is to explain the principles of operation rather than give practical examples (which would be a whole article in its own right).

Please also bear in mind as you read this section that it was written when the 3.15 kernel was released. The information in this section, being very Linux specific, may have changed since then.

CGroups

CGroups can be thought of as resource controllers (or limiters) on particular types of resources. The thing about most CGroups is that the control applies to a group of processes (hence the interior of the container becomes the group) that it's inherited across forks, and the CGroups can actually be set up hierarchically. The current CGroups are:

- blkio—controls block devices
- cpu and cpuacct—controls CPU resources

- cpuset—controls CPU affinity for a group of processes
- devices—controls device visibility, effectively by gating the `mknod()` and `open()` calls within the container
- freezer—allows arbitrary suspend and resume of groups of processes
- hugetlb—controls access to huge pages, something very Linux specific
- memory—currently controls user memory allocation but soon will control both user and kernel memory allocations
- net_cls and net_prio—controls packet classification and prioritization
- perf_event—controls access to performance events

As you can see from the brief descriptions, they're much more extensive than the old RLIMIT controls. With all of these controllers, you can effectively isolate one container from another in such a way that whatever the group of processes within the container do, they cannot have any external influence on a different container (provided they've been configured not to, of course).

Namespaces

Although, simplistically, we've described namespaces as being huge extensions of `chroot()`, in practice, they're much more subtle and sophisticated. In Linux there are six namespaces:

- Network—tags a network interface
- PID—does a subtree from the fork, remapping the visible PID to 1 so that init can work
- UTS—allows specifying new host and NIS names in the kernel
- IPC—separates the system V IPC namespace on a per-container basis
- Mount—allows each container to have a separate file-system root
- User—does a prescribed remapping between UIDs in the host and container

The namespace separation is applied as part of the `clone()` flags and is inherited across forks. The big difference from `chroot()` is that namespaces tag resources and any tagged resources may disappear from the parent namespace altogether (although some namespaces, like PID and user are simply remappings of resources in the parent namespace).

Container security guarantees are provided by the user namespace, which maps UID 0 within the container (the root user and up, including well known UIDs like bin) to unused UIDs in the host, meaning that if the apparent root user in the container ever breaks out of the container, it is completely unprivileged in the host.

Containers as the New Virtualization Paradigm

One of the ironies of container technology is that, although it has spent the last decade trying to look like a denser hypervisor (mostly for the VPS market), it is actually the qualities that set it apart from hypervisors that are starting to make container technology look interesting.

Green Comes to the Enterprise

Although the enterprise still isn't entirely interested in density for its own sake, other considerations besides hardware cost are starting to be felt. In particular, green computing (power reduction) and simply the limits imposed by a datacenter sited in a modern city—the finite capacity of a metropolitan location to supply power and cooling—dictate that some of the original container differentiators now look appealing. After all, although the hosting providers primarily demand density for cost reasons, the same three times density rationale can also be used to justify running three times as many applications for the same power and cooling requirements as a traditional hypervisor and, thus, might just provide the edge to space-constrained datacenters in downtown Manhattan, for example.

Just Enough Virtualization

The cost of the past decade of hypervisor-based virtualization has been that although virtual machine images mostly perform a specific task or run a particular application, most of the management software for hypervisor-based virtualization is concerned with managing the guest operating system stack, which is entirely superfluous to the running application. One of the interesting aspects of containers is that instead of being all or nothing, virtualization can be applied on a per-subsystem basis. In particular, because of the granularity of the virtualization, the amount of sharing between the guest and the host is adjustable on a continuous scale. The promise, therefore, is that container-based virtualization can be applied only to the application, as shown in Figure 3 where a traditional operating system container is shown on the left-hand side and a new pure-application

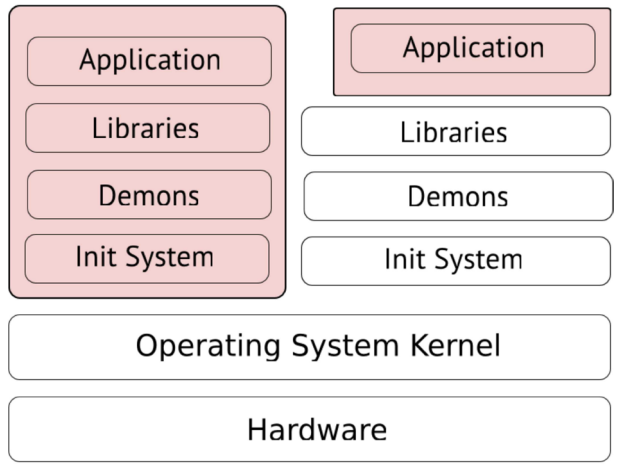


Figure 3: Containerizing just the application

container is shown on the right. If done correctly, this type of application virtualization can make management of the support operating system a property of the host platform instead of being, as it is today with hypervisors, a property of every virtual machine image.

This new "just enough virtualization" world promises to greatly reduce the image sprawl problem by making sure that the virtualized image contains only enough elements to support the application or task itself instead of being a full-fledged operating system image in its own right.

Solving Current Problems with Containers

As an illustration of the way containerization can solve existing problems in a new way, consider the problem of tenancy in the cloud: Standard enterprise applications are designed to serve a single tenant. What this means in practice is that one overall administrator for the enterprise application administers the application for all users. If this application is transferred to the cloud, in its enterprise incarnation, then each consumer (or tenant) wants to designate an administrator who can only administer users belonging to the tenant. The tenancy problem can be solved by running the application inside a virtual machine with one VM per tenant, but it can be solved much more elegantly by adding a small amount of containerization to the application. A simple recipe to take a single tenant application and make it multi-tenant is to fork the application once for each tenant; to each fork, add a new network namespace so that it can have its own IP address, and a new mount namespace so that it can have a private datastore. Because we added no other containerization, each fork of the application shares resources with the host (although we could add additional containerization if this becomes a concern), so the multi-tenant application we have created is now very similar to a fleet of simple single tenant applications. In addition, because containers are migratable, we can even scale this newly created multi-tenant application horizontally using container migration techniques.

Enabling a Containerized Future

The multi-tenant example above shows that there might be a need for even applications to manipulate container properties themselves. Thus, to expand the availability and utility of container technologies a consortium of companies has come together to create a library for manipulating basic container properties. The current C version of this library exists on GitHub (https://github.com/xemul/libct), but it will shortly be combined with a GO-based libcontainer to provide bindings for C, C++, Python, and Go. Although designed around the Linux container API, the library nevertheless has flexibility to be used as a backend to any container system (including Solaris Zones or Parallels Containers for Windows). This would mean, provided the portability works, that the direct benefits of containerizing applications would be exported to platforms beyond Linux.

Conclusions

Hopefully, you now have at least a flavor of what containers are, where they came from, and, most importantly, how their differences from hypervisors are being exploited today to advance virtualization to the next level of usability and manageability. The bottom line is that containers have a new and interesting contribution to make; they've gone from being an expense-reducing curiosity for Web applications to the enterprise mainstream, and they hold the possibility of enabling us to tailor container virtualization to the needs of the application, and thus give applications interesting properties that they haven't been able to possess before.

Resources

The subject of varied uses of containers is very new, so there are few articles to refer to. However, here are some useful Web references on the individual technologies that have been used to create containers on Linux.

Michael Kerrisk of *Linux Weekly News* did a good online seven-part write-up of what namespaces are and how they work: http://lwn.net/Articles/531114/.

Neil Brown as a guest author for *Linux Weekly News* has done a good summary of CGroups: http://lwn.net/Articles/604609/.

This blog post on network namespaces is a useful introduction to using the separated capabilities of namespaces to do interesting things in tiny semi-virtualized environments: http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/.

Rump Kernels

No OS? No Problem!

ANTTI KANTEE, JUSTIN CORMACK

Antti Kantee got bitten by the OS bug when he was young, and is still searching for a patch. He has held a NetBSD commit bit for fifteen years and for the previous seven of them he has been working on rump kernels. As a so-called day job, Antti runs a one-man "systems design and implementation consulting" show.
pooka@fixup.fi

Justin Cormack accidentally wandered into a room full of UNIX workstations at MIT in the early 1990s and has been using various flavors ever since. He started working with rump kernels last year and recently acquired a NetBSD commit bit. He is generally found in London these days.
justin@myriabit.com

In the modern world, both virtualization and plentiful hardware have created situations where an OS is used for running a single application. But some questions arise: Do we need the OS at all? And by including an OS, are we only gaining an increased memory footprint and attack surface? This article introduces rump kernels, which provide NetBSD kernel drivers as portable components, allowing you to run applications without an operating system.

There is still a reason to run an OS: Operating systems provide unparalleled driver support, e.g., TCP/IP, SCSI, and USB stacks, file systems, POSIX system call handlers, and hardware device drivers. As the name *rump* kernel suggests, most of the OS functionality not related to drivers is absent, thereby reducing a rump kernel's footprint and attack surface.

For example, a rump kernel does not provide support for executing binaries, scheduling threads, or managing hardware privilege levels. Yet rump kernels can offer a complete enough environment to support unmodified POSIXy applications on top of them (Figure 1). In this article, we explain how rump kernels work and give you pointers on how you can benefit from them in your projects.

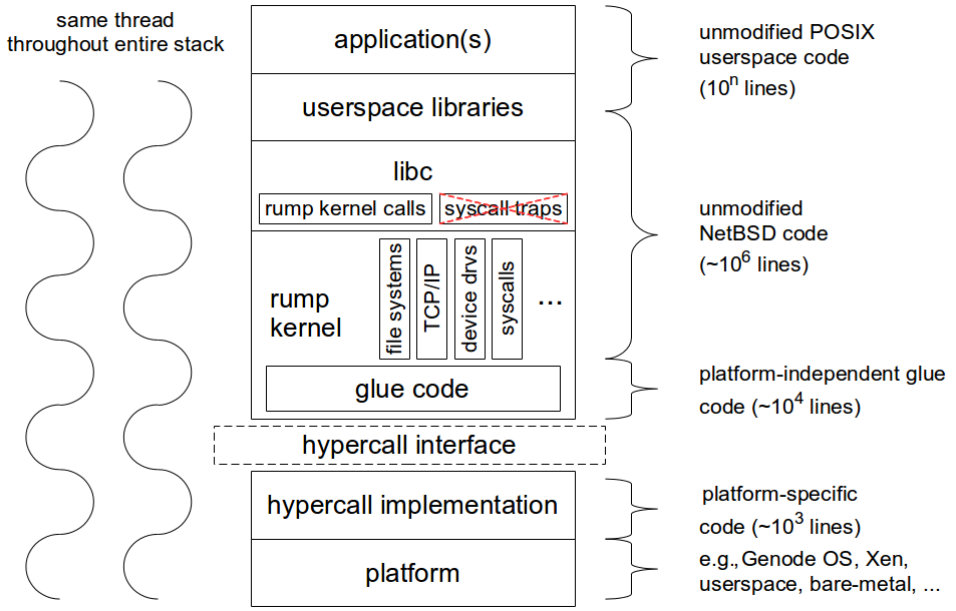


Figure 1: Rump kernels provide file system, network, and other driver support and run on bare metal systems or hypervisors by making use of a hypercall interface. In the depicted case, rump kernels provide the support necessary for running applications without requiring a full OS.

If you are building an OS-less system, why use rump kernels as your drivers? Rump kernels consist of a pool of roughly one million lines of unmodified, battle-hardened NetBSD kernel drivers running on top of a documented interface. The implementation effort for the interface, should your given platform not already be supported, is approximately 1,000 lines of C code. As the old joke goes, writing a TCP/IP stack from scratch over the weekend is easy, but making it work on the real-world Internet is more difficult. A similar joke about porting an existing TCP/IP stack out of an OS kernel most likely exists. Furthermore, the TCP/IP stack is only one driver, so you need plenty of spare weekends with the "roll your own" approach. The, we daresay, magic of rump kernels working in the real world stems from unmodified code. Driver bugs in operating systems have been ironed out over years of real-world use. Since rump kernels involve no porting or hacking of individual drivers, no new bugs are introduced into the drivers. The more unmodified drivers you use, the more free maintenance of those drivers you get. We are not suggesting that OS kernel drivers are optimal for all purposes but that it is easy to start with profiling and optimizing a software stack that works right off the bat.

In related work, there are a number of contemporary projects focusing on avoiding the overhead and indirection of the OS layer in the cloud: for example, MirageOS, OSv, and Erlang-on-Xen. But our goal with rump kernels is different. We aim to provide a toolkit of drivers for any platform instead of an operating environment for cloud platforms. In that sense, rump kernels can be thought of being like lwIP [1], except the scope is beyond networking (and the TCP/IP stack is larger). That said, we do also provide complete support for rump kernels on a number of platforms, including POSIXy user space and Xen. We also integrate with a number of other frameworks. For example, drivers are available for using the TCP/IP stack offered by rump kernels with user space L2 packet frameworks such as netmap, Snabb Switch, and DPDK.

The beef of rump kernels, pun perhaps intended, is allowing third-party projects access to a pool of kernel-quality drivers, and Genode OS [2] has already made use of this possibility. Although there are other driver toolkits (e.g., DDEKit [3]), we claim that rump kernels are the most complete driver kit to date. Furthermore, support for rump kernels is directly included in the NetBSD source tree. One example of the benefit of in-tree support is that in case of an attractive new driver hitting the NetBSD tree, there is no waiting for someone to roll the driver kit patches forwards, backwards, and sideways. You can simply use any vintage of NetBSD as a source of rump kernels.

We will avoid going into much technical detail in this article. The book [4] provides more detailed descriptions for interested parties.

History

Rump kernels started in 2007 as a way to make debugging and developing NetBSD kernel code easier. Developing complex kernel code usually starts out by sketching and testing the central pieces in the comfort of user space, and only later porting the code to the kernel environment. Despite virtual machines and emulators being plentiful in this age, the user space approach is still used, suggesting that there is something which makes user space a simpler platform to work with.

Even though rump kernels started out as running kernel code in user space, they were never about running the full OS there, because a user space OS is fundamentally not different from one running in a virtual machine and introduces unnecessary complexity for development purposes. From the beginning, rump kernels were about bringing along the minimum amount of baggage required to run, debug, examine, and develop kernel drivers. Essentially, the goal was to make developing drivers as easy as in user space, but without having to port kernel code to user space and back. From that desire a very significant feature of the rump kernel arose: It was necessary that exactly the same driver code ran both in debug/development mode and in the NetBSD kernel, and hacks like `#ifdef TESTING` were not permitted.

Problems related to development, testing, and debugging with rump kernels were more or less addressed by 2011, and the fundamental concepts of rump kernels have remained unchanged since then. Then a new motivation for rump kernels started emerging. The effort to make kernel drivers run in user space had essentially made most kernel drivers of NetBSD portable and easy to integrate into other environments. Adding support for platforms beyond user space was a simple step. The goal of development shifted to providing reusable drivers and a supporting infrastructure to allow easy adaptation. Testing, of course, still remains a central use case of rump kernels within NetBSD, as does, for example, being able to run the file system drivers as user-space servers.

Making Rump Kernels Work

Rump kernels are constructed out of components. The drivers are first built for the target system as libraries, and the final runtime image is constructed by linking the component-libraries together, along with some sort of application, which controls the operation of the drivers (see Figure 1 for an example). Notably, the application does not have to be a POSIXy user-space application. For example, when using rump kernels as microkernel-style user space file servers, the "application" is a piece of code that reads requests from the FUSE-like user space file systems framework and feeds them into the rump kernel at the virtual file system layer.

The starting point for coming up with the components was a monolithic kernel operating system. The problem is that we want to use drivers without bringing along the entire operating system kernel. For example, let us assume we want to run a Web server serving dynamically created content, perhaps running on an Internet-of-Things device. All we need in the rump kernel is the TCP/IP stack and sockets support. We do not need virtual memory, file systems(!), or anything else not contributing to the goal of talking TCP. As the first step, we must be able to "carve" the TCP/IP stack out of the kernel without bringing along the entire kitchen-sinky kernel, and give others an easy way to repeat this "carving." Second, we must give the rump kernel access to platform resources, such as memory and I/O device access. These issues are solved by the anykernel and the rump kernel hypercall interface, respectively.

Anykernel

The enabling technology for rump kernels in the NetBSD codebase is the anykernel architecture. The "any" in "anykernel" is a reference that it is possible to use drivers in any configuration: monolithic, microkernel, exokernel, etc. If you are familiar with the concept of kernel modules, you can think of the anykernel roughly as an architecture which enables loading kernel modules into places beyond the original OS.

We realize the anykernel by treating the NetBSD kernel as three layers: base, factions, and drivers. Note, this layering is *not* depicted in Figure 1, although one might replace the "rump kernel" box with such layers. The base contains fundamental routines, such as allocators and synchronization routines, and is present in every rump kernel. All other kernel layers are optional, although including at least some of them makes a rump kernel instance more exciting. There are three factions and they provide basic support routines for devices, file systems, and networking. The driver layer provides the actual drivers such as file systems, PCI drivers, firewalls, software RAID, etc. Notably, in addition to depending on the base and one or more factions, drivers may depend on other drivers and do not always cleanly fit into a single faction. Consider NFS, which is half file system, half network protocol. To construct an executable instance of a rump kernel supporting the desired driver, one needs the necessary dependent drivers (if any), a faction or factions, and the base.

Let us look at the problem of turning a monolithic kernel into an anykernel in more detail. Drivers depend on bits and pieces outside of the driver. For example, file system drivers generally depend on at least the virtual file system subsystem in addition to whichever mechanism they use to store the file system contents. Simply leaving the dependencies out of the rump kernel will cause linking to fail, and just stubbing them out as null functions will almost certainly cause things to not work correctly. Therefore, we must satisfy all of the dependencies of the drivers linked into the rump kernel.

Popular myth would have one believe that a monolithic kernel is so intertwined that it is not possible to isolate the base, factions, and drivers. The myth was shown to be false by the "come up with a working implementation" method.

Honestly speaking, there is actually not much "architecture" to the anykernel architecture. One could compare the anykernel to an SMP-aware kernel, in which the crux is not coming up with the locking routines, but sprinkling their use into the right places. Over the monolithic kernel, the anykernel is merely a number of changes that make sure there are no direct references where there should not be any. For example, some source modules that were deemed to logically belong to the base contained references to file system code. Such source modules were split into two parts, with one source module built into the base and the split-off source module built into the file system faction. In monolithic kernel mode, both source modules are included.

In addition, cases where a rump kernel differs from the full-blast monolithic kernel may require glue code to preserve correct operation. One such example revolves around threads, which we will discuss in the next section; for now, suffice it to say that the method the monolithic kernel uses for setting and fetching the currently running thread is not applicable to a rump kernel. Yet we must provide the same interface for drivers. This is where glue code kicks in. The trick, of course, is to keep the amount of glue code as small as possible to ensure that the anykernel is maintainable in NetBSD.

The anykernel does not require any new approaches to indirection or abstraction, just plain old C linkage. Sticking with regular C is dictated by practical concern; members of an operating system project will not like you very much if you propose indirections that hurt the performance of the common case where the drivers are run in the monolithic kernel.

Hypercalls

To operate properly, the drivers need access to back-end resources such as memory and I/O functions. These resources are provided by the implementation of the rump kernel hypercall interface, rumpuser [5]. The hypercall interface ties a rump kernel to the platform the rump kernel is run on. The name hypercall interface is, you guessed it, a remnant of the time when rump kernels ran only in user space.

We assume that the hypercall layer is written on top of a platform in a state where it can run C code and do stack switching. This assumption means that a small amount of bootstrap code needs to exist in bare-metal type environments. In hosted environments, e.g., POSIX user space, that bootstrap code is implicitly present.

Very recently, we learned about the Embassies project [6], where one of the goals is to come up with a minimal interface for running applications and implement a support for running POSIX programs on top of that minimal interface. This is more or less what rump kernels are doing, with the exception that we are running kernel code on top of our minimal layer. POSIX applications, then, run transitively on top of our minimal interface by going through the rump kernel. Interestingly, the rump kernel hypercall interface and the Embassies minimal interface for applications are almost the same, although, at least to our knowledge, they were developed independently. The convenient implication of interface similarity is the ability to easily apply any security or other analysis made about Embassies to the rump kernel stack.

Fundamental Characteristics

We present the fundamental technical characteristics of rump kernels in this section. They are written more in the form of a dry list than a collection of juicy anecdotes and use cases. We feel that presenting the key characteristics in a succinct form will give a better understanding of both the possibilities and limitations of the rump kernel approach.

A rump kernel is always executed by the host platform. The details, including how that execution happens, and how many concurrent rump kernel instances the platform can support, vary on the platform in question. For user space, it's a matter of executing a binary. On Xen, it's a matter of starting a guest domain. On an embedded platform, most likely the bootloader will load the rump kernel into memory, and you would just jump to the rump kernel entry point.

The above is in fact quite normal; usually operating systems are loaded and executed by the platform that hosts them, be it hardware, virtual machine, or something else. The difference comes with application code. A kernel normally has a way of executing applications. Rump kernels contain no support for executing binaries to create runtime processes, so linking and loading the application part of the rump kernel software stack is also up to the host. For simplicity and performance, the application layer can be bundled together with the rump kernel (see, e.g., Figure 1). In user space, it is also possible to run the rump kernel in one process, with one or more applications residing in other processes communicating with the rump kernel (so-called "remote clients"). In both cases the applications are still linked, loaded, and executed by the host platform.

The notion of a CPU core is fictional. You can configure the number of "cores" as you wish, with some restrictions, such as the number must be an integer >0. For a rump kernel, the number of cores only signifies the number of threads that can run concurrently. A rump kernel will function properly no matter what the mapping between the fictional and physical cores is. However, if performance is the goal, it is best to map a rump kernel instance's fictional cores 1:1 to physical cores, which will allow the driver code to optimize hardware cache uses and locking.

Rump kernels do not perform scheduling. The lack of thread scheduling has far-reaching implications, for example:

- Code in a rump kernel runs on the platform's threads—nothing else is available. Rump kernels therefore also use the platform's thread-scheduling policy. The lack of a second scheduler makes rump kernels straightforward to integrate and control, and also avoids the performance problems of running a thread scheduler on top of another thread scheduler.
- Synchronization operations (e.g., mutex) are hypercalls because the blocking case for synchronization depends on invoking the scheduler. Notably, hypercalls allow optimizing synchronization operations for the characteristics of the platform scheduler, avoiding, for example, spinlocks in virtualized environments.

A less obvious corollary to the lack of a scheduler is that rump kernels use a "CPU core scheduler" to preserve a property that code expects: no more than one thread executing on a core. Maintaining this property in rump kernels ensures that, for example, passive synchronization (e.g., RCU, or read-copy-update) and lock-free caches continue to function properly. Details on core scheduling are available in the book [4].

Since core scheduling is not exposed to the platform scheduler, there are no interrupts in rump kernels, and once a rump kernel core is obtained, a thread runs until it exits the rump kernel or blocks in a hypercall. This run-to-completion mode of operation is not to be confused with a requirement that the platform scheduler must run the thread to completion. The platform scheduler is free to schedule and unschedule the thread running in a rump kernel as it pleases. Although a rump kernel will run correctly on top of any thread scheduler you throw under it, there are performance advantages to teaching the platform scheduler about rump kernels.

Rump kernels do not support, use, or depend on virtual memory. Instead, a rump kernel runs in a memory space provided by the platform, be it virtual or not. The rationale is simplicity and portability, especially coupled with the fact that virtual memory is not necessary in rump kernels. Leaving out virtual memory support saves you from having to include the virtual memory subsystem in a rump kernel, not to mention figuring out how to implement highly platform-dependent page protection, memory mapping, and other virtual memory-related concepts.

The more or less only negative effect caused by the lack of virtual memory support is that the mmap() system call cannot be fully handled by a rump kernel. A number of workarounds are possible for applications that absolutely need to use mmap(). For example, the bozohttpd Web server uses mmap() to read the files it serves, so when running bozohttpd on top of a rump kernel, we simply read the mmap'd window into memory at the time the mapping is made instead of gradually faulting pages in. A perfect emulation of mmap() is hard to achieve, but one that works for most practical purposes is easy to achieve.

Machine (In)Dependencies

Rump kernels are platform-agnostic, thanks to the hypercall layer. But can rump kernels be run literally anywhere? We will examine the situation in detail.

One limitation is the size of the drivers. Since NetBSD drivers are written for a general purpose OS, rump kernels are limited to systems with a minimum of hundreds of kB of RAM/ROM. One can of course edit the drivers to reduce their size, but by doing so one of the core benefits of using rump kernels will be lost: the ability to effortlessly upgrade to later driver versions in order to pick up new features and bug(fixe)s.

As for the capabilities of the processor itself, the only part of the instruction set architecture that permeates into rump kernels is the ability to perform cache-coherent memory operations on multiprocessor systems (e.g., compare-and-swap). In a pinch, even those machine-dependent atomic memory operations can be implemented as hypercalls—performance implications notwithstanding—thereby making it possible to run rump kernels on a generic C machine.

To demonstrate their machine independence, rump kernels were run through a C->Javascript compiler so that it was possible to execute them in Web browsers. Running operating systems in browsers previously has been accomplished via machine emulators written in Javascript, but with rump kernels the kernel code went native. If you have always wondered what the BSD FFS driver looks like when compiled to Javascript and wanted to single-step through it with Firebug, your dreams may have come true. The rest of us will probably find more delight in being amused by the demo [7] for a few minutes. And, no, the NetBSD kernel did not and still does not support the "Javascript ISA," but rump kernels do.

So, yes, you can run rump kernels on any platform for which you can compile C99 code and which has a minimum of some hundreds of kilobytes of RAM/ROM.

Virtual Uniprocessor and Locking

Avoiding memory bus locks is becoming a key factor for performance in multiprocessor environments. It is possible to omit memory bus locks almost entirely for rump kernels configured to run with one fictional core, regardless of the number of physical cores visible to the platform. This optimization is based on the property of the rump kernel CPU core scheduler. Since there can be at most one thread running within the rump kernel, there is no need to make sure that caches are coherent with other physical cores, because no other physical core can host a thread running in the same rump kernel. Appropriate memory barriers when the rump kernel core is reserved and released are enough. The fastpath for locking becomes a simple variable check and assignment that can fully be handled within the rump kernel. Only where the lock is already held does a hypercall need to be made to inform the scheduler.

This locking scheme can be implemented in a single file without touching any drivers. In the spirit of the project, the name of the Uniprocessor locking scheme was decided after careful consideration: *locks_up*. A scientific measurement of a POSIXy application creating and removing files on a memory file system showed a more than 30% performance increase with locks_up. The actual benefit for real-world applications may be less impressive.

From Syscalls to Application Stacks

First, we introduce some nomenclatural clarity. Since there are no hardware privilege levels or system traps in rump kernels, there are strictly speaking no system calls either. When we use the term "system call" or "syscall" in the context of rump kernels, we mean a routine which performs the service that would normally be executed via a kernel trap.

From nearly the beginning of this project, rump kernels have supported NetBSD-compatible system call interfaces. Compatibility exists for both API and ABI, apart from the distinction that rump kernel syscalls were prefixed with "rump_sys" to avoid symbol collisions with libc when running in user space. ABI compatibility meant that in user space it was possible to `LD_PRELOAD` a hijacking library so that most system calls were handled by the host, but some system calls—e.g., ones related to sockets—could be handled by rump kernels.

On a platform without an OS, this approach of course does not work: There is no OS that can handle a majority of the system calls. The solution was simple (see Figure 1): we took NetBSD's libc and built it without the syscall bits that caused kernel traps. We then removed the "rump_sys" prefix for the rump kernel syscall handlers, because there was no host libc to conflict with. Regular user-space libraries—i.e., everything apart from libc and libpthread—and applications require no modification to function on top of a rump kernel; they think they are running on a full NetBSD system.

Among the three factions, rump kernels currently support roughly two-thirds, or more than 200, of the system calls offered by NetBSD. Some examples of applications tested to work out-of-the-box on top of a rump kernel include thttpd, the LuaJIT compiler, and wpa_supplicant.

Interestingly, getting the full application stack working in user space required more effort than getting it to work in an environment without a host OS. This is because user space gets crowded: The rump kernel stack provides a set of symbols that can, and almost certainly will, conflict with the hosting OS's symbols. However, it turns out that with judicious symbol renaming and hiding it is possible to avoid conflicting names between the host OS and the rump kernel stack. Having the full application stacks work in user space allows you to compile and run NetBSD-specific user space code (e.g., ifconfig) against rump kernels on other operating systems. Listing 1 illustrates this in more detail.

Trying It Out

The easiest way to familiarize yourself with rump kernels is to do it in the comfort of user space by using the buildrump.sh script. Clone the repository at http://repo.rumpkernel.org/buildrump.sh.git and, on a POSIXy open source operating system, run:

```
./buildrump.sh
```

When executed without parameters, the script will fetch the necessary subset of the NetBSD source tree and build rump kernel components and the POSIXy user space implementation of the hypercall interface. Follow the build by running a handful of simple tests that check for example file system access, IPv4/IPv6 routing, and TCP termination. Running these tests under GDB in the usual fashion—buildrump.sh builds everything with debugging symbols by default—and single-stepping and using breakpoints is an easy way to start understanding how rump kernels work.

Since rump kernel stacks work the same way in user space as they do on an embedded IoT device, once you learn one platform you've more or less learned them all. The flipside of the previous statement also applies: When you want to debug some code for your embedded device, you can just debug the code in user space, presence of hardware devices notwithstanding.

Also make sure to note that if your host is running on desktop/server hardware of a recent millennium, the bootstrap time of a rump kernel is generally on the order of 10 ms.

Listing 1 offers an idea of the component-oriented quality of rump kernels and shows how easily you can configure them as long as you are familiar with standard UNIX tools. Further up-to-date instructions targeting more specific use cases are available as tutorials and how-tos on wiki.rumpkernel.org.

Run a rump kernel server accepting remote requests, set up client programs to communicate with it, and check the initial network configuration.

```
rumpremote (NULL)$ rump_server -lrumpnet_netinet
    -lrumpnet_net -lrumpnet unix://ctrlsock
rumpremote (NULL)$ export RUMP_SERVER=unix://
    ctrlsock
rumpremote (unix://ctrlsock)$ ifconfig -a
lo0: flags=8049 mtu 33648
    inet 127.0.0.1 netmask 0xff000000
```

Oops, we want IPv6, too. Let's start another rump kernel with IPv6, listening to requests at a slightly different address.

```
rumpremote (unix://ctrlsock)$ rump_server -lrumpnet_
    netinet6 lrumpnet_netinet -lrumpnet_net
    -lrumpnet unix://ctrlsock6
rumpremote (unix://ctrlsock)$ export RUMP_SERVER=
    unix://ctrlsock6
rumpremote (unix://ctrlsock6)$ ifconfig -a
lo0: flags=8049 mtu 33648
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
    inet 127.0.0.1 netmask 0xff000000
```

Better. We check that the original is still without IPv6, and see which file systems are mounted in the new one.

```
rumpremote (unix://ctrlsock6)$ env
    RUMP_SERVER=unix://ctrlsock ifconfig -a
lo0: flags=8049 mtu 33648
    inet 127.0.0.1 netmask 0xff000000
rumpremote (unix://ctrlsock6)$ mount
mount: getmntinfo: Function not implemented
```

Oops, we did not include file system support. We will halt the second server and restart it with file system support.

```
rumpremote (unix://ctrlsock6)$ halt
rumpremote (unix://ctrlsock6)$ rump_server -lrumpnet_
    netinet6 -lrumpnet_netinet -lrumpnet_net
    -lrumpnet -lrumpvfs unix://ctrlsock6
rumpremote (unix://ctrlsock6)$ mount
rumpfs on / type rumpfs (local)
rumpremote (unix://ctrlsock6)$
```

Listing 1: Example of rump kernels running in user space. The process rump_server contains kernel components. The utilities we use contain the application layers of the software stack. In user space, the two can communicate via local domain sockets. This model allows for very natural use. The output was captured on Ubuntu Linux. $PATH has been set so that NetBSD utilities that are running on top of the rump kernel stack are run.

Conclusion

We present rump kernels, a cornucopia of portable, componentized kernel-quality drivers such as file systems, networking drivers, and POSIX system call handlers. Rump kernels rely on the anykernel architecture inherent in NetBSD, and can be built from any vintage of the NetBSD source tree. The technology is stable, as far as that term can be used to describe anything related to operating system kernel internals, and has been developed in NetBSD since 2007.

Everything we described in this article is available as BSD-licensed open source via **rumpkernel.org**. Pointers to usual community-type elements for discussing use cases and contributions are also available from rumpkernel.org. We welcome your contributions.

References

[1] lwIP, a lightweight open source TCP/IP stack: http://savannah.nongnu.org/projects/lwip/.

[2] Genode Operating System Framework: http://genode.org/.

[3] DDEKit and DDE for Linux: http://os.inf.tu-dresden.de/ddekit/.

[4] *The Design and Implementation of the Anykernel and Rump Kernels*: http://book.rumpkernel.org/.

[5] Rump kernel hypercall interface manual page: http://man.NetBSD.org/cgi-bin/man-cgi?rumpuser++NetBSD-current.

[6] Embassies project: https://research.microsoft.com/en-us/projects/embassies/.

[7] Javascript rump kernel: http://ftp.NetBSD.org/pub/NetBSD/misc/pooka/rump.js/.

Sirius

Distributing and Coordinating Application Reference Data

MICHAEL BEVILACQUA-LINN, MAULAN BYRON, PETER CLINE, JON MOORE, AND STEVE MUIR

Michael Bevilacqua-Linn is a Distinguished Engineer at Comcast. He's worked on their next generation IP video delivery systems as an architect and engineer for the past several years, and is interested in functional programming and distributed systems. He currently resides in Philadelphia.
Michael_Bevilacqua-Linn@comcast.com

Maulan Byron is passionate about making software scalable, fast, and robust. He has spent the majority of his career building and delivering mid-size to large-scale projects in the telecommunication and financial industries. His interests are in making things scale and perform better and finding solutions that make it easier to operationalize software products.
Maulan_Byron@cable.comcast.com

Peter Cline is a senior software engineer at Comcast. In the past two years there, his work has focused on distributed systems, hypermedia API design, and automated testing. Prior to Comcast, he worked in search and digital curation at the University of Pennsylvania.
Peter_Cline2@comcast.com

Sirius is an open-source library that provides developers of applications that require reference data with a simple in-memory object model while transparently managing cluster replication and consistency. We describe the design and implementation of Sirius in the context of TV and movie metadata, but Sirius has since been used in other applications at Comcast, and it is intended to support a broad class of applications and associated reference data.

Many applications need to use *reference data*-information that is accessed frequently but not necessarily updated in-band by the application itself. Such reference data sets now fit comfortably in memory, especially as the exponential progress of Moore's Law has outstripped these data sets' growth rates: For example, the total number of feature films listed in the Internet Movie Database grew 40% from 1998 to 2013, whereas commodity server RAM grew by two orders of magnitude in the same period.

Consider the type of reference data that drove the design and implementation of *Sirius*: metadata associated with television shows and movies. Examples of this metadata include facts such as the year *Casablanca* was released, how many episodes were in Season 7 of *Seinfeld*, or when the next episode of *The Voice* will be airing (and on which channel). This data set has certain distinguishing characteristics common to reference data:

It is **small**. Our data set is a few tens of gigabytes in size, fitting comfortably in main memory of modern commodity servers.

It is relatively static, with a **very high read/write ratio.** Overwhelmingly, this data is write-once, read-frequently: *Casablanca* likely won't get a different release date, *Seinfeld* won't suddenly get new Season 7 episodes, and *The Voice* will probably air as scheduled. However, this data is central to almost every piece of functionality and user experience in relevant applications—and those applications may have tens of millions of users.

It is **asynchronously updated.** End users are not directly exposed to the latency of updates, and some propagation delay is generally tolerable, e.g., in correcting a misspelling of "*Cassablanca.*" However, if a presidential press conference is suddenly called, schedules may need to be updated within minutes rather than hours.

Common service architectures separate the management of reference data from the application code that must use it, typically leveraging some form of caching to maintain low latency access. Such schemes force developers to handle complex interfaces, and thus may be difficult to use correctly.

Sirius keeps reference data entirely in RAM, providing simple access by the application, while ensuring consistency of updates in a distributed manner. Persistent logging in conjunction with consensus and "catch up" protocols provides resilience to common failure modes and automatic recovery.

Sirius has been used in production for almost two years, and it supports a number of cloud services that deliver video to Comcast customers on a variety of platforms. These services must support:

Jon Moore, a technical fellow at Comcast Corporation, runs the Core Application Platforms group, which focuses on building scalable, performant, robust software components for the company's varied software product development groups. His current interests include distributed systems, hypermedia APIs, and fault tolerance. Jon received his PhD in computer and information science from the University of Pennsylvania and currently resides in West Philadelphia.
Jonathan_Moore@comcast.com

Steve Muir is a senior director, Software Architecture, at Comcast. He works on a broad range of cloud infrastructure projects, with a specific focus on system architectures and environments that support highly scalable service delivery. He has a broad background in operating systems, networking, and telecommunications, and holds a PhD in computer science from the University of Pennsylvania.
Steve_Muir@cable.comcast.com

Multiple datacenters. We expect our services to run in multiple locations, for both geo-locality of access and resilience to datacenter failures.

Low access latency. Interactive, consumer-facing applications must have fast access to our reference data: service latencies directly impact usage and revenue [3].

Continuous delivery. Our services will be powering products that are constantly evolving. Application interactions with reference data change, and we aim to be able to rapidly deploy code updates to our production servers. Hence, easy and rapid automated testing is essential.

Robustness. In production we expect to experience a variety of failure conditions: server crashes, network partitions, and failures of our own service dependencies. The application service must continue operating—perhaps with degraded functionality—in the face of these failures.

Operational friendliness. Any system of sufficient complexity will exhibit emergent (unpredictable) behavior, which will likely have to be managed by operational staff. Sirius must have a simple operational interface: It should be easy to understand "how it works," things should fail in obvious but safe ways, it should be easy to observe system health and metrics, and there should be "levers" to pull with predictable effects to facilitate manual interventions.

This article describes how the design and implementation of Sirius satisfies these requirements. Further technical details, additional performance evaluation results, and in-depth consideration of related work are covered in the associated full-length paper [1].

Approach

As we have seen, our reference data set fits comfortably in RAM, so we take the approach of keeping a complete copy of the data (or a subset) on each application server, stored in-process as native data structures. This offers developers ultimate convenience:

- No I/O calls are needed to access externally stored data, and thus there is no need to handle network I/O exceptions.
- Automated testing and profiling involving the reference data only requires direct interaction with "plain old Java objects."
- Developers have full freedom to choose data structures directly suited to the application's use cases.
- There are no "cache misses" since the entire data set is present; access is fast and predictable.

Of course, this approach raises several important questions in practice. How do we keep each mirror up-to-date? How do we restore the mirrors after an application server restarts or fails?

Update Publishing

We assume that external systems manage the reference data set and push updates to our server, rather than having our server poll the system of record for updates. This event-driven approach is straightforward to implement in our application; we update the native data structures in our mirror while continually serving client requests. We model this interface after HTTP as a series of PUTs and DELETEs against various URL keys.

Replication and Consistency

To run a cluster of application servers, we need to apply the updates at every server. The system of record pushes updates through a load balancer, primarily to isolate it from individual server failures, and members of the cluster are responsible for disseminating those updates to their peers in a consistent manner.

The CAP theorem dictates that in the event of a network partition, we will need to decide between availability and consistency. We need read access to the reference data at all times and will have to tolerate some windows of inconsistency. That said, we want to preserve at least eventual consistency to retain operational sanity, and can tolerate some unavailability of writes during a partition, as our reference data updates are asynchronous from the point of view of our clients.

To achieve this, our cluster uses a variant of the Multi-Paxos [2] protocol to agree on a consistent total ordering of updates and then have each server apply the updates in order. A general consensus protocol also allows us to consistently order updates from multiple systems of record. We provide more detail in the section on Replication.

Persistence

As with many Paxos implementations, each server provides persistence by maintaining a local transaction log on disk of the committed updates. When a server instance starts up, it replays this transaction log to rebuild its mirror from scratch, then rejoins the replication protocol described above, which includes "catch up" facilities for acquiring any missing updates.

Library Structure

Finally, Sirius is structured as a library that handles the Paxos implementation, persistence, log compaction, and replay. The hosting application is then conceptually separated into two pieces (Figure 1); its external interface and business logic, and its mirror of the reference data. This approach allows full flexibility over data structures while still offering an easy-to-understand interface to the developer.

Programming Interface

As we just described, Sirius' library structure divides an application into two parts, with Sirius as an intermediary. The application provides its own interface: for example, exposing HTTP endpoints to receive the reference data updates. The application then routes reference data access through Sirius.

After taking care of serialization, replication, and persistence, Sirius invokes a corresponding callback to a request handler provided by the application. The request handler takes care of updating or accessing the in-memory representations of the reference data. The application developers are thus completely in control of the native, in-memory representation of this data.

The corresponding programming interfaces are shown in Figure 2; there is a clear correspondence between the Sirius-provided access methods and the application's own request handler. As such, it is easy to imagine a "null Sirius" implementation that would simply invoke the application's request handler directly.

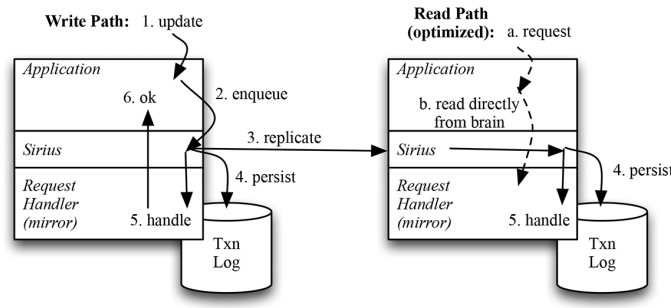


Figure 1: Architecture of a Sirius-based application

This semantic transparency makes it easy to reason functionally about the reference data itself.

The primary Sirius interface methods are all asynchronous; the Sirius library invokes request handlers in such a way as to provide eventual consistency across the cluster nodes. The overall contract is:

- The request handlers for PUTs and DELETEs will be invoked serially and in a consistent order across all nodes.
- Enqueued asynchronous updates will not complete until successful replication has occurred.
- An enqueued GET will be routed locally only, but will be serialized with respect to pending updates.
- At startup time, Sirius will not accept new updates or report itself as "online" until it has completed replay of its transaction log, as indicated by the isOnline method.

Sirius does not provide facilities for consistent conditional updates (e.g., compare-and-swap); it merely guarantees consistent ordering of the updates. Indeed, in practice, many applications do not use Sirius on their read path, instead reading directly from concurrent data structures in the mirror.

Replication

Updates passed to Sirius via `enqueuePut` or `enqueueDelete` are ordered and replicated via Multi-Paxos, with each update being a command assigned to a *slot* by the protocol; the slot numbers are recorded as sequence numbers in the persistent log. Our implementation fairly faithfully follows the description given by van Renesse [9], with some slight differences:

Stable leader. First, we use the common optimization that disallows continuous "weak" leader elections; this limits vote conflicts, and resultant chattiness, which in turn enhances throughput.

End-to-end retries. Second, because all of the updates are idempotent, we do not track unique request identifiers, as the updates can be retried if not acknowledged. In turn, that assumption means that we do not need to store and recover the internal Paxos state on failures.

```
public interface Sirius {
  Future<SiriusResult>
    enqueueGet(String key);
  Future<SiriusResult>
   enqueuePut(String key, byte[] body);
  Future<SiriusResult>
    enqueueDelete(String key);
 boolean isOnline();
}

public interface RequestHandler {
  SiriusResult handleGet(String key);
  SiriusResult handlePut(String key,
                    byte[] body);
  SiriusResult handleDelete(String key);
}
```

Figure 2: Sirius interfaces. A SiriusResult is a Scala case class representing either a captured exception or a successful return, either with or without a return value.

Similarly, we bound some processes, specifically achieving a quorum on the assignment of an update to a particular slot number, with timeouts and limited retries. During a long-lived network partition, a minority partition will not be able to make progress; this limits the amount of state accumulated for incomplete writes. Sirius thus degrades gracefully, with no impact on read operations for those nodes, even though their reference data sets in memory may begin to become stale.

Write behind. Nodes apply updates ("decisions" in Paxos terminology) in order by sequence number, buffering any out-of-order decisions as needed. Updates are acknowledged once a decision has been received, but without waiting for persistence or application to complete; this reduces system write latency and prevents "head-of-line" blocking.

However, this means that there is a window during which an acknowledged write can be lost without having been written to stable storage. In practice, since Sirius is not the system of record for the reference data set, it is possible to reconstruct lost writes by republishing the relevant updates.

Catch-up Protocol

Because updates must be applied in the same order on all nodes, and updates are logged to disk in that order, nodes are particularly susceptible to lost decision messages, which delay updates with higher sequence numbers. Therefore, each node periodically selects a random peer and requests a range of updates starting from the lowest sequence number for which it does not have a decision.

The peer replies with all the decisions it has that fall within the given slot number range. Some of these may be returned from a small in-memory cache of updates kept by the peer, especially if the missing decision is a relatively recent one. However, the peer may need to consult the persistent log for older updates no longer in its cache (see the section on Persistence). This process continues until no further updates need to be transmitted.

The catch-up protocol also supports auxiliary cluster members that do not participate in Paxos. Primary cluster members know about each other and participate in the consensus protocol for updates. Secondary cluster members periodically receive updates from primary nodes using the catch-up protocol. In practice, this allows a primary "ingest" cluster to disseminate update to dependent application clusters, often within seconds of each other and across datacenters.

In turn, this lets us keep write latencies to a minimum: Paxos only runs across local area networks (LANs). Different clusters can be activated as primaries by pushing a cluster configuration update, which the Sirius library processes without an application restart.

This leads to a large amount of topology flexibility: Figure 3 shows how four clusters A–D can be given different configuration files in order to control distribution of updates. Only A participates in the Paxos algorithm, while B and C directly follow A, and D follows both B and C.

Persistence

As updates are ordered by Paxos, Sirius also writes them out to disk in an append-only file. Each record includes an individual record-level checksum, its Paxos sequence number, a timestamp (used for human-readable logging, not for ordering), an operation code (PUT or DELETE), and finally a key and possibly a body (PUTs only). This creates variable-sized records, which are not ordinarily a problem: The log is appended by normal write processing and is normally only read at application startup, where it is scanned sequentially anyway.

However, there is one exception to this sequential access pattern: While responding to catch-up requests, we need to find updates no longer cached, perhaps because of a node crash or long-lived network partition. In this case, we must find a particular log entry by its sequence number.

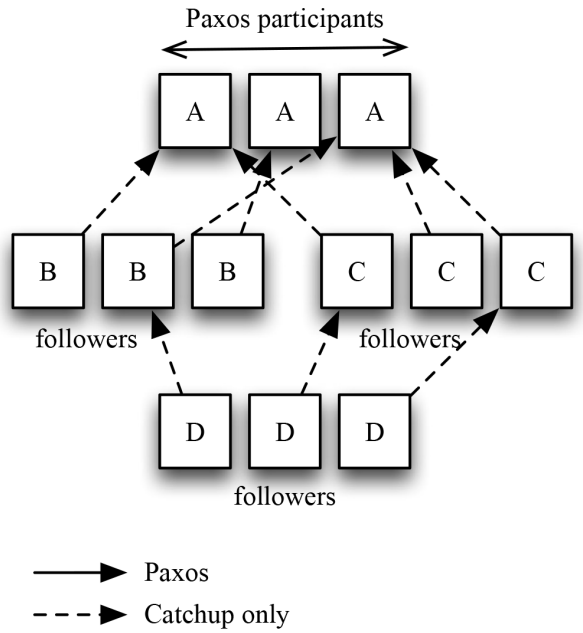


Figure 3: Flexible replication topologies with Sirius

This is accomplished by creating an index structure during the process of reading the update log at startup time. The index is small enough to be stored in memory and can thus be randomly accessed in an efficient manner, permitting use of binary search to locate a particular update by sequence number.

Sirius can *compact* its log file: because the PUTs and DELETEs are idempotent, we can remove every log entry for a key except the one with the highest sequence number. Because the overall reference data set does not grow dramatically in size over time, a compacted log is a relatively compact representation of it; we find that the reference data set takes up more space in RAM than it does in the log once all the appropriate indices have been created in the application's mirror. This avoids the need for the application to participate in creating snapshots or checkpoints, as in other event-sourced systems [2].

Early production deployments of Sirius took advantage of rolling application restarts as part of continuous development to incorporate offline compaction of the persistent log. However, frequent restarts were required to prevent the log from getting unwieldy.

Therefore, we developed a scheme for live compaction that Sirius manages in the background. The log is divided into segments with a bounded number of entries, as in other log-based systems [7, 8]. Sirius appends updates to the highest-numbered segment; when that segment fills up, its file is closed and a new segment is started.

Compaction is accomplished by using the most recent log segment to create a set of "live" key-value pairs and deleted keys. Prior log segments can then be pruned by removing updates that would be subsequently invalidated, while updates to other keys are added to the live set. After compaction of an individual segment, the system combines adjacent segments when doing so does not exceed the maximum segment size.

Live compaction in Sirius is thus incremental and restartable and does not require a manual operational maintenance step with a separate tool. Since the logs are normal append-only files, and compaction is incremental, copies can be taken while the application is running without any special synchronization. We have taken advantage of this to bootstrap new nodes efficiently, especially when seeding a new datacenter, or to copy a production data set elsewhere for debugging or testing.

Experimental Evaluation

The optimized read path for an application bypasses Sirius to access reference data directly, so we are primarily interested in measuring write performance. In practice Sirius provides sufficient write throughput to support our reference data use cases, but here we present experimental analysis.

The Sirius library is written in Scala, using the Akka actor library. All experiments were run on Amazon Web Services (AWS) Elastic Computer Cluster (EC2) servers running a stock 64-bit Linux kernel on *m1.xlarge* instances, each with four virtual CPUs and 15 GB RAM. These instances have a 64-bit OpenJDK Java runtime installed; Sirius-based tests use version 1.1.4 of the library.

Write Throughput

For these tests, we embed Sirius in a reference Web application that exposes a simple key-value store interface via HTTP and uses Java's `ConcurrentHashMap` for its mirror. Load is generated from separate instances running JMeter version 2.11. All requests generate PUTs with 179 byte values (the average object size we see in production use).

We begin by establishing a baseline under a light load that establishes latency with minimal queueing delay. We then increase load until we find the throughput at which average latency begins to increase; this establishes the maximum practical operating capacity. Our results are summarized in Figure 4.

This experiment shows write throughput for various cluster sizes; it was also repeated for a reference application with a "null" RequestHandler (Sirius-NoBrain) and one where disk persistence was turned off (Sirius-NoDisk). There are two main observations to make here:

Throughput degrades as cluster size increases, primarily due to the quorum-based voting that goes on in Paxos: In larger clusters there is a greater chance that enough machines are sporadically running "slow" (e.g., due to a garbage collection pause) to slow

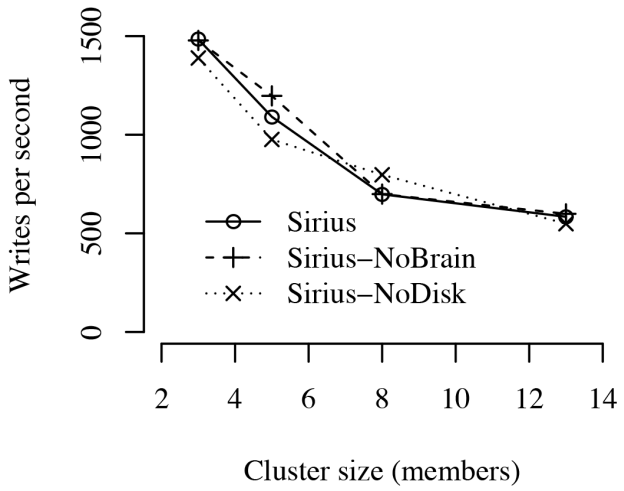


Figure 4: Sirius write throughput

down the consensus algorithm, as reported by Hunt et al. for ZooKeeper [4].

Throughput is not affected by disabling the persistence layer or by eliminating RequestHandler work; we conclude that the Paxos algorithm (or our implementation of it) is the limiting factor.

Operational Concerns

In addition to providing a convenient programming interface, we designed Sirius to be operationally friendly. This means that major errors, when they occur, should be obvious and noticeable, but it also means that the system should degrade gracefully and preserve as much functionality as possible. Errors and faults are expected, and by and large Sirius manages recovery on its own; however, operations staff may intervene if needed.

For example, operational intervention is helpful but not required in bringing a server online, either as a new cluster member or after recovering from a failure. The server may be far behind its active peers, and may have a partial or empty log. The catch-up protocol can be used to fetch the entire log, if necessary, from a peer, which can be accomplished in a few minutes for several gigabytes of log. However, operators can accelerate the process by copying an active node's log files, thus "seeding" the new server's state.

To support debugging and operations, we distribute a command-line tool along with Sirius. This tool reads and manipulates the Sirius log, providing functionality, including: format conversion, pretty printing log entries, searching via regular expression, and offline compaction. It also allows updates to be replayed as HTTP requests sent to a specific server.

Updates in the index and data files are checksummed. When corruption occurs and is detected Sirius will refuse to start. Currently, recovery is manual, albeit straightforward: Sirius reports the point at which the problematic record begins. An operator can truncate the log at this point or delete a corrupted index, and Sirius can take care of the rest, rebuilding the index or retrieving the missing updates as needed.

Conclusions and Future Work

Sirius has been deployed in support of production services for approximately two years, with very few operational issues. The library's simple and transparent interface, coupled with the ease and control of using native data structures, have led multiple independent teams within Comcast to incorporate Sirius into their services, all to positive effect. Nevertheless, we have identified some opportunities for improvements.

- The current Multi-Paxos-based consensus protocol limits write throughput to the capacity of the current leader; this could be alleviated by an alternative protocol, such as Egalitarian Paxos [5].
- Cluster membership updates are not synchronized with the consensus protocol. Consensus protocols like RAFT [6] that integrate cluster membership with consensus could simplify operations.
- WAN replication currently piggybacks on our cluster catch-up protocol, requiring one-to-one transfers of updates. A topology-aware distribution of updates could be beneficial in reducing bandwidth usage.
- Replaying write-ahead logs synchronously and serially at startup takes a significant time; hence a mechanism for safely processing some updates in parallel is desirable.

All things considered, the design and implementation of Sirius has been very successful. We would like to acknowledge the CIM Platform/API team at Comcast, Sirius' first users and development collaborators; Sirius would not have been possible without your help and hard work.

Sirius is available under the Apache 2 License from http://github.com/Comcast/sirius.

References

[1] M. Bevilacqua-Linn, M. Byron, P. Cline, J. Moore, and S. Muir, "Sirius: Distributing and Coordinating Application Reference Data," in *Proceedings of the USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX Association.

[2] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos Made Live: An Engineering Perspective," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2007), PODC '07, ACM, pp. 398-407.

[3] T. Hoff, "Latency Is Everywhere and It Costs You Sales—How to Crush It," *High Scalability* blog, July 2009: http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it, accessed January 20, 2014.

[4] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-Free Coordination for Internet-Scale Systems," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2010), USENIX ATC '10, USENIX Association, pp. 145–158.

[5] I. Moraru, D. G. Andersen, and M. Kaminsky, "There Is More Consensus in Egalitarian Parliaments," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 358-372.

[6] D. Ongaro and J. K. Ousterhout, "In Search of an Understandable Consensus Algorithm," October 2013: https://ramcloud.stanford.edu/raft.pdf, accessed January 13, 2014.

[7] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Comput. Syst.,* vol. 10, no. 1 (Feb. 1992), 26–52.

[8] J. Sheehy, and D. Smith, "Bitcask: A Log-Structured Hash Table for Fast Key/Value Data": http://downloads.basho.com/papers/bitcask-intro.pdf, accessed January 16, 2014.

[9] R. van Renesse, "Paxos Made Moderately Complex": http://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf, March 2011, accessed January 13, 2014..

Making "Push on Green" a Reality

DANIEL V. KLEIN, DINA M. BETSER, AND MATHEW G. MONROE

Daniel Klein has been a Site Reliability Engineer at Google's Pittsburgh office for 3.5 years. His goal is to automate himself out of a job, so that he can get on with the work of looking for new things to do at Google. Prior to Google, he bored more easily and did a myriad different things (look on the Web for details). dvk@google.com

Dina Betser is a Site Reliability Engineer who has worked on projects such as Google Calendar and Google's large machine learning systems that maintain high quality ads. As an SRE, she often works on ensuring that products behave reliably with as little manual intervention as possible. She studied computer science as an undergraduate and master's student at MIT. dinabetser@google.com

Mathew Monroe is a Site Reliability Engineer who has worked on both the payments and anti-malvertising systems at Google. When not trying to make the Internet a safer and better place, he is trying to make running Internet services a magical experience. He has a master's in software engineering from Carnegie Mellon University and worked in distributed file systems and computer security before coming to Google. onet@google.com

Updating production software is a process that may require dozens, if not hundreds, of steps. These include creating and testing new code, building new binaries and packages, associating the packages with a versioned release, updating the jobs in production datacenters, possibly modifying database schemata, and testing and verifying the results. There are boxes to check and approvals to seek, and the more automated the process, the easier it becomes. When releases can be made faster, it is possible to release more often, and, organizationally, one becomes less afraid to "release early, release often" [6, 7]. And that's what we describe in this article—making rollouts as easy and as automated as possible. When a "green" condition is detected, we can more quickly perform a new rollout. Humans are still needed somewhere in the loop, but we strive to reduce the purely mechanical toil they need to perform.

We, Site Reliability Engineers working on several different ads and commerce services at Google, share information on how we do this, and enable other organizations to do the same. We define "Push on Green" and describe the development and deployment of best practices that serve as a foundation for this kind of undertaking. Using a "sample service" at Google as an example, we look at the historical development of the mechanization of the rollout process, and discuss the steps taken to further automate it. We then examine the steps remaining, both near and long-term, as we continue to gain experience and advance the process towards full automation. We conclude with a set of concrete recommendations for other groups wishing to implement a Push on Green system that keeps production systems not only up-and-running, but also updated with as little engineer-involvement and user-visible downtime as possible.

Push on Green

A common understanding of Push on Green is "if the tests are good, the build is good, go push it!" but we define Push on Green in three distinct ways:

1. A pushmaster says "this build is ready to go—push it." The criteria for this judgment may be based on a predefined push/freeze schedule, may have political or compliance-related considerations, may need to be coordinated with other projects, etc. Automated testing may occur, but the human is the ultimate arbiter.
2. In a continuous-build system (also known as "continuous deployment" or "continuous delivery"), a collection of smoke tests (simple tests that examine high-level functionality) and regression tests for a current build all pass at a given revision. That revision is "green" and may be pushed to production. The testing framework is the ultimate arbiter.
3. A change to a configuration file is made (which may enable or disable an existing feature, alter capacity or provisioning, etc.). This rollout may likely reuse an already green build, so the incremental tests and approvals are substantially simpler, and the reviewers and the testing framework are together the arbiters.

Other definitions are certainly possible (including the current state of the production system, so that we can consider a green-to-green transition), but above are the three that we use in this article. In all cases, we consider a system supported by Site Reliability Engineers (SRE) who are responsible for both the manual steps and the construction of the automated processes in the rollout.

Development and Deployment Best Practices

With the complexity and interconnectedness of modern systems, some development/rollout best practices have evolved which attempt to minimize problems and downtime [2, 3]. To better understand the issues involved in creating a Push on Green system, an outline of a typical Google development environment and deployment process provides a useful introduction.

Development

All code must be peer reviewed prior to submitting to the main branch to ensure that changes make sense, adhere to the overall project plan, and that bug fixes are sanely and reasonably implemented. All changes must be accompanied by tests that ensure the correct execution of the code both under expected and unexpected conditions [5]. As new libraries, APIs, and standards are introduced, old code is migrated to use them. To provide as clean and succinct an interface as possible for developers, libraries are updated and old APIs are removed as new ones are introduced [8]. The next push after a library update, then, has the same chance of breaking production as a local developer change.

The at-times draconian review process can slow down release of new code, but it ensures that whatever code is released is as likely as possible to perform as desired. And, because we live in the real world, we also extensively test our code.

Tests

Everyone at Google uses tests—the developers have unit-level, component-level, and end-to-end tests of the systems they write in order to verify system correctness. SREs have deployment tests and may call upon other tests to ensure that the newly rolled-out production system behaves the same way as it did in a testing environment. Occasionally, tests are simply a human looking at the graphs and/or logs and confirming that the expected behavior is indeed occurring. Running a production service is a compromise between an ideal world of fully testable systems and the reality of deadlines, upgrades, and human failings [7].

When developing code, all of the existing tests must continue to pass, and if new functional units are introduced, there must also be tests associated with them. Tests should guarantee that not only does the code behave well with expected inputs, but also behaves predictably with unexpected inputs.

When a bug is found, the general rule is that test-driven development is favored. That is, someone first crafts a test that triggers the buggy behavior; then the bug is fixed, verifying that the previously failing test no longer fails. The notion of "fixing the bug" may simply mean "the system no longer crashes," but a better, more laudable behavior is "appropriately adjusts for the erroneous input" (e.g., logging the problem, correcting or rejecting the data, reporting the problem back to the developers for further debugging, etc.).

We acknowledge that mistakes happen and that they happen all the time. When someone makes a mistake that adversely affects production, it becomes their responsibility to lead the postmortem analysis to help prevent future occurrences. Sometimes, a fix can be made that checks for mistakes before they happen, and at other times, changes to the underlying assumptions or processes are put into effect. For example, it was assumed that adding a column to a certain database would be harmless. A postmortem discovered that a rollback of software also required a rollback of that database, which lost the data in the new compliance-required column. This resulted in a change in procedure, where schema changes were made visible in the release prior to the one in which the code changes are visible, making rollbacks separable.

Monitoring

At Google, we extensively monitor our services. Using monitoring data, we continually strive to make our services better and to notice, through alerting, when things go wrong.

The most effective alerting looks for symptoms (and not their causes), allowing the human in the loop to diagnose the problem based on symptoms. While extensive monitoring provides great insights into the interrelation of various components of the system, ultimately all alerting should be performed in defense of a service level agreement (SLA), instead of trying to predict what may cause a service level failure [1]. Real world failures have a way of setting their own terms and conditions, and by setting (and monitoring) appropriate SLAs, it is possible to notify on "failure to meet expectations." After SLA-based alerting has been triggered, extensive monitoring enables drill-down and detailed root-cause analysis.

When this style of monitoring and alerting is in place, then not only is it possible to alert under extraordinary circumstances (e.g., surges in activity or failure of a remote system), but it is also possible to alert quickly when a new version of the software is unable to meet the demands of ordinary circumstance. Thus, an automated rollout procedure can easily incorporate monitoring signals to determine whether a rollout is good or whether the system should be rolled back to a previous version.

Updates and Rollbacks

Rolling out a new version of software is often coordinated under the supervision of a pushmaster, and may involve updating a single job, a single datacenter, or an entire service. Where possible, canaries are used to test proper functioning of revised software. Named for the proverbial canary in a coal mine, the canary instances are brought up before the main job is updated, in one last pre-rollout test of the software. The canary typically receives only a small fraction (perhaps 1% or 0.1%) of production traffic; if it fails or otherwise misbehaves, it can be quickly shut off, leaving the rest of the code as-yet not updated, and returning the service to normal operation for all users. Canarying can also be done in stages, per job, by orders of magnitude, by datacenter, or by geographic region served.

Services handle updates in a few different ways. Disparate code changes must be integrated into "builds" (where the binaries are created from various sources and libraries), and the timing of the release of these builds is often planned well in advance. A production binary not only comprises the directly edited code of the team, but also those libraries released by teams that run supporting services utilized by the service. Many .jar/.so files are statically associated into a binary package, and there is no universally followed release cycle; each team produces new versions on their own timetable. Therefore, whenever a new binary is built for release, the changes that comprise it may come from far and wide.

Configuration changes are also considered rollouts. These may be in the form of runtime flags specified on the command line, or in configuration files read on startup; both require a job to be restarted to take effect. There may also be database updates or changes that impact the behavior of a system without restarting it. Configuration changes have the same potential to induce failure, but they also benefit from the same types of automation.

Safely Introducing Changes

Consider how you would add a new feature to a service. One common practice incorporates the following steps:

1. Create a new runtime configuration directive for a new feature, with the default value set to "disabled." Write the code that uses the new feature, and guard that code with the new directive (so that the new code is present but is disabled by default).
2. Release the new binaries with no references to the new directive in any configuration file. The feature should remain inactive, and failed rollout requires a rollback to the previous version of the binaries.
3. Update the configuration files to include the presence of the new directive (but explicitly specify that it is disabled), and restart the current system. The feature should continue to remain inactive, and a failed rollout simply requires a rollback to the previous version of the configuration files.
4. Update the system configuration files to enable the new directive in the canary jobs only, and restart the current version of the binaries in the canaries. A failed rollout simply requires turning off the canaries and later rolling back to the previous version of the configuration files.
5. Update the remainder of the jobs with the directive enabled. Failures are less likely at this stage since the canaries have not died or caused problems, but failure simply requires a rollback to the previous version of the configuration files. At this point, the new feature is enabled.
6. In a subsequent release, alter the code so that the directive is now enabled by default. Because the directive is currently enabled in the configuration file, changing the default flag value to match the specified configuration value should have no effect on behavior, so rolling out this change is usually deferred to occur along with a collection of other changes. However, a failed rollout requires a rollback to the previous version of the binaries.
7. Update the system configuration files to make no further reference to the directive—it is "live" by default. A failed rollout simply requires a rollback to the previous version of the configuration files.
8. Edit all conditional execution of code to always execute, since that is now the implicit behavior. A failed rollout requires a rollback to the previous version of the binaries.
9. Delete the now-unused definition of the directive in the code. A failed rollout at this stage is almost certainly due to a configuration file error, because the directive itself should not have been used since step 7—so a binary rollback is probably not needed.

Requiring nine steps to fully add a new feature may seem like overkill, but it ensures the safe release of new code. Additionally, the steps involved can take place over many months and many disparate releases. Complicating this process is the fact there may be dozens of such changes occurring in parallel, some simultaneously in-flight but starting and ending at widely different times. Automating as much of the rollout process as possible can help mitigate the overhead of keeping track of changes.

Types of Configuration Changes

We consider two kinds of configuration changes:

1. Changes to configuration directives that require job restart.
2. Changes to configuration directives that are automatically picked up by jobs.

There are advantages and disadvantages to both. When job-restart is required, one type of job can be updated with the configuration directive, regardless of which other jobs have the

directives available to them. This yields fine-grained control, but also requires that all restarts be tightly coordinated, so that an unrelated job restart does not pick up unintended configuration changes.

When jobs automatically pick up changes, configuration changes are more global in scope. While this has the advantage of easily automating changes on a large scale, it also means that greater care must be taken in hierarchically specified configuration-files to ensure that only the intended jobs are changed. In a real-world system with thousands of options across hundreds of jobs, it is easy for the hierarchy to break down or become unmanageable, riddled with special cases.

In both cases, great care must be taken to restrict the inadvertent propagation of unintended changes. Simplicity and flexibility are at odds using either scheme, while reliability and configurability are the goal of both.

Towards Push on Green

Much of the danger in releasing new code can be mitigated, but the process still has a large amount of mechanical human oversight, and the purpose of the Push on Green project is to minimize as much of this toil as possible.

Historical State of the Practice

We begin by examining a "sample service" at Google. The rollout process starts with a push request being filed against the current on-call, detailing the parameters of the rollout (the version number, people involved, and whether the push is for canary, production, or some other environment).

Previously, this service had a largely manual rollout process, comprising a collection of scripts that were manually invoked following a rollout run-book. The first step towards Push on Green was to replace this with a more automated process that effectively performed the same steps.

For the production jobs, the following steps are executed for binary pushes or command-line flag changes. The automated rollout procedure updates the push request at each step.

1. Silence alerts that will be known to fire during a rollout (for example, warnings about "mixed versions in production").
2. Push the new binaries or configuration changes to the canary jobs in a datacenter currently serving traffic.
3. Run the smoke tests, probers, and verify rollout health.
   a. If the tests fail, notify the on-call, who may initiate a canary rollback or bring down the canaries.
   b. Some health-check failure conditions are the result of an accumulation of errors, so some services require that tests can only pass after a sufficient amount of time is allowed for the binary to "soak."
4. Push the binaries to the remainder of the jobs in that datacenter.
5. Unsilence the previously silenced alerts.
6. Rerun smoke tests (step 3); if the tests pass, repeat steps 2–5 for each of the other datacenters.

This process still entails a lot of manual work. A push request must be filed for each rollout, and the binaries for each of the jobs must be built. The binary packages must be tagged, annotated, or accounted for in some way (so that the rollout pushes the correct binaries), and there are assorted handoffs between the release engineer, test engineers, and Site Reliability Engineers that limit the number of rollouts per day to only one or two. Although alerting in case of problems is largely automated, the entire push process must still be baby-sat to ensure correct functioning.

State of the Art—Recent Developments

Once the rollout process was made to be a largely push-button operation, steps were taken to make it more automated with even fewer buttons to push. These steps included:

RECURRING ROLLOUTS FOR INFRASTRUCTURE JOBS

Our services consist of jobs that are specific to the operation of the service and jobs and software packages that are maintained by other teams but that we configure for our service. For example, the monitoring and alerting jobs are standardized jobs that are custom-configured. The monitoring teams update the binaries that are available to use, but it is the responsibility of each service to periodically restart their jobs with new binaries at a time that is safe for the service involved.

Our recurring rollout updates those jobs maintained by other teams on a daily basis, keeping them current, even when there are service-specific production freezes. This recurring rollout was the first step to Push on Green automation.

ROLLOUT LOCKING

Some rollout steps have the potential to interfere with other rollouts. For example, if we are doing a production rollout, we do not want to simultaneously do an infrastructure rollout, so that we know which rollout to blame in case of a problem. With inter-rollout locking, we can also provide an inter-rollout delay, so that the effects of each rollout are clearly delineated from each other.

ROLLOUT REDUNDANCY

Reliability of the rollout system is just as important as reliability of the system being supported. A rollout that fails part way through due to a failure in the rollout automation can leave a production service in an unstable and unknown state. As such, the rollout systems should run in a replicated production configuration to guard against failure.

TIME RESTRICTIONS
We have on-call teams spanning multiple continents and separated by anywhere between five and ten time zones. The next step towards Push on Green was to provide a rollout schedule that took into account waking hours, weekends, and holidays (with different cultural norms). The software that recognizes these holidays was made to be easily configurable so other teams could reuse it for similar automation that includes other countries.

ROLLOUT MONITORING
The on-call must often consult a collection of logs to determine when a rollout started and ended, and attempt to correlate that data with problems that are reflected in latency and availability graphs.

Push on Green avoids manual searches of disparate sources of information, so another automation component was creating variables in the rollout system that could be queried by the monitoring and alerting system. This has enabled us to overlay a graph that visually displays the start and end of rollout components on top of the latency and availability graphs, so it is easy to see whether an inflection point in a graph exactly corresponds to a rollout.

AUTOMATIC ROLLOUT OF CONFIGURATION CHANGES
Adding a new configuration option requires nine discrete steps, and half of these are manual processes. The next step in automation is to have a single recurring rollout simply look for changes in the revision control system which match two specific criteria:

- Affect a specific set of files
- Have approvals from the right people

The rollout then automatically creates and annotates a new push request, and processes the rollout steps. When this is combined with rollout locking and time restrictions, we have an automatic Push-On-Green system (according to our third definition in "What is Push on Green"), dramatically reducing engineer toil. This cautious first step does not eliminate the human component of arbitration but, instead, removes much of the checklist labor that needs to be done.

State of the Art—Future Plans
Some of what follows is work in progress, and some of it is still in the planning stages, but all of it logically follows the work that has been accomplished so far in that it advances the automation of our rollout processes.

- Rollback-feasibility rollout step: use the testing environment to roll out a new binary, then roll it back after some traffic has been routed to the new jobs. If the smoke and regression tests still confirm job health, then the rollout can safely proceed in production jobs.
- Continuous delivery: automatically create push requests for versioned builds that pass the required tests, taking the "push early, push often" philosophy to its logical extreme. We can then use the monitoring data in the staging environment to ascertain which builds we believe are safe to push to production.
- Rollout quotas: we may want to limit the number of rollouts per day, or limit the number of rollouts that a single user can initiate, etc.
- Pushing Green-on-Green: perform a correlative analysis of a collection of indicators to determine overall system health before performing a new push. The system may not be out of SLA, but it might be dangerously close to the edge. Unless a service is currently green, it is a bad idea to automatically perform a new rollout.

We Still Need Manual Overrides!
Regardless of how much we want to automate everything, some processes will stay manual for now:

- We will always need the ability to do a manually induced rollback.
- Some tests are flaky. It may be the case that a build is not green due to a flaky test or that the system is healthy but the tests say otherwise. We need to be able to manually override the requirement for "green"; sometimes we believe that the problem being reported does not exist, or the problem exists but the rollout will fix it.
- Every automated system needs a big red button to halt execution. If we have the required means of switching off automatic rollouts, then we still need a way to do manual rollouts of any kind.

Real Numbers, Real Improvement
Since introduction of Push on Green, the on-calls in our service have experienced the improvements seen in Table 1.

We have increased the number of rollouts by an order of magnitude in two years, while at the same time saving almost a whole SRE FTE (and freeing the developers from much of their involvement in rollouts). Once our rollout system begins automatically detecting green conditions, we expect that the number of rollouts will increase even more, and the level of engineer engagement will continue to decrease.

Towards Push on Green: Recommendations
The fundamental components of an automated rollout system are as follows:

Monitoring and Alerting
If you don't know how your system is behaving in a measurable way, you only have assumptions to deal with. While naked-eye observation can tell you the gross symptoms ("the page loads

	Original (manual) rollouts 2012	Mechanized rollouts 2013	Semi-automated rollouts 2014
Rollouts/ month	12–20	60	160
Time saved for on-call/ month	0 hours (baseline)	20 hours	50–60 hours

Table 1: Rollouts have increased by an order of magnitude over two years, while time spent on them has decreased.

slowly" or "that looks better"), you need automated monitoring at a fine-grained component level to give you insights as to *why* problems are happening or whether changes have had the desired effect.

- Monitoring needs to be performed on production systems, and monitoring is different from profiling. Profiling helps you find hotspots in your code with canned data; monitoring tracks the responsiveness of a running system with live data.
- Monitoring needs to be combined with alerting so that exceptional conditions are rapidly brought to the attention of production staff. Although it is possible to eyeball a collection of graphs [4], a well-monitored system has far more variables than any human can reasonably scan by eye (in a Google production service, it is not unusual to monitor millions of conditions with tens of thousands of alerting rules).
- The state of monitored variables and resulting alerts must be available in a form that allows programmatic queries, so that external systems can determine the current and previous state. Both these data points are needed to make determinations as to whether things have improved or degraded.
- If at all possible, separately monitor canary versions of jobs and their non-canary production counterparts. If all other things are equal (traffic, load, data sets, etc.), then it is possible to assess the health and quality of a canary job relative to the previous version of production.

Builds and Versions

A repeatable mechanism for building new binary releases must be part of the overall release cycle. Static builds ensure that what you build and test today is exactly the same as what you push next week. While different components may have different build procedures, they all must be regularized into some standard format.

- Versions must be tracked, preferably in a way that makes it easy for both the developers and release engineers to correlate a given build with a specific production service. Rather than sequential version numbers (like v3.0.7.2a), we recommend versions that incorporate the date and some other distinguishing nomenclature (such as tool_20140402-rc3) so that a human can readily correlate versions.
- Versions should be tagged, annotated, or otherwise consistently accounted for. This means that rather than "push version X to production," you should "mark version X with the production tag" and then "push the current production version." This allows for separation of responsibilities (developers build releases, release engineers tag them, and production engineers update the jobs) while still maintaining a coherent view of the service.
- Builds should be tested at a number of levels, from unit tests through to end-to-end tests and live production tests. Finding problems earlier in the process helps eliminate them faster.

Scripted and Parameterized Rollouts and Rollbacks

A systematized and regularized rollout procedure must exist. If automated steps are interspersed with manual steps, there must also be checks to ensure that all of the manual steps have been properly performed.

- As many steps in the rollout process as possible should be fully automated. If customizations need to be done on a per-rollout basis, these should be specified in a configuration file so that nothing is left to the memory of the person starting the rollout. This is especially important when rolling back to some previous version.
- Rollouts steps (and thus the entire rollout) should be idempotent. A rollout should not suffer from any step being performed twice. If a rollout fails, rerunning the rollout should either fail in the same way or cure the problem.
- A rollback should be the same as a rollout to a previous version. This is more of an ideal goal than a practical reality—there will always be some special case where a schema change cannot be rolled back. However, the more regular the rollout process, the less likely this will happen, and the more likely it is that developers will avoid changes that cannot be rolled back.

Process Automation and Tracking

Once the basic infrastructure is in place for scripted rollouts, you can contemplate automatically detecting and rolling out new versions.

- Once the process of versioning has been regularized, you can query the build system to determine when a new version is available, and roll it out when the appropriate preconditions have been met (i.e., build and tests are green, the version number is incrementally larger, production is green, etc.).
- When a new version is rolled out, the monitoring and alerting should be queried at various stages in the rollout to ascertain

whether there are any problems. If any anomalous behavior is detected, the rollout should be paused at whatever state it is in until a determination of overall rollout health can be made (since rollouts should be idempotent, it is also valid to abort the rollout, with the expectation that it can be restarted later).

- An anomaly in your monitored data may be the responsibility of your just-pushed system, but it may also be the result of some dependent system having been coincidentally just pushed. Coordinating rollouts with dependent teams can avoid this problem.

Conclusions

Building our Push on Green rollout system has been an evolutionary process. It has worked well because of the great degree of caution that we have exercised in incrementally adding functionality and automation. Although we are all in favor of having computers do our jobs for us, we are also averse to the disasters that mindless automation can bring.

We are only one of many teams at Google who are automating their rollout process. The needs of the project and the constraints of the production environments influence how each of these teams perform their jobs. However, regardless of the particulars, each group has addressed the same concerns and exercised the same degree of caution in handing over the reins to an automated system. With apologies to Euripides, "The mills of process automation grind exceedingly slow, but grind exceedingly fine..." Anyone can do it—just be prepared for a long haul.

References

[1] Alain Andrieux, Karl Czajkowski, Asit Dan et al., "Web Services Agreement Specification (WS-Agreement)," September 2005: http://www.ogf.org/documents/GFD.107.pdf.

[2] J. R. Erenkrantz, "Release Management within Open Source Projects," in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, May 2003.

[3] J. Humble, C. Read, D. North, "The Deployment Production Line," in *Proceedings of the IEEE Agile Conference*, July 2006.

[4] D. V. Klein, "A Forensic Analysis of a Distributed Two-Stage Web-Based Spam Attack," in *Proceedings of the 20th Large Installation System Administration Conference*, December 2006.

[5] D. R. Wallace and R. U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, vol. 6, no. 3 (May 1989), pp. 10, 17.

[6] H. K. Wright, "Release Engineering Processes, Their Faults and Failures," (Section 7.2.2.2) PhD Thesis, University of Texas at Austin, 2012.

[7] H. K. Wright and D. E. Perry, "Release Engineering Practices and Pitfalls," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)* (IEEE, 2012), pp. 1281-1284.

[8] H. K. Wright, D. Jasper, M. Klimek, C. Carruth, Z. Wan, "Large-Scale Automated Refactoring Using ClangMR," in *Proceedings of the 29th International Conference on Software Maintenance (IEEE, 2013)*, pp. 548–551.

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

15th Workshop on Hot Topics in Operating Systems (HotOS XV)

Sponsored by USENIX, the Advanced Computing Systems Association **May 18–20, 2015, Kartause Ittingen, Switzerland**

Important Dates

- Paper submissions due (no extensions): **Friday, January 9, 2015, 11:59 p.m. GMT/UTC**
- Notification to authors: **Monday, March 16, 2015**
- Revised papers due: **Monday, April 20, 2015**
- Workshop: **Monday, May 18–Wednesday, May 20, 2015**

Workshop Organizers

Program Chair

George Candea, *École Polytechnique Fédérale de Lausanne (EPFL)*

Program Committee

Marcos K. Aguilera, *Microsoft Research*
Peter Druschel, *Max Planck Institute for Software Systems (MPI-SWS)*
Michael Freedman, *Princeton University*
Shyam Gollakota, *University of Washington*
Steve Hand, *Microsoft Research*
Butler Lampson, *Microsoft*
Shan Lu, *University of Chicago*
Petros Maniatis, *Intel Labs*
David Mazières, *Stanford University*
Bianca Schroeder, *University of Toronto*
Margo Seltzer, *Harvard University and Oracle Labs*
Emmett Witchel, *The University of Texas at Austin*
Nickolai Zeldovich, *Massachusetts Institute of Technology*

Steering Committee

Armando Fox, *University of California, Berkeley*
Casey Henderson, *USENIX*
Petros Maniatis, *Intel Labs*
Matt Welsh, *Google*

Overview

The 15th Workshop on Hot Topics in Operating Systems will bring together researchers and practitioners in computer systems, broadly construed. Continuing the HotOS tradition, participants will present and discuss new ideas about systems research and how technological advances and new applications are shaping our computational infrastructure.

Computing systems encompass both traditional platforms—smartphones, desktops, and datacenters—and new technologies like implantable embedded devices, geographically distributed stream processing systems, and autonomous flight control. In this context, a deluge of personal, corporate, sensitive, ephemeral, or historical information being produced, transmitted, processed, and stored poses interesting systems challenges. Systems are expected to guard this information, and do so better, faster, and using less energy.

We solicit position papers that propose new directions of systems research, advocate innovative approaches to long-standing systems problems, or report on deep insights gained from experience with real-world systems. HotOS areas of interest include operating systems, storage, networking, distributed systems, programming languages, security, dependability, and manageability. We are also interested in systems contributions influenced by other fields such as hardware design, machine learning, control theory, networking, economics, social organization, and biological or other nontraditional computing systems.

To ensure a vigorous workshop, attendance is by invitation only. Authors will be invited based on their submission's originality, topical relevance, technical merit, and likelihood of leading to insightful technical discussions that will influence future systems research. We will heavily favor submissions that are radical, forward-looking, and open-ended, as opposed to mature work on the verge of conference publication.

Paper Submission Instructions

Position papers must be received by the paper submission deadline mentioned above. **This is a hard deadline, and no extensions will be granted.**

Submissions must be no longer than 5 pages including figures and tables, plus as many pages as needed for references. Text should be formatted in two columns on 8.5 x 11-inch paper using 10-point Times Roman font on 12-point (single-spaced) leading, 1-inch margins, and a 0.25-inch gutter (separation between the columns). The title, author names, affiliations, and an abstract should appear on the first page. Pages should be numbered. Figures and tables should not require magnification for viewing; they may contain color, but should be legible when printed or displayed in black and white. Submissions not meeting these criteria will be rejected without review, and no deadline extensions will be granted for reformatting. Papers should be submitted as PDF files via the Web submission form, which will be available soon on the Call for Papers Web site.

Revised versions of all accepted papers will be available online to registered attendees before the workshop. After the workshop, accepted papers will be made available on the USENIX Web site, along with slides of the presentation, and a summary of the discussion at the workshop. Registered users of the USENIX Web site will have the ability to comment and/or ask questions, and the authors will be able to respond online, as well as expand their ideas and produce new versions of their papers. This online discussion will constitute an active history of how the idea or system evolves from the "hot idea" stage to mature work published in conferences and eventually released into the real world.

Simultaneous submission of the same work to multiple venues, submission of previously published work, or plagiarism constitutes dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may take action against authors who have committed them. See the USENIX Conference Submissions Policy, www.usenix.org/conferences/submissions-policy, for details.

Questions? Contact the program chair, hotos15chair@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Papers accompanied by nondisclosure agreement forms will not be considered. Accepted submissions will be treated as confidential prior to publication on the USENIX HotOS XV Web site; rejected submissions will be permanently treated as confidential.

www.usenix.org/hotos15/cfp

/var/log/manager
Parables of System Administration Management

ANDY SEELY

Andy Seely is the Chief Engineer and Division Manager for an IT enterprise services contract, and an adjunct instructor in the Information and Technology department at the University of Tampa. His wife Heather is his PXE Boot and his sons Marek and Ivo are always challenging his thin-provisioning strategy. andy@yankeetown.com

I've managed intelligent, educated, certified, opinionated, strong-willed, hard-working, and brilliant system administrators across many different companies, industries, states, and countries. Over the years I've found different approaches to connect with and motivate people. Sometimes people need to be directly told to do something, sometimes they need to be left alone to figure things out, and sometimes it's helpful to give them a story that they can use to cope with and overcome challenges. I call these stories my "Parables of Sysadmin."

Dealing with Difficult People: The Parable of Camping

We're colleagues, co-workers, friends. We work hard together and we like each other and we get along well. So if I just sucker-punched you right now, what would you do? You'd have two reactions: First, you'd fight back. Punch, pow, crash! Once the dust settled, you'd have a second reaction: You'd be offended. We're colleagues, co-workers, friends; why would you just attack me like that?

Now, imagine we're in the Ocala National Forest campground. It's a beautiful night. We're camping by the lake. It's a moonlit night, stars, campfire. S'mores. And, suddenly, out of the lake, a giant alligator jumps out and attacks you. You'd have two reactions: First, you'd fight back. Punch, pow, crash! Once the dust settled, and assuming you're still alive, you'd have a second reaction: You'd adjust your campsite to be less susceptible to attack. But you wouldn't be "offended." Why not? Because that gator is just an animal, it's not acting with malicious intent.

So now, when we're camping in the Workplace National Forest and the animals come out of the cubicle farm to attack you, why do you get so offended? Just adjust your campsite to be less susceptible to the attack and get back to camping.

Understanding Real Prioritization: The Parable of the Fireman

It's December and you're going to buy a new calendar for the coming year. You want an exciting calendar to hang in your cubicle, and you're looking at all the calendars with pictures of firemen. What do they look like? Sweaty, muscle-bound, wearing a helmet, holding an ax in one hand and a rescued kitten in another, with a five-alarm fire behind them. Heroic and sexy.

Did you ever realize that each fireman in this scene represents failure? Every fire that gets put out is a fire that wasn't prevented, which means that the checks and balances of the fire prevention system failed. We didn't conduct safety inspections. We didn't understand the limitations of our internal controls. We didn't have sufficient alerting and suppression systems in our environment. We had to call in some heroes to perform heroic acts to save the day, just like we do every day in our operational environment: We lionize, fetishize even, the heroic act of getting the SAN or the server back online. We recognize people in their annual performance reviews for heroic acts, effectively putting them on the sexy fireman calendar as a reward for saving our kittens.

Let's think about what's really effective. It's the old fire marshal, driving around in a boring old car, carrying a clipboard and walking into places to check sprinkler systems, verify extinguisher charge levels, and validate training plans. No one ever notices this person, yet if the job is done right, there's never a fire. No need for the heroics. No loss incurred. No downtime suffered. Sometimes heroes are necessary, but the truly impactful effort is prevention, monitoring, and fixing the little things. We never give an annual salary bonus to someone for just monitoring a system and reporting on syslog anomalies, since that's someone just "doing their job." Done right, that simple job prevents the need for all the heroics, and it should be rewarded, if not with salary bonuses then at least with a sexy calendar with pictures of syslog files and kittens who never needed rescuing.

Understanding True Root Causes: The 10-Layer OSI Model

We all know the Seven-Layer OSI model: physical, data link, network, transport, session, presentation, application. This is a great tool for understanding the layered, encapsulated nature of network communications. If the model is going to be truly useful for understanding complexity and solving problems in complex environments, it must be flexible and account for all the complexity. In the spirit of flexibility in changing times and with recognition of the debt we owe to Evi Nemeth's original expansion [1], I propose an updated expansion of the model from seven to 10 layers.

Most sysadmins unconsciously extend the seven-layer model to an eighth layer: the user. That element just on the other side of the application. The space between the chair and the keyboard. Sometimes referenced with the codes 1D-10-T. PEBCAK. L-User. Or, as I call it, "layer eight," the type of problem caused by lack of attention, lack of training, lack of discipline, or lack of patience on the part of the user.

Layer nine is the naturally occurring condition of groups of people, organizations, hierarchies, and how they interact with each other. When a technical problem is caused, propagated, or expanded due to organizational conflicts, disagreements between executives, or people refusing to do something because of something going on in another part of an organization, this is layer nine: the political layer.

Large organizations can introduce major system problems, like significant software purchased for non-existent or incompatible systems, due to misinformed and overly excited executives and senior leaders. Any time an environment is made too complex to manage due to a decision that was not vetted by technical staff, the problem is in layer 10 of the expanded model: the religious layer. Something was done because a true-believer took action without facts, and now we have to live with it.

Motivating Sysadmins to Be Effective and Relevant

These parables are stories that I tell every day. They're not the only tools in my manager's kit, but they're fun ways to help people. We hear people say the old cliché, "work smarter, not harder," but that's usually as effective as saying, "I want to cure world hunger" or "Let's hire consultants to tell us what to do." What does "working smarter" actually mean to a professional sysadmin? Does it mean to know more about the operating system? Does it mean to take another certification exam? Does it mean agreeing with the manager until the manager goes away and the sysadmin can get back to what he was doing?

Working smarter means maximizing the specific work activities that provide the most benefit to the company's primary business goals. It's a management challenge to understand those goals and keep top technical people on the tasks that make the whole team more effective at supporting those goals. Time spent complaining about others is wasteful (parable of camping), time spent with "all-hands-on-deck" heroic actions is inefficient (parable of the fireman), and time spent trying to solve people-politics-religion problems with more technology just makes things worse in the end (parable of the 10-layer OSI model). I'm the manager, and helping smart sysadmins be effective in big teams is my job.

References

[1] As cited by Avi Deitcher in "The Challenges of Open Source in the Enterprise": http://www.linuxjournal.com/article/10726.

Educating System Administrators

CHARLES BORDER AND KYRRE BEGNUM

Charles Border teaches networking and system administration classes in the BS and MS programs in networking and system administration in the Department of Information Science and Technology at the Rochester Institute of Technology in Rochester, NY. He is an active international educator offering classes at RIT Dubai and in partner programs in the Dominican Republic. His areas of research include system administration education, international higher education, enhancing the innovation and creativity in professional master's degrees, cloud computing, and teaching in the cloud. Mr. Border earned a PhD in higher education administration in 2000 and an MBA in 1993, both from the University at Buffalo. He lives in South Wales, NY, with his wife Kathleen and their three children.
cbbics@rit.edu

Kyrre Begnum, PhD is an associate professor at Oslo and Akershus University College of Applied Sciences in the Department of Computer Science, and is an adjunct professor at Gjøvik University College in Oslo, Norway, where he teaches primarily systems-related courses. Dr. Begnum has published extensively in the field of cloud computing and virtualization.
Kyrre.Begnum (at) hioa.no

If you are a long-time attendee of LISA conferences, you will be very familiar with the educators groups that have met either as part of LISA or just before the LISA conference. The Summit for Educators in System Administration (SESA) had its first official meeting in 2013 under the guidance of Kyrre Begnum from Oslo University College and Caroline Rowland from the USENIX Board and NIST. The meeting was a big success with more than 30 academics and others interested in the education of the next generation of system administrators in attendance. Later in 2014, the USENIX Board decided to embrace SESA as a new group under the USENIX banner.

During meetings around SESA, we decided to petition the USENIX Board to form a journal, separate from but affiliated with SESA, called the *Journal of Education in System Administration (JESA)*. Our vision for SESA and *JESA* is to give academics and others interested in the education of the next generation of system administrators a place to discuss their efforts and to share best practices.

The reason it makes sense to do this under the USENIX banner, rather than the other academic computing organizations such as the ACM or the IEEE, relates to our vision of system administration and operations as a very applied field within computing that has not received its fair share of respect within the more theoretical computing organizations. As academics, we feel more at home in the USENIX community and feel that it is a better home for our vision of what system administrators do in the world of work. The professionalism of the USENIX community fits better with our vision of what we want to instill in our students, and we look forward to working with the community to help us advance our shared profession.

The Future of Computing

Computing, as an academic discipline, has just hit its fiftieth birthday and is undergoing a period of introspection something like what many of us go through around mid-life. Since its inception, the idea that computer science was really a "science" has been an item for debate [1]. One of the main concepts behind the idea that computer science was not a "science" was the notion that, unlike the other three branches of science (physical, life, and social sciences), computer science dealt with an "artificial" environment. This is becoming less persuasive as we start to gain a better understanding of the similarities between the computation that we do with computers and the computation involved with some of the most basic life processes such as evolution, natural selection, chemistry, gene regulatory systems, and neuronal networks [2].

There is another way to look at computing as "the union of three disparate perspectives: the mathematical perspective, the scientific perspective, and the engineering perspective." [2] From this perspective, computing as a discipline derives its use of various formalisms from mathematics, its drive for continuous improvements from engineering, and its desire to make empirical predictions from the small and simple to the very large and complex from all the sciences.

As computing has evolved from its emphasis on engineering and the initial development of systems in the early days to our current situation of very large and complex systems, the question still remains: What is the future of computing? Frustrated with all the other names (e.g., the science of computation, or the science of computer science), Richard Snodgrass has proposed an entirely new term to describe the future of computing, ergalics. "The goal of Ergalics is to express and test scientific theories of computational tools and of computation itself and thus to uncover general theories and laws that govern the behavior of these tools in various contexts. The challenge now before the CS discipline is to broaden its reliance upon the mathematical and engineering perspectives and to embrace the scientific perspective." [3]

This matters for us as we begin to think of working to define curricula around system administration, because we need to have a better understanding of what the goals and outcomes should be for our programs. One of the biggest changes that has happened to higher education in the last decade or two has been a growing demand from our stakeholders for an increase in accountability for the resources that we consume. This has been instantiated through the rise of the assessment movement.

Measuring the Performance

Assessment requires that each program approach the measurement of the success of the program from a three-step process. Each institution defines a mission statement that defines who the institution serves and the relationship of the institution to the world around it. Based on this mission statement, each program defines a set of broad program educational outcomes that define the characteristics of the graduates of the program three to five years after graduation. The idea behind having this time lag is that we do not want to educate our students to just be able to get that first job; we feel that educating students properly prepares them to be lifelong learners. Measurement of a program's educational outcomes is a problematic thing. A lot of important things can happen to a person between the ages of 22 and 27, and just contacting our graduates can be difficult. Our attempts to measure program educational outcomes are imperfect at best and rely on the use of surveys and our other contacts with our graduates. Lastly, each program defines its student outcomes, which are the things that students should be able to do when they graduate. Through our assessment process we measure the results of our programs, and we feed back the results of our assessment into program changes to make the program better over time.

Student outcomes are generally measured in individual courses that all students are required to take. The actual means of measurement depends on the type of outcome. If an outcome relates to a student's ability to communicate effectively, we might measure this by grading student writing assignments against a rubric that breaks down the grade for the assignment into several categories, with a score assigned to each category. If the outcome relates to the ability of a student to do something (e.g., configure a BIND DNS server), we might have a standard lab assignment that all our students need to complete that is graded against a rubric.

In the old days prior to assessment, we asked our constituencies to trust us about how good a job we were doing. Now we have a process in place to measure the contribution each program makes toward the institution's ability to live up to its mission statement and find ways to make the program better over time. From the perspective of a teaching faculty member, a couple of the most important points about this process are that we develop our own set of program educational outcomes, student outcomes, and a process by which we use metrics to improve the program. This may sound very bureaucratic, but in the end it is very much a faculty developed and led process through which we can improve our programs.

From a day-to-day perspective, assessment, in essence, asks us to define a set of program goals, break those goals down into outcomes, align those outcomes with specific courses, and find ways to measure the ability of each course to contribute to the overall success of the program. If a course does not enhance the ability of students to satisfy the program outcomes, it should be removed from the curriculum. If students successfully accomplish the goals we have set out for them, we acknowledge that and move on, and if they do not, we examine what we do and try to find ways to do a better job in the individual courses.

To make assessment work effectively, we need to have the right program outcomes, and this is the area where a curriculum that concentrates on the applied skills of a professional system administrator should be very different from the more theoretical skills of a computer science program. If system administration and computer science program outcomes were the same, there would be no reason to have a separate system administration program.

Program Outcomes

There are two dimensions to the design of new program outcomes. The first dimension concerns the window of time that students are in our programs. What can we teach students when they are fresh out of high school that will be meaningful to a career that does not begin for four years? A relevant way to think about this is to reflect on the systems practices that we were pursuing four years ago. What has changed since then and what has stayed the same? A related issue is the amount of time that we have with the students. If we are to add things to an already busy curriculum, what can we take out? If we add configuration management because we decide that every systems person should have a working knowledge of how to maintain the consis-

tency of the configuration of many machines, can we remove a semester of Java? Tradeoffs must be made, and the place that we make them is in the program outcomes.

A second dimension of the development of program outcomes relates to what we determine are the main things that you want an entry-level system administrator to be able to do. Is programming in a specific language such as Perl or Python the most important thing, or is it more important to have a general understanding of, for example, troubleshooting skills or service deployment architectures? When we work to develop the educational outcomes associated with programs, we have to be very careful, because you might get what you ask for. If we lean too heavily on the needs of the moment will we have a very narrowly skilled employee who is unable to adapt to changes in the demands placed on him? Even though many organizations are relying heavily on Puppet for configuration management and EMC for large scale storage architectures, should we design our curriculum around these specific technologies or should we concentrate on developing a more generalist curriculum that stresses things like Bash scripting, Web services, and storage area networks and networking?

There is a difference between developing lab exercises that require students to use specific technologies (deploy this Web service on this Web server, running on this operating system) and building context around basic technologies by discussing the general concepts involved with the technologies. The concepts last, but the specific technologies change very rapidly. The same distinction applies when we develop the outcomes associated with our programs. If our outcomes are too specific and technology-focused, we run the risk of having to change our outcomes with each iteration in technology, and of having students whose education loses its relevance before they even graduate.

Student Recruitment

We have heard from many of you that it is very difficult to recruit the right new employees for your businesses, and we in higher education have heard you and we want to help. But we also have a problem. The kind of very bright, hardworking, and creative students that you want to recruit to run your systems have many options when they choose a major and very little understanding of what the different majors and the careers they lead to actually consist of once they graduate. Many students show up at college not knowing about different careers, but knowing that they want to major in something related to computing. While this is fine, it presents a problem for those of us seeking to recruit them into a specialized field such as system administration that they may never have even heard of. Although this generation of students is just as rebellious as we were (which is good), parents play a larger role in the student's decision-making than we usually give them credit for. But the same problem remains: The parents may not know what a system administrator is either. The current growth in computer science enrollments may be a response to the uncertainty that many people feel about jobs (let alone careers) these days, with students opting to major in the more well known, generalist computer science degree rather than a specific career path that they don't understand and that might (so they fear) be outsourced, leaving them in debt and unemployed.

This is particularly a problem as we try to recruit a more diverse student body. Just as industry is being asked more pointed questions about the diversity of their employees, we are also receiving the same types of questions. It is very important for us to expand the pool of students interested in systems, educate all the students, and create an environment where all students can succeed.

To successfully recruit the kind of students that you will want to recruit as employees, we need to create an interesting curriculum that allows students to gain an understanding of the field of system administration and, at the same time, excites their interest, creativity, and problem-solving skills. Our goal in developing a system administration curriculum should be to develop our students into employees who feel empowered to be creative and find their work engaging, interesting, and worth concentrating their efforts on.

The Future of System Administration and Operations Education

With only a very few exceptions, computing programs in higher education are dominated by computer science programs based on a more theoretical understanding of what computing is all about. While this might be sufficient for many careers in computing we don't feel that it is the right approach for all careers in computing and all organizations. The goal of SESA and *JESA* is to create a venue where people interested in a different side of computing can exchange ideas and information relevant to the development of new curricula in system administration. These new curricula may come in many different flavors, with some being more programming focused and others more focused on hardware/service deployment issues. And they may rely on different phrases to describe their curriculum (operations seems to be a bigger concept than system administration, but if students don't know what system administration means, they certainly don't know what operations means) and/or rely more on business concepts (operations management is an interesting topic to many people) than strictly on computing, but we want to provide a place for all of them.

For the academics reading this article, we want to provide a place to discuss your plans for the future and goals for your curriculum. For the industry people reading this, we want to encourage you to become involved both in our new group and

especially with your local colleges and universities. They need your input into the curriculum design process, and they need your talents as an instructor. If you have never taught a college course, you may find it to be a very interesting change of pace for you that puts you in contact with some very bright and hard-working students and gives the students an opportunity to benefit from your experience. At SESA we also need your thoughts and experience as we try to distill from our rapidly changing industry those things that will last and that can form the basis for an interesting and challenging curriculum.

References

[1] Peter J. Denning, "The Science in Computer Science," *Communications of the ACM*, vol. 56, no. 5 (May 2013), pp. 35–38.

[2] Richard Snodgrass and Peter Denning, "The Science of Computer Science: Closing Statement," Ubiquity Symposium: The Science of Computer Science (June 2014), DOI=10.1145/2633608.

[3] Richard Snodgrass, *Ergalics: A Natural Science of Computation* (University of Arizona, 2010).

DIVERSITY

CRA-W Grad Cohort

Guiding Female Graduate Students Towards Success

DILMA DA SILVA

Dilma Da Silva is professor and department head of the Department of Computer Science and Engineering at Texas A&M University. She is also director for the Computer Science and Engineering Division of the Texas A&M Engineering Experiment Station (TEES). She previously worked at Qualcomm Research in California, IBM Research in New York, and University of Sao Paulo in Brazil. She received her PhD from Georgia Tech in 1997. She has published more than 80 technical papers and filed 14 patents. Dilma is an ACM Distinguished Scientist, an ACM Distinguished Speaker, a member of the board of CRA-W and CDC, co-founder of Latinas in Computing, and treasurer for ACM SIGOPS.
dilmamds@gmail.com

Presented by

I enjoy attending conferences such as OSDI, USENIX ATC, VEE, and SOSP because I get exposed to exciting technical ideas as I discuss great research results with colleagues whom I cherish. But, by far, my favorite event is the annual CRA-W Grad Cohort Workshop.

Grad Cohort is a two-day workshop where 300+ female graduate students interact with 25+ senior women in computing research. Grad Cohort accepts students in their first, second, or third year of graduate school in computer science or engineering. Senior women come from academia, industry research, and national labs, covering a diverse set of computing disciplines; in 2014, three of them were active members of the USENIX community.

The program includes a mix of formal presentations and informal discussions and social events. Some presentations cover mentoring advice such as how to become more effective in professional networking, improve communication skills, and balance graduate school and personal life. These are traditional mentoring topics, but at Grad Cohort they come alive as presenters include more personal information and insights about their experiences in handling the specific opportunities and challenges they faced in their research careers. Every year that I attended such sessions, I planned to half-listen to them as I tackled work on my laptop, but I found myself mesmerized by the relevant and fresh perspectives being presented.

The program also includes information on graduate school survival skills, organized in parallel tracks targeting first year, second year, and third year students. Sessions include: master's versus PhD programs; strategies for finding an advisor, research topic, and financial support; thesis proposal preparation; dissemination of research results; internships; and job search and interview tips for academic and industry jobs. Panels covering topics such as building self-confidence and a professional persona are very popular with attendees. The program ends with direct feedback sessions, such as a resume writing clinic and individual advising. You can find the slides from presentations for the 11 editions of the event at cra-w.org/gradcohort.

Beyond the strength of the program, Grad Cohort is special to me because of the unique atmosphere emanating from a group of 300+ women discussing their experiences as graduate students in computer science. Attendees have commented that the welcoming and supportive environment in the workshop leaves them feeling more empowered to handle challenges back in school and eager to deploy what they learned in the workshop to make the best out of the opportunities they have. And, yes, after dinner on Friday evening, we dance our hearts out.

CRA-W carries out extensive data collection and analysis to demonstrate that Grad Cohort is effective in improving the success and retention of women in computing research. Information on how the 2014 edition of the workshop impacted attendees is available on the evaluation report [1]. Another report [2] from the CRA Center for Evaluating the Research Pipeline contrasts data from Grad Cohort participants and non-participants.

According to the Taulbee Survey [3], in 2014 only 292 out of the 1475 (19.8%) PhDs in computer science or computer engineering were awarded to women. Grad Cohort can have

an even bigger impact on the computing research pipeline if expanded to meet actual demand. Generous funding from industry, nonprofit associations, university computer science departments, and individual donors covers all participant travel and workshop expenses. In 2014, the funding allowed the workshop to accept only 304 of the 503 applicants. The participant selection process maximizes the number of schools represented in the group. With more funding, we can get more women prepared to excel in computer science research. If your institution is in a position to sponsor a few students (or many!), please contact me so that I can provide you with detailed information about the Grad Cohort initiative.

As I write this article, the 2015 dates for Grad Cohort have not been defined yet, but by the time you read this we may be approaching the application deadline. Usually, the workshop happens in April and student applications are due in late November. If you are a female graduate student, I encourage you to consider applying to the workshop. If you work with female graduate students who may not be aware of this program, please advise them to check out the CRA-W Web site, cra-w.org.

References

[1] J. L. Cundiff, J. G. Stout, and H. Wright, CRA-W Grad Cohort 2014: Pretest/Posttest Evaluation Report, May 2014 (Computing Research Association: Washington, DC): http://cra.org/cerp/evaluation-reports.

[2] J. G. Stout and J. L. Cundiff, CRA-W Grad Cohort: Comparative Evaluation Report of 2011-2012 Participants, February 2014 (Computing Research Association: Washington, DC): http://cra.org/cerp/evaluation-reports.

[3] S. Zweben and B. Bizot, 2013 Taulbee Survey, *Computing Research News,* vol. 26, no. 5, May 2014: http://www.cra.org/uploads/documents/resources/crndocs/2013-Taulbee-Survey.pdf.

Practical Perl Tools
Get Your Health Checked

DAVID N. BLANK-EDELMAN

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

Every once in a while I like to inject a little reality into this column, more specifically my reality. This month, instead of writing about some abstract technology or documenting a done deal, I thought it might be fun to work together on a small project that is actually in flight as I write this. This will give you a chance to listen in on my current thoughts (such as they are), and together we can examine some rough code that implements these ideas.

The project I have in mind revolves around a new LDAP cluster that we are currently installing. LDAP stands for Lightweight Directory Access Protocol and is basically the de facto standard for talking to a directory server. Directory servers are used to provide the backbone for most authentication/authorization setups. For example, if you log into a machine that uses some sort of central authentication scheme, chances are the client is doing an LDAP operation at some point as part of the process. This is truly cross-platform (e.g., if you log into a Windows network, you'll be talking LDAP at some point to your ActiveDirectory server(s)).

If you've never dealt with LDAP before, never fear, we won't be assuming much knowledge of it nor will we go very deep. There's a lot that can be written about it (and, indeed, I have a whole chapter and an appendix on it in my book). For the purpose of this column, I'll try to provide enough context so the code makes sense. And, actually, if you take a step back and squint at this column from a little distance away, you'll find that LDAP is just a small detail in the larger picture of health checks, the true subject for today.

So what's a health check and why do I (and maybe you) care? In my case, the LDAP cluster we have set up consists of four LDAP servers that are "behind" a load balancer (actually a pair of them, but that's another story for another column). The load balancer's job is to transparently take in LDAP requests and parcel them out to the actual servers in a balanced way so the load is spread evenly amongst the operational machines. The key word for this column has just been spoken: "operational." One other key purpose for using a load balancer is to make sure that if a machine in a cluster becomes dysfunctional, the clients of that cluster don't notice because the load balancer has cleverly removed that machine from the list of servers it is sending traffic to. If and when that machine returns to service, the load balancer may decide to bring it back into the fold.

Here comes the rub: A load balancer has to know which machines it stands in front of are working and which are not. The way this is typically done is to have the load balancer continuously perform a "health check" on each of the cluster members. Health checks can be simpleminded and naive or fiendishly clever. Right now our current health checks are barely the former, and that's the problem. At the moment, the load balancing software (keepalived, if you are curious) is just checking to see if it can connect to the LDAP port on each of the servers. That's not good enough—we can do much better. Let's rough out a few ways we can improve the situation.

Do What I Do

Being able to connect to the server is great and all that, but LDAP clients connect for a reason. They expect to be able to talk to the server and perform LDAP operations. When writing health checks for almost any service, you'll be off to a great start if your checks mimic even a minimal set of operations a client would be expected to perform. In the case of LDAP this set includes an LDAP bind operation (think of it as "logging into the server"), an LDAP search operation, and an LDAP unbind (which the RFC describes as "the 'quit' operation...the client, upon transmission of the UnbindRequest, and the server, upon receipt of the UnbindRequest, are to gracefully terminate the LDAP session."). Let's look at a little Perl code that does all three things:

```
use strict;

use Net::LDAP;

my ( $server, $binddn, $bindpw, $lookup ) = @ARGV;

my $ldap = Net::LDAP->new($server) or die "Can't connect: $!";
print "connected.\n";

my $res = $ldap->bind( $binddn, password => $bindpw );
$res->code && die "Can't bind: " . $res->error;
print "bound to server.\n";

$res = $ldap->search(
   base   => 'ou=people,dc=example,dc=edu',
   scope  => 'one',
   filter => $lookup,
);
$res->code && die "Search failed: " . "$res->error";
print "entries found " . $res->count . "\n";

$res = $ldap->unbind;
$res->code && die "Unbind failed: " . "$res->error";
print "unbound to server.\n";
```

To quickly walk you through the code, we create an Net::LDAP object that connects us to the server. We then bind() (login) to it. At this point, we execute a search that starts at a particular place in the tree (base), looks at only the part of the tree one level down under that place (one), and filters the result. Lastly, we unbind() to the server. Here's what happens when we run the code:

```
$ ldap.pl localhost 'managerdn' 'managerpw' '(sn=smith)'
connected.
bound to server.
entries found 11
unbound to server.
```

Here you can see we're testing just a few LDAP operations. There are definitely others (compare and modify come to mind) that we should add to this test. More on that last one later. I should also note that this is very simple code that doesn't take into account slow or hung servers (ideally, we should build timeouts into the script to cause it to abort if operations take too long).

If we wanted to be a little cooler, we could go to the logs of a running version of the service and pull a representative slice of the live workload and use it to form the basis of an even better test. Note I said "basis," because we probably don't want to replay it verbatim to our servers, especially if it contains write operations. It would be more than a little embarrassing to have our health checks repeatedly overwrite live data in our directory, though it wouldn't surprise me if this has happened before.

Ah, But How Fast Did I Do It?

Once we know how to pretend to be a client of the server and perform the same operations it might perform, a logical step forward is to model another thing we can expect from our clients: impatience.

In the last section we concerned ourselves with whether our service would answer the phone, reply to our request, and then hang up properly. LDAP clients care about all of these things, but they also care about how long those things take. In many cases a server that replies too slowly might as well be down ("you are dead to me"). Our health check needs to catch this case as well. The first step towards this is timing how long each operation takes. We can do that with code that looks a bit like this:

```
use Net::LDAP;
use Time::HiRes qw(time);

my $start = time();
my $ldap = Net::LDAP->new($server) or die "Can't connect: $!";
my $end   = time();
print "connected: " . ( $end - $start ) . "\n";

$start = time();
my $res = $ldap->bind( $binddn, password => $bindpw );
$end = time();
$res->code && die "Can't bind: " . $res->error;
print "bound to server: " . ( $end - $start ) . "\n";
```

In the above sample we are using the module Time::HiRes because the Perl's native time resolution is seconds (i.e., time() returns the number of seconds since the epoch). In this breakneck world we live in, we expect response times in less than a second. Time::HiRes gives us the extra resolution we need. Take a look at the difference between what time() returns without and with Time::HiRes loaded:

```
$ perl -e 'print time(),"\n"'
1406570529
$ perl -e 'use Time::HiRes qw(time);print time(),"\n"'
1406570567.63434
```

Once we know how fast an operation is, we can then enforce a standard in our health check. Something easy like:

```
($bind_time < 1.0) ? 'up' : 'down';
```

Now, on my unloaded server, that standard is far too generous. Here's what the new code that prints how long each operation takes shows when I run it against an unloaded server:

```
connected: 0.00156688690185547
bound to server: 0.00202512741088867
entries found 11: 0.0248799324035645
unbound to server: 0.000317096710205078
```

Oversimplification Alert! Taking a server out of service based on just a single slow operation sounds both draconian and inefficient. It's more likely you would be better served if you did some math to determine whether the server is consistently reporting back times within a reasonable range. This requires some sort of persistent state be kept around between health checks, a topic we're not going to touch on in this column.

Yup, Still Me in the Mirror

The previous mention of the LDAP modify operation (and write operations in general) brings up another useful aspect to consider when writing health checks. One important way to test a service that includes write operations is through "round trip" tests. Sure, we could write code that performs an LDAP modify of the data and then believe the server if it reports back a successful modification, but it would be far better if we actually did another read to confirm it worked. As the Russian proverb says, "trust, but verify." The idea of round-trip verification comes in handy in many places. For example, when health-checking a mail system, it would be great to have the health check send mail to the system and then attempt to retrieve it a few moments later.

In our case, we can do something like this:

```
my $testdn = 'uid=canaryuser,ou=people,dc=example,dc=edu';
# ...connect and bind as usual, then
$start = time();
my $res = $ldap->modify( $testdn,
             replace => { 'displayName' => $start } );
$res->code && die "Can't modify: " . $res->error;

$res = $ldap->search(
   base   => $testdn,
   scope  => 'base',
   filter => "(displayName=$start)",
);
$end = time();

$res->code && die "Search failed: " . $res->error;
print "entries found " . $res->count . ": " .
                          ( $end - $start ) . "\n";
```

In this code we modify the value of the displayName attribute in a test user's LDAP entry—we set it to be a timestamp. The next code section attempts to search for that user with a filter that should only return back an entry if the displayName is set to that timestamp correctly. If we return an entry, success. If not, sad trombone.

By the way, a more efficient way to check whether the displayName value has been set to the desired timestamp would be to use a compare operation instead of a search:

```
use Net::LDAP::Constant qw(LDAP_COMPARE_TRUE
                           LDAP_COMPARE_FALSE);
$res = $ldap->compare( $testdn,
                       attr => 'displayName',
                       value => $start);
print "compare succeeded"
      if ($res->code == LDAP_COMPARE_TRUE);
```

In this section we've seen one very simple round-trip test. I'll mention a slightly more sophisticated one related to this test at the end of this column.

Tell Me How You Feel

A piece of well-instrumented server software has a way of reporting its internal sense of health. In the case of the LDAP server we are using (OpenLDAP), it provides a special LDAP suffix we can query to return all kinds of internal counters and statistics. Here's some code that dumps one interesting set:

```
use Net::LDAP;

my ( $server, $binddn, $bindpw ) = @ARGV;

my $monitordn = 'cn=Operations,cn=Monitor';

my $ldap = Net::LDAP->new($server) or die "Can't connect: $!";

my $res = $ldap->bind( $binddn, password => $bindpw );

$res->code && die "Can't bind: " . $res->error;

$res = $ldap->search(
   base   => $monitordn,
   scope  => 'one',
   filter => '(objectClass=*)',
   attrs  => [ 'monitorOpInitiated', 'monitorOpCompleted' ],
);
$res->code && die "Search failed: " . $res->error;

my @operations = $res->entries;
foreach my $operation (@operations) {
   my $dn = $operation->dn;
   my ($opname) = $dn =~ /cn=(\w+),/;
   print "$opname: "
      . $operation->get_value('monitorOpInitiated')
      . " initiated, "
```

```
        . $operation->get_value('monitorOpCompleted')
        . " completed\n";
}

$res = $ldap->unbind;
$res->code && die "Unbind failed: " . $res->error;
```

When run on a pretty fresh server, we get results that look like this:

```
Bind: 34 initiated, 34 completed
Unbind: 25 initiated, 25 completed
Search: 29 initiated, 28 completed
Compare: 3 initiated, 3 completed
Modify: 3 initiated, 3 completed
Modrdn: 0 initiated, 0 completed
Add: 0 initiated, 0 completed
Delete: 0 initiated, 0 completed
Abandon: 0 initiated, 0 completed
Extended: 0 initiated, 0 completed
```

Once you can figure out just which statistics are important to you, it is easy to write a health check that uses them as an indicator of health (with or without the kind of math we discussed in the timing section above). Perhaps you consider a server healthy if it has a small ratio of Modify to Search operations; too many writes could indicate a problem. A query like the one above can determine whether this condition is being met. One last note before we move on: If your server isn't well-instrumented, get a better server (if you can).

Happy Family

For the last section, let's pull the camera back a little further. In a multiple server setup where the servers keep themselves in sync with each other like we have, replication status (i.e., is the data in all of the servers in sync) can be pretty important. So important, we should have a health check for that. OpenLDAP provides a fairly simple mechanism for this. Each time a replication takes place, the server sets an operational attribute called contextCSN with data about the most recent entry that this server contains (it can also keep track of the latest entries it has seen from its replication partners). We can compare contextCSN in two servers to determine whether they are in sync. The structure of this attribute (as per the docs) is:

```
GT '#' COUNT '#' SID '#' MOD

GT: Generalized Time with microseconds resolution,
without timezone/daylight saving:

YYYYmmddHHMMSS.uuuuuuZ

YYYY: 4-digit year (0001-9999)
mm: 2-digit month (01-12)
dd: 2-digit day (01-31)
HH: 2-digit hours (00-23)
MM: 2-digit minutes (00-59)
SS: 2-digit seconds (00-59; 00-60 for leap?)
.: literal dot ('.')
uuuuuu: 6-digit microseconds (000000-999999)
Z: literal capital zee ('Z')

COUNT: 6-hex change counter (000000-ffffff); used to
distinguish multiple changes occurring within the same time
quantum.

SID: 3-hex Server ID (000-fff)

MOD: 6-hex (000000-ffffff); used for ordering the modifications
within an LDAP Modify operation (right now, in OpenLDAP it's
always 000000)
```

Here's a sample set of them from one of my servers:

```
$ ldapsearch -x -LLL -H ldap://localhost
    -s base -b 'dc=example,dc=edu' 'contextCSN'
dn: dc=example,dc=edu
contextCSN: 20140729211439.000593Z#000000#001#000000
contextCSN: 20140514223302.072724Z#000000#002#000000
contextCSN: 20140514224132.047675Z#000000#003#000000
contextCSN: 20140514231128.299773Z#000000#005#000000
```

We could parse this in Perl either with a regular expression like the following (which I found in the Nagios plugin at ltb-project.org):

```
m/(\d{14})\.?(\d{6})?Z#(\w{6})#(\w{2,3})#(\w{6})/g;
```

or by using unpack(), as in:

```
unpack("A14 A1 A6 A1 A1 A6 A1 A3 A1 A6");
```

Parsing these values gives us the time of the latest entry on this server and the latest entry this server has seen from the other servers. If we then go query the other servers, we can start to compare the contextCSN values and get a sense of how in sync they are. On a busy cluster with lots of write activity, you would expect the numbers to drift apart some.

For health check purposes, the question then becomes: How big a difference between servers is acceptable to you before you declare a server "not in sync"? Calculating the difference between times is just a matter of subtraction (perhaps wrapped in an abs() to get the absolute number). As we did before with the timing question, we can then compare it against an acceptable range (or at least an acceptable upper bound).

The key thing here is we are now determining health of a server by its relationship to other servers, a pretty big leap in our thinking. That leap might lead you to revisit the round trip idea from an earlier section. It's not hard to envision a round-trip

test where you attempt to see how quickly a write made to one server appears on another (or several, or perhaps all?) replicated server(s). Hopefully, this idea shows you there are a ton of directions we could continue to explore around the simple idea of a health check.

Take care and I'll see you next time.

Endnote: Lest you think this isn't a true reflection of my reality, while working on the section about inter-server synchronization, I realized much to my chagrin that the servers in the LDAP cluster I was building were not properly keeping themselves in sync (they were constantly doing a full synchronization, which is not the way they are supposed to work). Two days of blood, sweat, and tears later, I now have a much better understanding of the role contextCSN plays in replication and how it is supposed to work. (Oh, and the cluster is fixed, too.) Thanks *;login:* column!

A Path Less Traveled

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply.html). Beazley is based in Chicago, where he also teaches a variety of Python courses dave@dabeaz.com

If you're like me, you've probably written a Python script or two that had to manipulate pathnames. For that, you've probably used the much beloved `os.path` module—and perhaps the `glob` module. And let's not forget some of their friends such as `fnmatch`, `shutil`, `subprocess`, and various bits of functionality in `os`. Aw, let's face it, who are we kidding here? Pathname handling in Python is an inexplicable mess, has always been a mess, and will always continue to be a mess. Or will it?

In this installment, I take a look at the new `pathlib` standard library module added to Python 3.4 [1]. More than 10 years in the making, it aims to change the whole way that you manipulate files and pathnames—hopefully, for the better.

Classic Pathname Handling

In programs that need to manipulate files and pathnames, certain tasks seem to arise over and over again. For example, splitting pathname components apart, joining paths together, dealing with file extensions, and more. To further complicate matters, POSIX and Windows systems don't agree on basic features such as the path separator (/ vs. \) or case sensitivity. So if you try to write all of the code yourself, it quickly becomes a mess. For these tasks, the `os.path` module is usually the recommended solution. It mainly provides common operations that you might apply to strings containing file names and does so in a platform-independent manner. For example:

```
>>> filename = '/Users/beazley/Pictures/img123.jpg'
>>> import os.path

>>> # Get the base directory name
>>> os.path.dirname(filename)
'/Users/beazley/Pictures'

>>> # Get the base filename
>>> os.path.basename(filename)
'img123.jpg'

>>> # Split a filename into directory and filename components
>>> os.path.split(filename)
('/Users/beazley/Pictures', 'img123.jpg')

>>> # Get the filename and extension
>>> os.path.splitext(filename)
('/Users/beazley/Pictures/img123', '.jpg')
>>>

>>> # Get just the extension
>>> os.path.splitext(filename)[1]
'.jpg'
>>>
```

In practice, using these functions gets a bit a more messy. For example, suppose you want to rewrite a file name and change its extension. To do that, you might write code like this:

```
>>> filename
'/Users/beazley/Pictures/img123.jpg'
>>> dirname, basename = os.path.split(filename)
>>> base, ext = os.path.splitext(basename)
>>> newfilename = os.path.join(dirname, 'thumbnails',
base+'.png')
>>> newfilename
'/Users/beazley/Pictures/thumbnails/img123.png'
>>>
```

Actually, all of that code is probably embedded inside some sort of larger task. For example, processing all of the images in an entire directory:

```
import os.path
import glob

def make_thumbnails(dirname, pat):
  filenames = glob.glob(os.path.join(dirname, pat))
  for filename in filenames:
     dirname, basename = os.path.split(filename)
     base, ext = os.path.splitext(basename)
     newfilename = os.path.join(dirname, 'thumbnails',
                   base+'.png')
     print('Making thumbnail %s -> %s' % (filename, newfilename))
     out = subprocess.check_output(['convert', '-resize',
                   '100x100', filename, newfilename])

# Example
make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

Here's a more complicated example that recursively walks an entire directory structure, making directories, and launching subprocesses:

```
import os
import os.path
import subprocess
from fntmatch import fnmatch

def make_thumbnails(topdir, pat):
  for path, dirs, files in os.walk(topdir):
    filenames = [filename for filename in files
                      fnmatch(filename, pat)]
    if not filenames:
      continue

    newdirname = os.path.join(path, 'thumbnails')
    if not os.path.exists(newdirname):
      os.makedir(newdirname)

    for filename in filenames:
      base, _ = os.path.splitext(filename)
      newfilename = os.path.join(newdirname, base+'.png')
      origfilename = os.path.join(path, filename)
      print('Making thumbnail %s -> %s' % (origfilename,
      newfilename))
        out = subprocess.check_output(['convert', '-resize',
                  '100x100', origfilename, newfilename])

if __name__ == '__main__':
  make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

Again, if you've written any kind of Python code that manipulates files, you're probably already pretty well familiar with this sort of code (for better or worse).

Past Efforts to Improve Path Handling

Complaints about Python's pathname handling in `os.path` are varied but tend to focus on a couple of common themes. First, there is the fact that the interface doesn't really match other parts of Python, which are usually more object-oriented. Second, a lot of the useful functionality concerning files tends to be spread out over many different standard library modules. As such, file-name handling code becomes more messy than it probably needs to be.

Efforts to improve Python's path handling apparently go back nearly 15 years. To be honest, this is not an aspect of Python that has garnered much of my own attention, but the rejected PEP 355 cites discussions about the matter going as far back as 2001 [2]. The third-party `path` module, created by Jason Orendorff, may be the best-known attempt to clean up some of the mess [3]. With `path`, you create path objects and manipulate them in a more object-oriented manner:

```
>>> from path import path
>>> filename = path('/Users/beazley/Pictures/img123.
jpg')

>>> # Get the base directory name
>>> filename.parent
path(u'/Users/beazley/Pictures')

>>> # Get the base filename
>>> filename.name
path(u'img123.jpg')

>>> # Get the base filename without extension
>>> filename.namebase
u'img123'

>>> # Get the file extension
>>> filename.ext u'.jpg'
>>>
```

path objects can be joined together using the / operator in a way that mimics its use on the file system itself. For example:

```
>>> filename.parent / 'thumbnails' / (filename.
namebase + '.png')
path(u'/Users/beazley/Pictures/thumbnails/img123.png')
>>>
```

path objects include a large variety of other methods related to manipulating files, including globbing, reading, writing, and more. For example:

```
>>> # Read the file as bytes
>>> data = filename.bytes()
>>>

>>> # Remove the file
>>> filename.remove()
path(u'/Users/beazley/Pictures/img123.jpg')
>>>

>>> # Check for existence
>>> filename.exists()
False
>>>

>>> # Walk a directory tree and produce .JPG files
>>> for p in path('/Users/beazley/Pictures').walk('*.
JPG'):
...     print(p)
...
/Users/beazley/Pictures/Foo/IMG_0001.JPG
/Users/beazley/Pictures/Foo/IMG_0002.JPG
/Users/beazley/Pictures/Foo/IMG_0003.JPG
...
/Users/beazley/Pictures/Bar/IMG_1024.JPG
/Users/beazley/Pictures/Bar/IMG_1025.JPG
```

Here is a revised version of the image thumbnail code that uses path.

```
from path import path
import subprocess

def make_thumbnails(topdir, pat):
  topdir = path(topdir)
  for filename in topdir.walk(pattern=pat):
     newdirname = filename.parent / 'thumbnails'
     if not newdirname.exists():
        newdirname.mkdir()
     newfilename = newdirname / (filename.namebase + '.png')
     print('Making thumbnail %s -> %s' % (filename,
                  newfilename))
     out = subprocess.check_output(['convert', '-resize',
                   '100x100', filename, newfilename])

if __name__ == '__main__':
   make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

For various reasons, the path module was never incorporated into the standard library. The main reason may have been the kitchen-sink aspect of the whole implementation. Under the covers, the path object inherits directly from the built-in string type and adds more than 120 additional methods. As a result, it's a kind of "god object" that combines all of the functionality of strings, pathnames, files, and directories all in one place. To emphasize this point, there is the potential for confusion between string and path methods. For example:

```
>>> # A string method
>>> filename.split('/')
  [u'', u'Users', u'beazley', u'Pictures', u'img123.jpg']

>>> # A path method
>>> filename.splitpath()
(path(u'/Users/beazley/Pictures'), u'img123.jpg')
>>>
```

There are even methods for features you might not expect such as cryptographic hashing:

```
>>> filename.read_md5()
'\x98\x05\xdd\x97\xe0\xd3\x1f\xedH*xb\x179\xbf\x18'
>>>
```

It's a legitimate concern to wonder whether it's appropriate for a single object to contain every possible operation that one might think to do with a file—probably not.

Introducing pathlib

Starting in Python 3.4, a new standard library module pathlib was added to manipulate paths. It is the work of Antoine Pitrou and is described in some detail in PEP 428 [4]. As with previous efforts, it takes an object-oriented approach as before by defining a Path class. However, this class no longer derives from built-in strings. It's also much more refined in that it only focuses on functionality related to paths, and not everything that someone might want to do with a file in general.

To illustrate, here are some earlier examples redone using pathlib:

```
>>> from pathlib import Path
>>> filename = Path('/Users/beazley/Pictures/img123.
jpg')

>>> # Get the base directory name
>>> filename.parent
PosixPath('/Users/beazley/Pictures')

>>> # Get the base filename
>>> filename.name
'img123.jpg'
```

```
>>> # Get the file extension
>>> filename.suffix
'.jpg'

>>> # Get the file stem
>>> filename.stem
'img123'

>>> # Get the parts of the filename
>>> filename.parts
('/', 'Users', 'beazley', 'Pictures', 'img123.jpg')
>>>
```

Path also allows the / operator to be used to easily form new pathnames:

```
>>> filename.parent / 'thumbnails' / (filename.stem + '.png')
PosixPath('/Users/beazley/Pictures/thumbnails/img123.png')
>>>
```

Common operations for replacing/changing parts of the file name are also provided:

```
>>> filename.with_suffix('.png')
PosixPath('/Users/beazley/Pictures/img123.png')
>>> filename.with_name('index.html')
PosixPath('/Users/beazley/Pictures/index.html')
>>>
```

You will notice that in these examples an object of type PosixPath is created. This is system dependent—on Windows an object of type WindowsPath is created instead. Differences in the path implementation are used to support features such as case-sensitivity on the file system. For example, on Windows, you'll find that path comparison works as expected even if the file names have varying case:

```
>>> # Windows case-insensitive path comparison (only works on Windows)
>>> a = Path('pictures/img123.jpg')
>>> b = Path('PICTURES/IMG123.JPG')
>>> a == b
True
>>>
```

Last, but not least, pathlib provides a few basic functions for querying, directory walking, and other similar operations. For example, you can test whether a file matches a glob pattern as follows:

```
>>>> filename.match('*.jpg')
True
>>>
```

Here is a recursive glob over a directory structure:

```
>>> topdir = Path('/Users/beazley/Pictures')
>>> for filename in topdir.rglob('*.JPG'):
...  print(filename)
...
/Users/beazley/Pictures/Foo/IMG_0001.JPG
/Users/beazley/Pictures/Foo/IMG_0002.JPG
/Users/beazley/Pictures/Foo/IMG_0003.JPG
...
/Users/beazley/Pictures/Bar/IMG_1024.JPG
/Users/beazley/Pictures/Bar/IMG_1025.JPG
...
```

Putting this all together, here is an example of the thumbnail script using pathlib.

```
from pathlib import Path
import os
import subprocess

def make_thumbnails(topdir, pat):
  topdir = Path(topdir)
  for filename in topdir.rglob(pat):
     newdirname = filename.parent / 'thumbnails'
     if not newdirname.exists():
        print('Making directory %s' % newdirname)
        newdirname.mkdir()
     newfilename = newdirname / (filename.stem + '.png')
     out = subprocess.check_output(['convert', '-resize',
              '100x100', str(filename), str(newfilename)])

if __name__ == '__main__':
  make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

On the whole, I think you'll find the script to be bit cleaner than the original version using os.path. If you've used the third-party path module, there are a few potential gotchas stemming from the fact that Path objects in pathlib do not derive from strings. In particular, if you ever need to pass paths to other functions such as the subprocesss.check_output() function in the example, you'll need to explicitly convert the path to a string using str() first.

Final Words

I'll admit that I've always been a bit bothered by the clunky nature of the os.path functionality. Although this annoyance has been minor (in the grand scheme of things, there always seemed to be bigger problems to deal with), pathlib is a welcome addition. Now that I know it's there, I think I'll start to use it. If you're using Python 3, it's definitely worth a look. A backport to earlier versions of Python can be found at https://pypi.python.org/pypi/pathlib/.

References

[1] pathlib documentation: https://docs.python.org/3/library/pathlib.html.

[2] PEP 355: http://legacy.python.org/dev/peps/pep-0355/.

[3] path.py package: https://pypi.python.org/pypi/path.py.

[4] PEP 428: http://legacy.python.org/dev/peps/pep-0428/.

iVoyeur
Lies, Damned Lies, and Averages

DAVE JOSEPHSEN

Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

You know that movie where the guy takes hostages and duct-tapes them together and/or makes them wear gauche vests laden with random assortments of electronic components, and then demands all sorts of zany things like millions of dollars and a helicopter capable of flying him to Tahiti?

Sometimes I fantasize about being that guy. Not because I want to scare or harm anyone, and I certainly wouldn't wish those terrible vests on my worst enemy, but it *would* be fun to make zany commands over a bullhorn to a group of confused, yet eager to please, FBI agents.

Just think of the fun we could have. We could establish a holiday for things that are pickled. We could demand that every law-enforcement-related uniform and vehicle in the nation, regardless of jurisdiction, be painted pink (especially the drones, tanks, and mobile command-center RVs). We could bring back *Firefly*, banish Michael Bay AND George Lucas, force Starbucks to admit that granulated sugar really is sweeter than raw sugar...we could outlaw tactical vests.

You know, while we're on the subject, there is something that's been bothering me. Something for which I'd like to demand a fix. There's some talk going around lately about how we collect and persist metrics from systems and applications in the wild (a good thing) [1]. If I could strap ugly vests to people and demand something today, it might be a fix for one of my own metrics pet peeves that, for whatever reason, doesn't seem to have entered the discussion.

Metrics data is deceptively large because it's composed of such disarmingly innocuous little date/value tuples. It just seems unlikely that such harmless little measurements could possibly strain the storage device of even a respectable smartphone much less a grandiose server.

They add up, though. Every one of those little metrics, stored as a float, measured every five seconds, and persisted for a year, takes up around 400 MB of space. Two metrics from a single source, stored in their raw format, therefore, requires almost a gigabyte of storage. This modest storage conundrum is the primary hurdle to overcome in time-series data systems. We simply don't have the space to store thousands of measurements from hundreds of systems in their raw form, for periods of a year or more.

Enter Consolidation Functions

Most contemporary databases that are designed to store time-series data begin with a fundamental observation, namely, the older the data is, the less we care about it. If this is true, it means we don't actually need to store the raw measurements forever. Instead, we can keep the raw measurements for a short time and consolidate the older data points to form a smaller set of samples that adequately summarizes the data set as a whole.

This is usually accomplished automatically inside the datastore with a series of increasingly drastic data consolidations. You can think of the datastore itself as a series of time buckets. High-resolution, short-term buckets are very large. They can keep a bunch of data points in

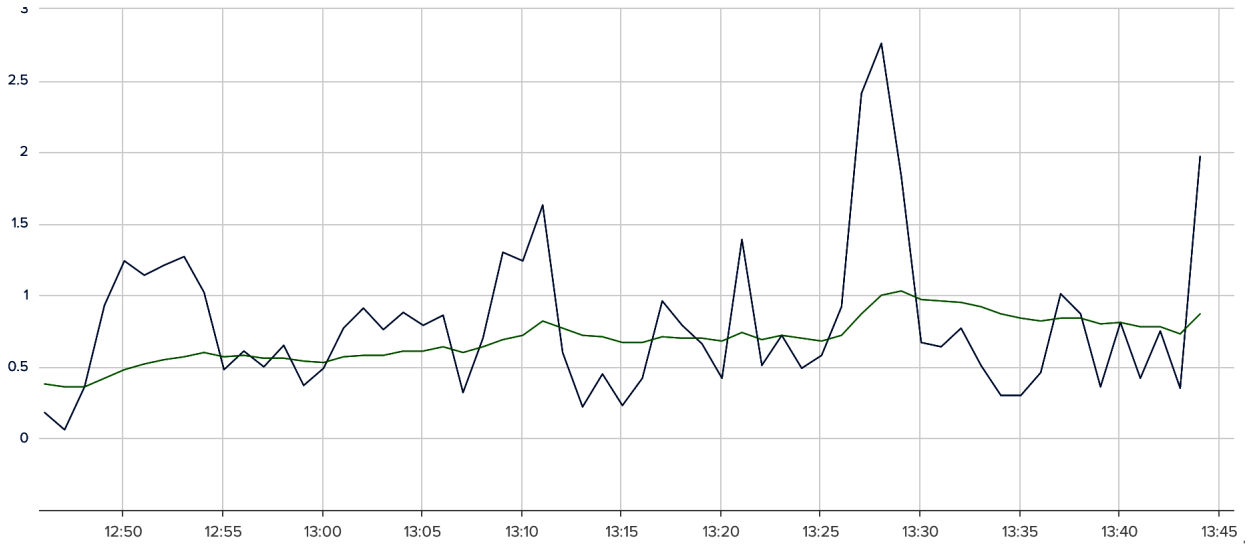


Figure 1: The effect of consolidating individual data points (the spiky line) using the arithmetic mean (the smooth line)

them, but longer-term buckets are smaller. As the data comes in, it's passed from bucket to bucket on a set interval; first into the *now* bucket, then into the *five seconds ago* bucket, and so on. Eventually, when the data points reach the *24-hours ago* bucket, they'll find that it's too small to fit them all. So they need to elect a representative to continue on for them, and so a means of carrying out this election must be chosen (this is not at all how they actually work internally, but it's a useful mental model).

As a user of these databases, you'll commonly need to configure a storage layout like the one laid out above, which, for example, stores raw measurements for the first 24 hours, then keeps one consolidated data point for each hour for the next two weeks, and then keeps one data point for every five hours, for six months, and etc. This summarization is a critically important piece of every time-series database. In a practical sense, it's what makes storing time-series data possible.

The databases that do automatic data summarization also expect you to control the method they use to consolidate the individual data points into summarized data points. Usually called the "summarization function" or "consolidation function," this is the means by which the database will decide who keeps going when the buckets get too small. You commonly need to configure this when you first create the datastore, and once set, it cannot be changed. This is dangerous, because your choice of consolidation function has a dramatic impact on the quality of your stored measurements over time, and although computing the arithmetic mean (AM) of all the data points in a period is a terribly destructive way to accomplish this, it's also by far the most commonly used consolidation function.

Averages Produce Below-Average Results

Using AM in this context is bad for two reasons. First, averages are horribly lossy. In the graph in Figure 1, for example, I've plotted the same data twice. The spiky line is the raw plot, while the smooth line is a five-minute average of the same data.

Second, averages are not distributive, which is to say, you start to get mathematically incorrect answers when you take the average of already averaged data. Both of these effects are detrimental in the context of monitoring computery things, because they have a tendency to smooth the data, when the peaks and valleys are often what we're really interested in.

Every time you create an RRD [2] with an RRA set to AVERAGE, or fail to modify the default storage-schemas.conf in Whisper [3], you're employing AM to consolidate your data points over time. These effects corrupt your data whenever you scale a graph outside the raw window or call a function that includes already averaged data.

Yes, even if your raw-window is 24 hours and your graph is displaying 24.5 hours, the entire data set you're looking at is averaged. If your raw-window is 24 hours, and you're calling a function to compare last week's data to this week's data, your entire data set has been averaged.

Worst of all, if your raw-window is 24 hours, and you're doing something like pulling a week's worth of data and running a function on it to depict it as thingies per hour instead of its native resolution (like for the marketing team or whatever), then you're looking at the average of already averaged data (once averaged for the rollup consolidation, and then again in the function to re-summarize it at a different scale). What you're seeing in this case is almost certainly mathematically incorrect.

To be sure, sometimes using the arithmetic mean is the best all around option, but if we all took a moment to fully understand the storage layer, and think about what we're measuring on a per-metric basis before we committed to the consolidation function, I think we'd pretty commonly choose one of the alternatives.

When I check the weather at wunderground.com, I don't get the average temperature for the day because that would be meaningless and silly. Instead, I get the max and min temperature for the day, and usually, because I'm a Texan, the max is the only value I care about.

Likewise, if I'm measuring 95th percentile inter-service latency, I want the *max*, which is an alternative consolidation function to average that drops all values in the period except the largest. This way, I preserve an accurate representation of the maximum 95th percentile latency value for that hour, or day, or week. In fact, in this example (like many others), the older the data gets, the more irrelevant the average becomes (and the more relevant the max).

Many of my day-to-day metrics are incrementor counters. That is, they're just +1s, adding up to some value that I don't actually care about, because I'm turning around and computing the derivative of that number to make it into a rate metric. So I don't even need to know the *value* of these metrics (because their value is always "1"), I really only need to know how many of them there are. For these, a consolidation function that just counts the number of measurements in each interval equates to lossless data compression.

Amazon.com shows me the average customer review score on every item I look at, but they can also give me a histogram of that data. Unfortunately, there is no *sum-of-squares* consolidation function in RRDtool or Whisper, but if there were, I could compute a statistical distribution from that value at display time.

Spread Data to the Rescue

So if it were me in the movie strapping vests to frightened extras, here would be my unreasonable demand this week: Let's store spread data in lieu of date/value tuples.

Imagine for a moment that you were building a system that needed to record and display at a one-second resolution of a metric that was being measured 400 times per second. In this example, there isn't a huge difference between just keeping the first metric that arrived in every one-second interval, or averaging all 400 together. No single measurement within the one-second is more important than any other. If the first measurement was extremely aberrant, I would probably choose to keep it over the average. The point is, even though we don't know what we're measuring, and even though we have 400 samples to average, the average of them still isn't as interesting as any single point in the set.

But it's a shame to throw away all of that wonderful data, even if you only strictly need 1/400th of it. I think most of us would like to have some idea of how it's distributed, some way of meaningfully combining those 400 measurements into something that is more significant than any single measurement alone. I think this is why it "feels" like taking the AM is the right thing to do. What if, instead of just storing a date/value tuple for this set, we stored something like this instead:

- date: What's the timestamp on this set?
- count: How many data points make up this set?
- sum: What's the sum of all data points in the set?
- min: What was the smallest value in the set?
- max: What was the largest value in the set?
- sos: What's the sum of squares for the set?

If we stored a struct like this instead of date/value, we wouldn't need to make the user choose a consolidation function when they created the datastore, because these data points self-summarize. When you need to consolidate them over a period of time, you compute the sum and sos, record the max, min, and count, and slap a new timestamp on it.

Even better, when the user wants a graph of this data, *then* you can ask them what they would like displayed. Do they want you to display the average value for the set? No problem, divide the sum by the count (this, by the way, ensures that you never average already averaged data). Do they want a min, max, sum, or count? No problem, display those things.

Notice that this struct doesn't even contain a variable to hold the original value of the measurement. That's because *value* is superseded for single measurements by sum, min, and max; all of those summarizations yield the correct value for an individual measurement (value/1 == value for averages, etc.), so you don't need to detect that case, it'll *just work* with the user-provided consolidation function at display time.

The drawback, of course, is that this struct is roughly 3x the size of a date/value tuple (assuming six floats instead of two), but I think fat data points are worth the stretch for a number of reasons. First, we could use fat data points as a better default consolidation function than arithmetic average. If the end-user wants to hard-code a consolidation function up front and gain a 3x reduction in storage requirements, that's a win for everyone, otherwise they get fat data points.

Second, some of the more modern data stores, like OpenTSDB, are eschewing consolidation entirely by making metrics collection a big data problem. I think fat data points fit very well between classic time/value stores like RRDtool and something like OpenTSDB that's going to require Hadoop infrastructure.

Finally, the future of metrics persistence is in purpose-specific data-handling layers built atop general-purpose databases like Cassandra, LMDB, and LevelDB. Graphite is moving in this direction with the Cyanite [4] project, and InfluxDB [5] was designed that way from the get-go. This trend is largely driven by the requirement to horizontally scale the persistence layer, and with that in place, the price of using fat data points is vastly reduced.

So let's all adopt fat data points before something happens to the imaginary hostages in my head. I think I speak for all of them when I say it's an easy fix that will simplify your time-series persistence layer while helping you preserve the integrity of your time series data.

Take it easy.

References

[1] Metrics 2.0: http://metrics20.org/.

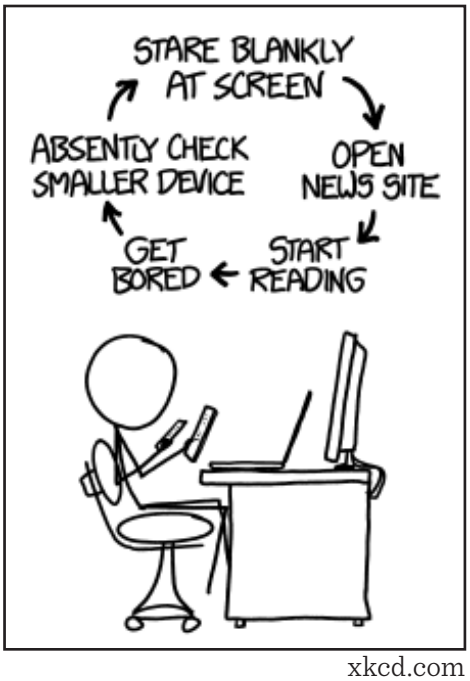
[2] RRDtool: http://oss.oetiker.ch/rrdtool/.

[3] Graphite, Whisper: graphite.wikidot.com/whisper.

[4] Cyanite: https://github.com/pyr/cyanite.

[5] InfluxDB: http://influxdb.com/docs/v0.8/advanced_topics/sharding_and_storage.html.

XKCD



xkcd.com

For Good Measure
Testing

DAN GEER

Dan Geer is the CISO for In-Q-Tel and a security researcher with a quantitative bent. He has a long history with the USENIX Association, including officer positions, program committees, etc. dan@geer.org

More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded; indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.—*Boris Beizer*

Testing for the presence of a characteristic is commonplace in all sorts of arenas including cybersecurity. In its simplest form, a test either returns True or False for a state of nature that is likewise either True or False. This leads to the classic 2x2 table:

	Truth	
Test	+	-
+	a	b
-	c	d

Using medical terms for the moment,

true positives
a = patients who do have disease and test positive

true negatives
d = patients who are without disease and test negative

false positives
b = patients who are without disease but test positive

false negatives
c = patients who do have disease but test negative

Expanding the table with row and column totals,

	Truth		
Test	+	-	
+	a	b	a+b
-	c	d	c+d
	a+c	b+d	t

we now have:

prevalence
(a+c)/t = fraction of population that has disease

sensitivity
a/(a+c) = fraction of those with disease who test positive

specificity
d/(b+d) = fraction of those without disease who test negative

predictive value positive
a/(a+b) = fraction of positive testers who actually have disease

predictive value negative
a/(a+b) = fraction of negative testers who are without disease

That collection of terms describe the nature of the test and what it is good for. Those working in information retrieval will know sensitivity as "recall" and predictive value positive as "precision."

If you have a highly sensitive test, then a negative test result is likely to be a true negative, and you can "rule out" disease in the patient. If you have a highly specific test, then a positive test result is likely to be a true positive, and you can "rule in" disease in the patient. Predictive value depends on the prevalence of the condition, while sensitivity and specificity do not. In other words, we can describe how good the test is without knowing prevalence, but we cannot say what an individual test result predicts without prevalence estimates. Specificity and sensitivity of a test are characteristics of the test independent of the population on which that test is used, while the predictive values positive and negative are dependent on those populations. Put differently, a test of constant specificity and constant sensitivity will have a different predictive value when the true rates of disease change (see below).

If a false negative is serious, such as when the treatment is painless and cheap but the disease is serious, you might favor a test with high sensitivity; re-imaging a virtual machine when there is any doubt about its integrity, say. If a false positive is serious, such as when the treatment is painful or costly while the disease is mild, you might favor a test with high specificity; skipping emergency patch rollout just to correct a spelling error, say.

A single test that is, at the same time, highly sensitive and highly specific is harder to engineer than you might think. As a rule of thumb, you cannot increase sensitivity and specificity at the same time. A multi-stage test is one where different tests are done sequentially. As such, the results of any one stage are conditional on the results of the previous stage. This can have significant economic impact.

For a reasonably rare disease, non-cases will strongly outnumber cases; hence, a negative test result is more likely. Working with that, you have a first stage (S1) that confirms negative status—i.e., it is highly sensitive resulting in false positives but, in turn, low false negatives. In other words, the first test releases as many as possible (and no more) from further work-up. The second stage (S2) wants no false negatives, so it is highly specific and, if indeed most subjects were rejected in the first stage, that second stage test can be quite expensive (and definitive). You can call Stage 1 "screening" and Stage 2 "confirmation" if you like. We have many parallels of this in cybersecurity:

- Router logs (S1) post-processed by log-analysis tools (S2)
- Anomaly detection (S1) reviewed by human eyes (S2)
- SIGINT traffic analysis (S1) to sieve which crypto is worth breaking (S2)
- Anti-virus heuristic scans with low detection threshold (S1) followed by direct malware process analysis (S2)

A worked example may make this clearer. Suppose you have a million people, lines of code, or whatever to screen, and the prevalence of what you are looking for is 1%—i.e., you want to cost-effectively find the 10,000 buried in the 1,000,000. This is what we know:

	Truth		
Test	+	-	
+			
-			
	10,000	990,000	10^6

We begin with a test that is sensitive but not especially specific—i.e., which misses few true positives at the cost of a meaningful number of false positives, and for which a negative result is not enormously meaningful. Let's say sensitivity is 99.99% and specificity is 90%,

	Truth		
Test	+	-	
+	*99.99%*	*10%*	
-	*.01%*	*90%*	
	10,000	990,000	10^6

meaning we now have:

	Truth		
Test	+	-	
+	9,999	99,000	108,999
-	1	891,000	**891,001**
	10,000	990,000	10^6

The predictive value negative is .999999 while the predictive value positive is .09. In other words, with a sensitivity of 99.99%, we get one false negative and we can forget about 89% of the pop-

ulation. Combined with the prevalence of 1%, a negative result is .999999 likely to be correct. Now we take just the remaining 108,999 and use a second test that has, for convenience, the reverse sensitivity and specificity, that is to say 90% sensitivity and 99.99% specificity. S2 thus returns:

	Truth		
Test	+	-	
+	8,999	10	9,009
-	1,000	98,990	**99,990**
	9,999	99,000	108,899

With a predictive value positive of .9989 and a predictive value negative of .99, we can forget an additional 99,990 test subjects. The big picture of S1 followed by S2 is therefore:

	Truth		
Test	+	-	
+	8,999	10	9,009
-	1,001	989,990	990,991
	10,000	990,000	10^6

We now have a compound result in which the predictive value of the compound test is high both for positives and for negatives—which is arguably what we would want, although debate may ensue on the downstream cost of a false negative versus a false positive.

89.99% sensitivity with 10 false positives
99.999% specificity with 1,001 false negatives

To further illustrate the cost-effectiveness of combining tests, let's say the cost of S1 is 30¢ while the cost of S2 is two orders of magnitude higher at $30.00. Everybody has to be tested in some way, but the question is by which protocol. Here are our four choices, with the results displayed graphically in Figure 1:

only S1 @ 30¢/test => $0.3M & 99,001 wrong
only S2 @ $30/test => $30M & 1,099 wrong
S1|S2 => $3.6M & 1,011 wrong
S2|S1 => $30M & 1,011 wrong

where "S1|S2" means S1 then S2 for only those which S1 did not rule out (and similarly for "S2|S1"). The calculation works like this for the S1-only line: Apply the 30¢ S1 test one million times costing $0.3M. That test will tell you that there are 108,999 cases to treat, so the cost of finding one "case" is $2.75, but you also get 99,000 false positives plus one false negative for a total of 99,001 that are wrong. The calculation for the S2-only line is parallel: Apply the $30 S2 test one million times costing $30M. That test will tell you that there are 9,099 cases to treat, so the

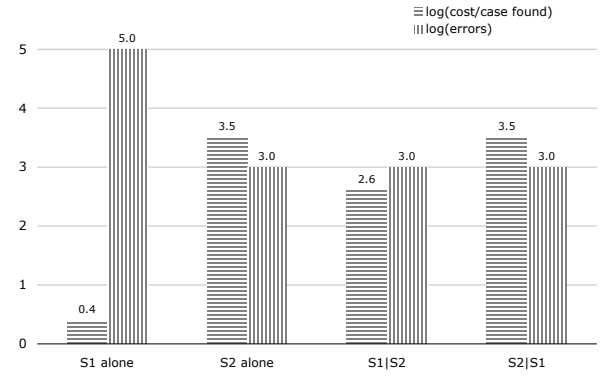


Figure 1: Cost and error rates for the four options, where the prevalence rate is 1%

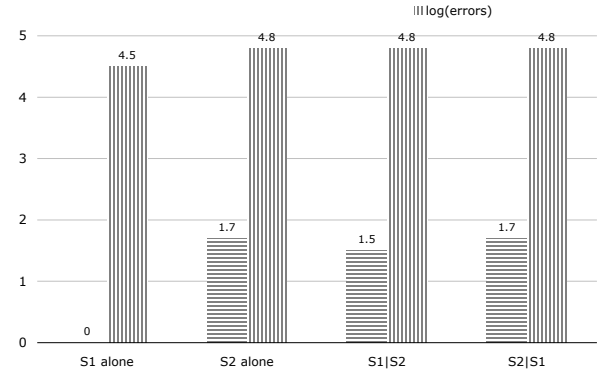


Figure 2: Same as Figure 1, but where the prevalence is 70%

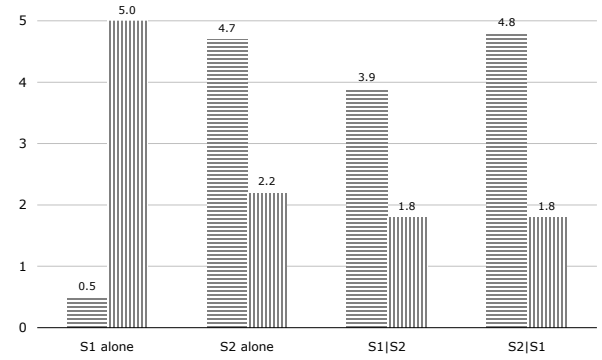


Figure 3: Same as Figure 1, but where the prevalence is 0.05%

cost of finding one case is $3,297.07, but you also get 99 false positives plus 1,000 false negatives for a total of 1,099 that are wrong.

Neither of the S1-only nor the S2-only testing protocols is attractive. If you do the S1 testing first and then the S2 testing on just those who tested positive with S1, then you've spent 30¢ one million times for the S1 stage plus $30 108,999 times for the S2 stage. The overall cost of finding one case, therefore, is $396.27

and you get 10 false positives plus 1,001 false negatives, for a total of 1,011 that are wrong. This is an improvement in cost-effectiveness, and that improvement is dependent on the order of testing: If you do the S2 testing first then the S1 testing on just those who tested positive with S2, then you've spent $30 one million times for the S2 stage plus 30¢ 9,009 times for the S2 stage. This means that the cost of finding one case is $3,331.01 and you still get those 10 false positives plus 1,001 false negatives, for a total of 1,011 that are wrong—i.e., the same total error rate but a lot poorer cost-effectiveness than the S1-then-S2 version.

Suppose the prevalence is not 1% but rather 70%. Then Figure 2 is what we have, and the decision on testing strategy is harder. On the other hand, if the prevalence is neither 1% nor 70% but rather 0.05%, then Figure 3 captures the situation. Comparing 1% prevalence to 70% prevalence to 0.05% prevalence highlights the choices to be made, and how they are dependent on the prevalence of the disease. Or, as we said above, a test of constant specificity and constant sensitivity will have a different predictive value when the true rates of disease change.

In summary, testing, including multi-stage testing, already has obvious roles in cybersecurity,

- AVS signature finding
- IDS anomaly identification
- Automated code analyses
- Firewall packet inspection
- Patch management performance

and we perhaps should know more about the terms and techniques used elsewhere rather than inventing new ones.

/dev/random
The Internet of Things

ROBERT FERRELL

Robert G. Ferrell is a fourth-generation Texan, literary techno-geek, and finalist for the 2011 Robert Benchley Society Humor Writing Award. rgferrell@gmail.com

As you read this, I will be a retired US federal government special agent, a job that I have for the most part studiously avoided mentioning over the past eight and a half years in these pages as a condition of being permitted herein to abide. I worked for the Department of Defense, which should really be called the Department of Unbridled Spending Except When it Comes to Employee Welfare, but that's a story over scotch and soda, or rather several scotch and sodas leading to straight scotch.

From this point on, I will consider myself solely a professional writer, at which pronouncement the less charitable among you will think, and perhaps even convey to the nearest editor, "When are you going to learn to write, then?" To this snide jab I have no answer, because I never read the "Letters to the Editor" for that very reason. Not that *;login:*, mercifully, features such an abomination formally. I know the editor pretty well, and he generally spares me the details of said missives, merely mentioning in passing that not *everyone* is a fan of /dev/random. To those malcontents I can only reply, in the concise and direct manner of my Gaelic ancestors, *póg mo thóin*. I expect, incidentally, that you will see a blank space or some innocuous phrase after the word "ancestors" in the preceding sentence, as the aforementioned editor will be horrified when he looks up what the phrase I wrote in Gaeilge means.

Today is a good day to di...I mean address the next spasm in the prolonged tetanic demise of the once noble TCP/IP, the Internet of Things™. Don't correct me if I'm wrong, but haven't we been fighting the SCADA security wars for a number of years now? Have we learned absolutely nothing from the threat of having our dams, factories, and traffic lights manipulated at will by a 14-year-old in a Minsk basement? Shall we now allow said juvenile delinquent access to our refrigerators, home security systems, and aquarium heaters? Have I used up my question mark quota for October yet?

Looking at the network source code for many of the devices on the IoT, it's as though we've regressed to 1990 and the ubiquity of Telnet. Plaintext authentication credentials (when there are any at all), no respect for egress filters, and rampant strcpy()-esque code flaws drag the IoT down security-wise to the point that it is more accurate to refer to it as the Internet of Targets. I've struggled through some dense and esoteric debates on the fridge-as-spam-relay topic, with respected infosec pundits asserting that this particular manifestation of embedded processor insecurity is not significant in the larger picture. Perhaps they're right, but if my household appliances are going to forward 419 scams and the expressions of interest by foreign women in a fictitious profile I never posted, I should at the very least be given options in that process.

For example, I would want some form of load balancing in place. If my refrigerator's processor is devoted to dispensing advertisements for erectile dysfunction treatments, it won't be very efficient at dispensing ice from the ice-maker or keeping my frozen yogurt from melting and leaking from the carton to coat the contents of my freezer in an uneven layer of crème fraîche. That computing task needs to be shared with, say, the smoke detector, while the thermostat and light dimmer can alternate pumping out pleas from friends and relatives who

had all their belongings stolen while on a sudden trip overseas and need some money wired to them.

Priorities need to be taken into consideration, as well. If the day's tasks consist of sending out notifications of huge lottery wins by random email addresses, scatter-gunning phishing attempts containing malicious links to everyone in some harvested database, and stock tips for non-existent securities, my appliances need to know in what order these tasks are to be carried out. There should be a master scheduler—perhaps the microwave or immersion blender—ensuring that the day's work is accomplished on time and in the most efficient manner. There's little worse than finding out your leased botnet took longer than it should have to distribute those 100,000 cheap wristwatch and generic drug spams because it had to cook dinner, turn on the sprinklers, or wash a load of grody old dishes.

I can see malware coming that, when installed on your domestic IoT, considers the functions for which the system was designed to be nuisance processes and kills them whenever they try to start. You might notice that you've lost control of your home appliances...or you might not. The second alternative is more intriguing, in the bleak dystopian world view that seems to be popular in the entertainment media these days. Any universe where Archie is murdered trying to stop the assassination of a gay senator is not a world I would choose to inhabit. Reggie, maybe.

Imagine, if you will, the White House of the relatively near future: filled with IoT gadgets to make the lives of the President, the First Family, and the White House staff more productive, efficient, and less cluttered. The Secret Service and Executive Office of the President have a squad of certificate-laden cyber-stars in charge of firewalling the bejeezus out of the internal network to keep those pesky hackers from taking control over the Royal Household. Sadly, like most canned "experts" born of boot camps rather than boots on the ground, they are ill-equipped for the task and miss some rather important entry points. One fine morning, the staff come to work to discover that not a single IoT device on the premises is functioning as expected.

At first, it's just annoying: Devices turn on or off when they aren't supposed to, settings change themselves, and so on. At some point, however, it escalates into something more sinister and intrusive, until at last the very lives of the people involved are at stake. The situation nosedives, spinning out of control until the President, who is currently airborne aboard Air Force One, can't even trust his own plane or pilots. Nothing is as it seems. World stability hangs in the balance, and you have to take the book into the bathroom with you because it's just that hard to put down.

Look for my future novel *Cybergeist* if you want to find out what happens. Man, I love being a writer.

Book Reviews

RIK FARROW AND MARK LAMOURINE

The Design and Implementation of the FreeBSD Operating System (2nd Edition)

Marshall Kirk McKusick, George V. Neville-Neil, and Robert N. M. Watson
Addison-Wesley Professional, 2014; 928 pages
ISBN 978-0-321-96897-5

Reviewed by Rik Farrow

This book comes out of a lineage of books about the BSD operating system, starting with *The Design and Implementation of 4.3BSD UNIX* in 1989. While its focus on FreeBSD sets this book apart from other operating systems books, where the focus is Linux, that's not all that sets it apart.

Kirk McKusick has been involved in key design decisions that still have bearing on UNIX-related systems since he was a graduate student sharing an office with Bill Joy. And this book reflects not only McKusick's influence on the designs of file systems and virtual-memory systems, but also that of its two other authors.

Whereas a book like Robert Love's *Linux Kernel Development* dives into getting, building, and examining kernel code, *Design and Implementation* stays at a higher level. Algorithms and data structures are explained, but so are the design decisions behind *why* a particular algorithm or design was chosen.

Soft updates provide a particularly contentious example. Early Linux file systems could create and delete files much faster than the 4.3 BSD fast file system (FFS), because the authors of ext2 had decided to do away with ordered, synchronous writes of file-system metadata. The FreeBSD developers' response, led by McKusick, was to create a process called *soft updates*, which allows metadata updates to occur asynchronously, but still in an ordered manner. In the Linux world, soft updates are spurned as too complicated. In this book, they are explained in clear and concise text, both why they are considered necessary and how they need to work. Approaches that log metadata updates are considered in the following section (the approach used in ext3).

Like operating systems books in general, the book begins with a history of UNIX (but written by one of its participants), followed by an overview of the kernel. Process management follows, then a completely rewritten chapter on security. If you are seriously interested in operating system security features, this chapter provides an excellent overview of the many mechanisms that have appeared, and been implemented, over the past 25 years. While the Linux security module and the related SELinux and type enforcement get only brief mention, there are thorough discussions of access control lists, mandatory access control, the new NFSv4 ACLs, security event auditing, cryptographic services, random number generator, jails, and the Capsicum capabilities model—a recent addition to FreeBSD.

The next chapter, on memory management, is just as long as the security chapter, and just as detailed. The next part of the book covers the I/O system, starting with overview, then devices in general, moving to FFS, then a new chapter on the Zettabyte File System. Again, this chapter would be useful to anyone who wants a deep understanding of ZFS, whether you are using FreeBSD, Linux, or Solaris descendants like illumos. The I/O section ends with a chapter on NFS, including NFSv4.

Part four covers Interprocess Communication, which begins with IPC and continues with chapters on network layer protocols, like IPv4 and IPv6, and transport layer protocols. The book concludes with a chapter on system startup and shutdown and a glossary.

Each chapter ends with exercises and one or more pages of references. The exercises cover ideas from each chapter and help the dedicated reader to think about potential solutions that go beyond what's covered in each chapter.

I did what I usually do with large technical books: I jumped around, after reading all of the introductory material, focusing on the parts I found most interesting. The writing makes this easy to do, in that I rarely found myself referred to another section in the book. This is not unlike the design of FreeBSD itself, which tends to be more modular than Linux.

In a world, especially an OS research world, dominated by Linux, you might really wonder why you would take the time to read a book on FreeBSD. The real reason is that there is a wealth of experience, a record of different approaches taken, written by three FreeBSD committers, all with stellar records. It would be a shame to miss out on all of this knowledge because of parochialism.

Think Bayes

Allen B. Downey
O'Reilly Media, Inc., 2013; 190 pages
ISBN 978-1-449-37078-7

Reviewed by Mark Lamourine

If you've read my previous reviews of Allen Downey's books, you'll know I'm a fan. His first three books covered Python programming, statistics, and complexity. His most recent is a practical exploration of Bayesian statistics, and I like this one as well.

Downey's purpose in all of his books is to set the reader on an exploration of the topic rather than to sit her down in a lecture hall. In each chapter or section, he introduces a real-world problem and then shows the reader the toolbox that will be needed to solve it. This usually includes external references to more in-depth treatments and often to primary source online data sets. His tone and style are very easy going, but this sometimes belies the difficulty and significance of the subject matter.

In *Think Bayes*, as in his other books, Downey aims to achieve something that might seem oxymoronic: applied theory. Bayes' theorem is derived in the first three pages of the first chapter. Everything else in the book is aimed at helping the reader learn what it *means*. You won't even see a lot of the hairiest statistical code. Downey provides a set of libraries that implement the tool set of statistical analysis: distributions and their characteristics. The code in the book illustrates how to use those libraries to model and then solve the problem at hand.

Downey displays a sense of lightness and humor in his selection of many of the problems and his approach to the solutions (though the Kidney Tumor problem was rather more somber). The problems include calculating the best solution to the Monty Hall problem, finding where a hidden paint-ball opponent is located using scatter of the paint ball hits on the walls of an arena, and estimating the number of bacterial species that inhabit the human belly button, the last from a real survey of human microfauna. If nothing else, the set of questions he addresses will provide hours of fun for curious geeks like me.

The real lesson in *Think Bayes* is how to recognize problems that are suited to Bayesian analysis and then how to model them. Building the model in code leads to a computable solution. This makes it relatively easy to understand the characteristics of the problem by tweaking the model or the inputs and observing how that affects the output. Downey uses the notation of continuous math when it is useful to describe a problem, but he concentrates on discrete solutions that are susceptible to computational solution. In the end, the reader (and experimenter) will come away with a deep practical understanding of this increasingly common set of analytical tools.

Becoming Functional

Joshua Backfield
O'Reilly Media, Inc., 2014; 135 pages
ISBN-13 978-144936817-3

Reviewed by Mark Lamourine

The proponents of functional programming have been gaining strength in recent years. Pure functional languages like Haskell are being used in production environments. Erlang, while not 100% functional, has strong functional traits and is heavily used in the telco industry. Functional features are being added to existing imperative languages such as Java, and these are giving the champions of functionality more room to play. Even Scheme and Lisp, the most venerable of functional languages, have had an academic niche for decades but are finding wider use.

Backfield isn't trying to claim you should use a pure functional language like Scheme or Haskell, or even adopt strict functional style in all cases. Rather, his goal is to demonstrate the tenets of functional programming using a mixture of imperative languages with some functional features (Java 7) as well as a couple of more functional languages (Groovy and Scala). Java 8 is getting full functional features like lambdas and closures, but Backfield avoids giving more than one or two examples in Java 8 because the Java 7 user base is well established and will have a long life even after 8 is released. His method is to introduce each concept in the context of refactoring some existing imperative code. This is in fairly stark contrast to some other books that teach functional programming using only formal lambda calculus and a pure functional language.

In the first chapter Backfield introduces the major techniques of functional programming. In each of the following chapters he details these techniques and contrasts them to the equivalent imperative code to do the same job. He also presents a chapter called "Functional OOP," showing how objects can still be used to contain related data while using class methods to provide namespacing for the related functions. In the final chapter, he offers an outline of a refactoring plan, first recognizing the imperative patterns and then applying the appropriate functional transformation.

The book is pretty slim for the depth of the content. Backfield doesn't spend any time on language syntax or constructs except as he applies them to the example at hand. This book probably is not a good choice for a beginning coder. Someone with multiple-language experience shouldn't have any problem though. I'm familiar with Java but not with either Groovy or Scala. The syntax is clear enough that this did not get in the way of understanding the point of each example.

I must say I'm not yet sold on the idea that functional programming is universally superior to traditional imperative style. Clearly, each has value. People don't naturally think in a functional style. It takes significant training and practice to do it well. Functional programming techniques like statement chaining quickly become clever obscurities unless they are well commented.

That said, *Becoming Functional* provides a good introduction to functional programming technique without going too deeply into theory. It's a book that I will probably keep nearby to help me recognize and exploit opportunities to use functional programming constructs where they seem to be the best solution to a problem.

A Go Developer's Notebook

Eleanor McHugh
Lean Publishing, 2014; 84 pages (and counting)
https://leanpub.com/GoNotebook

Reviewed by Mark Lamourine

Some interesting things are happening in the publishing world. Publishers large and small are experimenting with alternate ways of writing and distributing books. One recent trend is the release of "rough cuts" or "beta versions" of technical manuals. The maturation of the ebook and e-readers has made this possible and even easy. Some publishers have found that offering early access to new texts and inviting comment both drums up interest and improves the final result. There is even a new breed of publishers that uses this public development model as their core business. Lean Publishing is one of these, and Lean is where Eleanor McHugh is writing *A Go Developer's Notebook*.

I first encountered *A Go Developer's Notebook* in an announcement in the Go+ community on Google Plus. The book has its own community now as well, where readers make comments and Eleanor posts updates and progress reports.

McHugh starts off typically with the Go version of "Hello World," but she dwells on it as something more than a cliché. Through the first chapter, she enhances the simple CLI program until it's a small Web server that can respond with a customized hello based on the queries it receives. It can serve both HTTP and HTTPS running in concurrent routines, and it includes a signal handler to shut down the services cleanly when the process is interrupted. This is rather a lot to pack into an introductory chapter. The second chapter, entitled "Echo," is just as packed, covering CLI and environment input and string management.

The writing style and progression of examples are engaging and interesting. They don't always follow a traditional sequence, but they are coherent and introduce useful concepts. They also serve as an introductory survey of commonly useful standard packages and modules.

Part way through Chapter 2 is where the nature of the writing and publishing process becomes evident. This really is McHugh's notebook. She's clearly got an outline, but the chapter kind of peters out. The next chapter on Types picks up strong again. You're seeing the mind of the writer at work. Several of the chapters just have heading skeletons while others have sparse content.

There are only two more chapters that have significant content. The first provides examples of looping constructs in Go. The other is entitled "Software Machines" and seems to be about techniques of using goroutines to create simple machines like stacks, queues, and processor simulations. Neither one contains any of the normal explanatory texts yet. The code provides examples of more complex behaviors and usage. It takes some work to understand what it is meant to do, but they are definitely interesting to read.

A Go Developer's Notebook wouldn't be a bad introduction to Go syntax for an experienced coder. It's not nearly a complete text but what is there promises to become something good.

In a previous decade, this review would have been an indictment, not a recommendation, but then you would have only gotten one copy in paper with no possibility of getting updates or giving feedback. The author would have had to do a lot of up-front writing or pitch an idea to a publisher before getting any sense of whether readers would be interested.

With a service like Leanpub, the authors can put incomplete but promising ideas and the text in front of readers directly. They can "test" text and get responses from readers. They can do incremental updates to approach a working document.

This is the way software development works. Writing for humans too, but until recently it was always hidden behind editors and publishers. In traditional publishing, it would be unacceptable to let the reader pay for something that was flawed and incomplete.

The way Leanpub works, the author registers and creates the template for her book, sets the title, and uploads the content in whatever state it is in. She sets pricing and can also offer a preview chapter. When someone purchases a book, he gets a copy in the current state. He also gets email notifications of updates as they come. The author gets 90% of the payment and retains all of her rights. She is free to take the text to a traditional publisher at any time.

The pricing model is also flexible. The author sets a minimum price but can also suggest a retail price. McHugh has posted a minimum price of $6 US and a retail request of $22 US. The buyer decides how much the book (and updates) are worth. The author is paid a "royalty" of 90% after a base transaction fee of $0.50 US on the actual amount paid. That is, on each transaction, Leanpub keeps $0.50 US plus 10% of the remainder. This ensures that Leanpub gets something substantial from each transaction and encourages the authors to set a reasonable minimum price, low enough to feel reasonable to the buyer, but high enough to see some return for each sale.

Leanpub and other ebook self-publishing sites offer a good place for budding authors to float ideas and practice their writing. They also look like a good place to fish for new and different takes on all kinds of subjects, as long as you're aware of what you're looking at: the seeds, not the trees.

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

Buy the Box Set!

Whether you had to miss a conference or just didn't make it to all of the sessions, here's your chance to watch (and re-watch) the videos from your favorite USENIX events. Purchase the "Box Set," a USB drive containing the high-resolution videos from the technical sessions. This is perfect for folks on the go or those without consistent Internet access.

Box Sets are available for:

- **USENIX Security '14:** 23rd USENIX Security Symposium
- **3GSE '14:** 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education
- **FOCI '14:** 4th USENIX Workshop on Free and Open Communications on the Internet
- **HealthTech '14:** 2014 USENIX Summit on Health Information Technologies
- **WOOT '14:** 8th USENIX Workshop on Offensive Technologies
- **URES '14:** 2014 USENIX Release Engineering Summit
- **USENIX ATC '14:** 2014 USENIX Annual Technical Conference
- **UCMS '14:** 2014 USENIX Configuration Management Summit
- **HotStorage '14:** 6th USENIX Workshop on Hot Topics in Storage and File Systems
- **HotCloud '14:** 6th USENIX Workshop on Hot Topics in Cloud Computing
- **NSDI '14:** 11th USENIX Symposium on Networked Systems Design and Implementation
- **FAST '14:** 12th USENIX Conference on File and Storage Technologies
- **LISA '13:** 27th Large Installation System Administration Conference
- **USENIX Security '13:** 22nd USENIX Security Symposium
- **HealthTech '13:** 2013 USENIX Workshop on Health Information Technologies
- **WOOT '13:** 7th USENIX Workshop on Offensive Technologies
- **UCMS '13:** 2013 USENIX Configuration Management Summit
- **HotStorage '13:** 5th USENIX Workshop on Hot Topics in Storage and File Systems
- **HotCloud '13:** 5th USENIX Workshop on Hot Topics in Cloud Computing
- **WiAC '13:** 2013 USENIX Women in Advanced Computing Summit
- **NSDI '13:** 10th USENIX Symposium on Networked Systems Design and Implementation
- **FAST '13:** 11th USENIX Conference on File and Storage Technologies
- **LISA '12:** 26th Large Installation System Administration Conference

Learn more at: www.usenix.org/boxsets

USENIX ASSOCIATION FINANCIAL STATEMENTS FOR 2013

The following information is provided as the annual report of the USENIX Association's finances. The accompanying statements have been reviewed by Michelle Suski, CPA, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public Accountants. The 2013 financial statements were also audited by McSweeney & Associates, CPAs.

Accompanying the statements are charts that illustrate the breakdown of the following: operating expenses, program expenses, and general and administrative expenses. The operating expenses for the Association consist of the following: program expenses, management and general expenses, and fundraising expenses, as illustrated in Chart 1. The operating expenses include the general and administrative expenses allocated across the Association's activities. Chart 2 shows the breakdown of USENIX's general and administrative expenses. The program expenses, which are a subset of the operating expenses, consist of conferences and workshops, programs (including *;login:* magazine) and membership, student programs and good works projects, and the LISA Special Interest Group; their individual portions are illustrated in Chart 3.

The Association's complete financial statements for the fiscal year ended December 31, 2013, are available on request.

Casey Henderson, Executive Director

USENIX ASSOCIATION
STATEMENTS OF FINANCIAL POSITION
As of December 31, 2013 & 2012

ASSETS	2013	2012
Current Assets		
Cash & cash equivalents	$ 217,555	$ 552,100
Receivables	19,017	85,858
Prepaid expenses	53,434	74,531
Total current assets	290,006	712,489
Investments at fair market value	5,066,749	5,118,785
Property and Equipment		
Office furniture and equipment	370,609	364,776
Website	700,765	640,713
Leasehold improvements	29,631	29,631
Less: accumulated depreciation	(594,491)	(445,007)
Net property and equipment	506,514	590,113
Other assets	79,371	46,961
	$ 5,942,640	$ 6,468,348
LIABILITIES AND NET ASSETS		
Current Liabilities		
Accounts payable	$ 109,667	$ 490,299
Accrued expenses	42,466	95,554
Evi Nemeth Student Fund	1,250	-
Deferred revenue	31,540	35,250
Total current liabilities	184,923	621,103
Long-Term Liabilities	79,371	46,961
Total liabilities	264,294	668,064
Net Assets		
Unrestricted Net Assets	5,678,346	5,800,284
Net Assets	5,678,346	5,800,284
	$ 5,942,640	$ 6,468,348

USENIX ASSOCIATION
STATEMENTS OF ACTIVITIES
For the Years Ended December 31, 2013 & 2012

	2013	2012
REVENUES		
Conference & workshop revenue	$ 2,860,442	$ 3,510,227
Membership dues	300,666	326,834
Product sales	7,121	975
LISA SIG dues & other revenue	49,749	65,759
General sponsorship & ads	25,000	3,005
Total revenues	3,242,978	3,906,800
OPERATING EXPENSES		
Conferences & workshops	2,891,344	3,018,520
Membership ;login:	559,676	533,356
Projects & good works	18,580	44,889
LISA SIG expenses	71,001	82,367
Management and general	348,244	607,115
Fund raising	93,426	49,622
Total operating expenses	3,982,271	4,335,869
Net operating deficit	(739,293)	(429,069)
NON-OPERATING ACTIVITY		
Donations	23,305	-
Interest & dividend income	139,920	143,291
Gains on marketable securities	514,185	356,587
Loss on disposition of equipment	-	(3,987)
Investment fees	(60,097)	(62,341)
Other non-operating	42	(313)
Net investment income & non-operating expense	617,355	433,237
Change in net assets	(121,938)	4,168
Net assets, beginning of year	5,800,284	5,796,116
Net assets, end of year	$ 5,678,346	$ 5,800,284

Chart 1: USENIX 2013 Operating Expenses

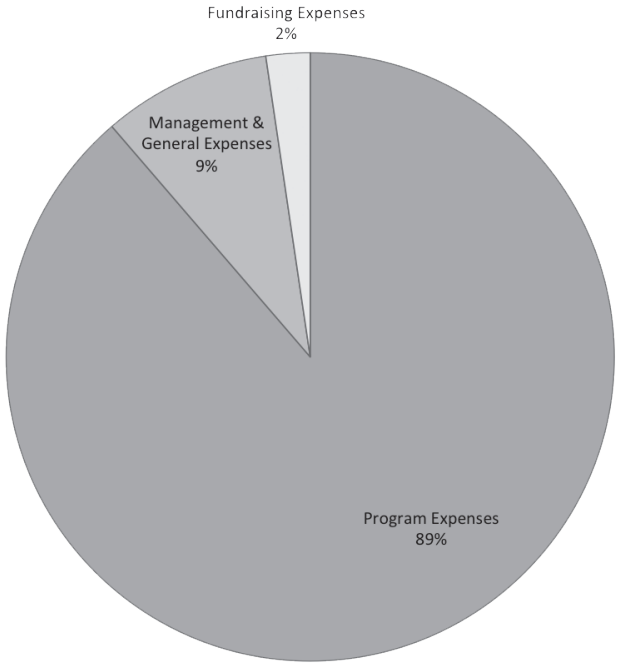


Chart 2: USENIX 2013 General & Administrative Expenses

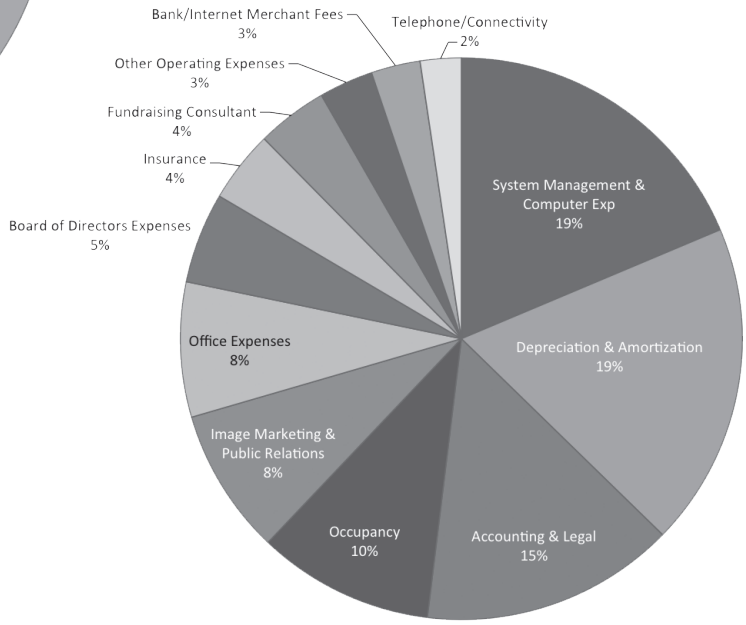
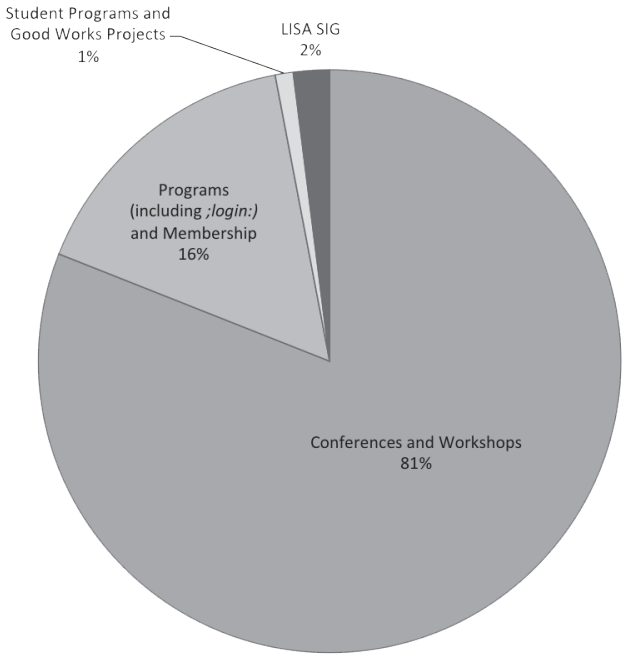


Chart 3: USENIX 2013 Program Expenses



Conference Reports

In this issue:

The reports from ICAC '14: 11th International Conference on Autonomic Computing and WiAC '14: 2014 USENIX Women in Advanced Computing Summit are available online: www.usenix.org/publications/login.

USENIX ATC '14: 2014 USENIX Annual Technical Conference

June 19–20, 2014, Philadelphia, PA

Summarized by Daniel J. Dean, Rik Farrow, Cheng Li, Jianchen Shan, Dimitris Skourtis, Lalith Suresh, and Jons-Tobias Wamhoff

Opening Announcements

Summarized by Rik Farrow (rik@usenix.org)

Garth Gibson (CMU) opened the conference by telling us that 245 papers were submitted, a record number. Of these, 49 were short papers. Thirty-six papers got accepted right from the start of reviews, and 11 papers were sent back to authors for revisions, resulting in eight more accepted papers for an 18% acceptance rate. Overall, there were 834 reviews by the 32-person PC, along with 16 additional reviewers. To fit 44 papers into two days, there were no keynotes or talks, just 20-minute paper presentations each crammed into two-hour sessions.

Nickolai Zeldovich (MIT), the co-chair, took over the podium and announced two best paper awards. The first was "In Search of an Understandable Consensus Algorithm," by Diego Ongaro and John Ousterhout, Stanford University. The other best paper award went to "HACK: Hierarchical ACKs for Efficient Wireless Medium Utilization," by Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp, University College London.

Brian Noble, the president of the USENIX Board, presented two of the three annual awards. Tom Anderson, Mic Bowman, David Culler, Larry Peterson, and Timothy Roscoe received the Software Tools User Group (STUG) award for PlanetLab. Quoting Brian as he made the presentation to Tom Anderson, "PlanetLab enables multiple distributed services to run over a shared, wide-area infrastructure. The PlanetLab software system introduced distributed virtualization (aka 'slicing'), unbundled management (where management services run within their own slices), and chain of responsibility (mediating between slice users and infrastructure owners). The PlanetLab experimental platform consists of 1186 machines at 582 sites that run this software to enable researchers to evaluate ideas in a realistic environment and offer long-running services (e.g., content distribution networks) for real users."

Almost as soon as he had sat down, Tom Anderson was on his way back to the podium to receive the USENIX Lifetime Achievement award, also known as the Flame Award. Again, quoting Brian as he read the text accompanying the award, "Tom receives the USENIX Flame Award for his work on mentoring students and colleagues, constructing educational tools, building research infrastructure, creating new research communities, and communicating his substantial understanding through a comprehensive textbook."

This year's ATC featured a second track, the Best of the Rest, where the authors of award-winning systems papers were invited to present their papers a second time. Eight out of the 11 people invited came to the conference, providing an alternate track.

Big Data

Summarized by Cheng Li (chengli@mpi-sws.org)

ShuffleWatcher: Shuffle-aware Scheduling in Multi-tenant MapReduce Clusters

Faraz Ahmad, Teradata Aster and Purdue University; Srimat T. Chakradhar, NEC Laboratories America; Anand Raghunathan and T. N. Vijaykumar, Purdue University

Faraz Ahmad presented his work on improving the performance of the resource utilization in multi-tenant MapReduce clusters. In these clusters, many users simultaneously submit MapReduce jobs, and these jobs are often running in parallel. The most time-consuming phase is to shuffle data from the finished map tasks to the scheduled reduce tasks. For example, 60% of jobs at Yahoo! and 20% of jobs at Facebook are shuffle-heavy. These jobs have negative impacts on the other jobs since they often run longer and incur high network traffic volume. Many related works tried to ensure fairness among jobs and often focused on how to improve throughput. In this work, the authors wanted to improve latency and throughput without loss of fairness.

Their main solution is to shape and reduce the shuffle traffic. To do so, they designed a tool called ShuffleWatcher, which consists of three different policies. The first policy is network-aware shuffle scheduling (NASS). NASS specifies that shuffle may be delayed if the communication and computation from different concurrent jobs are overlapping. Second, the Shuffle-Aware Reduce Placement (SARP) policy assigns reduce tasks to racks containing more intermediate data to make cross-rack shuffle traffic lower. Third, Shuffle-Aware Map Placement (SAMP) leverages the input data redundancy to locate map tasks to fewer racks to reduce remote map traffic.

To demonstrate the effectiveness of the joint work of these three policies, they evaluated the MapReduce jobs running with their tool and compared them to a baseline policy, the fair scheduler. They deployed all their experiments in Amazon EC2 with 100 nodes, each of which had four virtual cores. The results show improved latency and throughput and reduced cross-rack traffic.

Peng Wu (Huawei) wanted to know more about related and similar work in Google or Facebook. Faraz answered that some previous work also targets improvement in throughput and fairness, but they didn't optimize the map or reduce phases. He also said that he didn't know any public results from either Google or Facebook. The session chair, Anthony Joseph (UCB), asked about the interference introduced by longer jobs. Faraz replied that if the policy allows the jobs to run longer or to use more resources, then that is fine.

Violet: A Storage Stack for IOPS/Capacity Bifurcated Storage Environments

Douglas Santry and Kaladhar Voruganti, NetApp, Inc.

Douglas presented their work on persistent in-memory data structures. He said in-memory computing is important because applications like OLAP and OLTP, and ERP jobs cannot tolerate disk and network latencies. In addition to the in-memory computing, one still needs to store the data (e.g., memory state or transactions) on disk. There are two conventional solutions: (1) using a database to manage data and specifying your own table schema; and (2) designing new data structures and explicitly making state persistent. However, if memory is persistent, then developers don't need to map the memory state back to disk. It would be great if there were a persistence layer that divorced data structure selection and implementation from persistence.

Douglas et al. designed Violet, which introduces the notion of a named heap and replicates updates to memory or to persistent storage at the granularity of a byte. It also automatically enforces ordering and consistent image, and supports snapshot creation. There are two core parts in Violet: (1) a Violet library defines a set of data structures that will be automatically replicated to the memory and stored in disks; (2) Violet exposes to developers an interface which can be used to express memory updates and to group multiple updates into a transaction. To use Violet, developers only have to instrument their code with a few Violet keywords. The experimental results show that the instrumentation overhead is not significant. The asynchronous replication improves throughput numbers. The restore time decreases if the number of machines increases.

Somebody asked for a comparison between levelDB and Violet. Douglas replied that they are completely different, since levelDB is a key-value store. Developers can use Violet to build such a data store. The second question regarded what would happen while restoring data from disk if the virtual address is already taken. Douglas replied that Violet always maps the physical address to the same virtual address.

ELF: Efficient Lightweight Fast Stream Processing at Scale

Liting Hu, Karsten Schwan, Hrishikesh Amur, and Xin Chen, Georgia Institute of Technology

Liting Hu explained that buying things from Amazon involves other applications running to serve micro-promotions, likes, and recommendations. Besides these applications, there will also be data mining, for example, to predict games that will become popular. Liting Hu then displayed data flows for these various applications, where some flows are best processed in batches, others require more frequent processing as streams, and still others, like user queries, require immediate processing and millisecond response times. To provide this level of flexibility, the authors developed ELF.

Typically, data gets collected, using tools like Facebook's Flume or Kafka, into storage systems, like HBase. Instead, ELF runs directly on the Web server tier, skipping the just-mentioned data flow. ELF also uses a many master-many worker structure, with a peer-to-peer overlay using a distributed hash table to locate job masters. Job masters aggregate data from workers, then broadcast answers. ELF provides an SQL interface for programmers for reducing data on masters and workers. ELF also uses aggregation and compressed buffer trees (CBTs). In evaluations, ELF outperforms Storm and Muppet for large windows (more than 30 seconds) because of the ability to flush the CBTs.

Chuck (NEC Labs) asked whether they are running the ELF framework on the Web server itself, and Liting answered that yes, they run ELF as agents on Web servers. Chuck then asked whether they were concerned with fault-tolerance issues. Liting said that, yes, their DHT handled failure by finding new masters quickly. Derek Murray (Microsoft Research) asked how they do the streaming, sliding window for connecting components. Liting answered that the flush operation of CBTs allows them to adjust the size of the window, fetch pre-reduced results. Derek said he had a couple of more questions but would take them offline.

Exploiting Bounded Staleness to Speed Up Big Data Analytics

Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, and Gregory R. Ganger, Carnegie Mellon University; Phillip B. Gibbons, Intel Labs; Garth A. Gibson and Eric P. Xing, Carnegie Mellon University

Henggang presented this work on how to trade data freshness for better performance in big data analytics. Big data like Web pages and documents constitutes a huge amount of data. Analytics often have to perform computation over data iteratively. To speed up the computation, developers normally make iterative jobs run in parallel. However, to ensure the correctness, the sync operation is called periodically, and this pattern slows down the computation. The goal of this work is to reduce the sync overhead.

Henggang described their three core approaches to improve performance. The first approach is called Bulk Synchronous Parallel (BSP), in which every iteration is an epoch, and parallel jobs must synchronize with each other at the end of each

epoch. Arbitrarily-Sized BSP is a variant that allows developers to decide the size of epochs to do the synchronization. The last solution is called Stale Synchronous Parallel (SSP) and is more powerful than the previous two solutions. SSP could tune the number of epochs to call sync and also could allow different jobs to synchronize at different speeds.

They evaluated their approaches by running Gibbs Sampling on LDA jobs in eight machines, with the *NY Times* data sets as input. The results show that performance gets better if staleness increases. SSP performs better than BSP. Regarding convergence, more iterations are required if staleness increases.

The first question concerned the bound on the slowness of stragglers. Henggang replied that they have techniques to try to help stragglers catch up. Some people asked whether they checked the quality of results. He replied that they use the likelihood to measure the quality. In other words, if the likelihood is the same or close, then the generated models are qualified. The last question was about examples. He answered that they built the model to discover possible cancers, and compared the experimental results to a doctor's decision.

Making State Explicit for Imperative Big Data Processing

Raul Castro Fernandez, Imperial College London; Matteo Migliavacca, University of Kent; Evangelia Kalyvianaki, City University London; Peter Pietzuch, Imperial College London

Raul presented work on how to explore parallelism in programs written in imperative languages. He started his talk by showing that it is challenging to make imperative big data processing algorithms run in parallel and be fault tolerant, since the algorithms are stateful. On the other hand, many platforms like MapReduce, Storm, and Spark achieve good performance and manage fault tolerance by assuming there is no mutable state. The downside of using these platforms is that developers must learn new programming models.

This work aims to run legacy Java programs with good performance while tolerating faults. The key idea is to have a stateful data flow graph (SDG). In this graph, there are three elements: state element, data flow, and task element. The state element is often distributed, and has two abstractions: partitioned state and partial state. Partitioned state can be processed by a local task element, and partial state may be accessed by a local or global task element. Additionally, the partitioned state requires application-specific merge logic to resolve conflicts. To figure out which state is partitioned or partial requires programmers to provide annotations. To make the computation tolerate faults, they also implemented asynchronous snapshot creation and distributed recovery mechanisms.

Following his talk, some people asked about the size of their data sets used for experiments. Raul answered that the size of data sets varies from a few GBs to 200 GBs. The second question was about the principles used to specify either partitioned or partial. He answered that programmers need to know.

Virtualization

Summarized by Jianchen Shan (js622@njit.edu)

OSv—Optimizing the Operating System for Virtual Machines

Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov, Cloudius Systems

Nadav Har'El presented their research on OSv, which is a new OS designed specifically for cloud VMs. The goal of OSv is to achieve better application performance than traditional Linux. The OSv is small, quick booted, and not restricted to a specific hypervisor or platform. Nadav Har'El claimed that OSv was actively developed as open source so that it could be a platform for further research on VM OSes. OSv is developed using C++11 and fully supports SMP guests.

Nadav Har'El pointed out that the key design of OSv is to run a single application within a single process, multiple threads within a single address space, because the hypervisor and guest are enough to isolate the application just like the process is isolated in traditional OSes. There is no protection between user space and kernel space such that system calls are just function calls, which means less overhead. Nadav Har'El also emphasized that OSv entirely avoids spinlock by using a lock-free scheduler, sleeping mutex, and paravirtual lock. Basically speaking, there is no spinlock in OSv, so serious problems such as lock holder preemption are avoided. OSv takes advantage of network stack redesign proposed by Van Jacobson in 2006. OSv provides a new Linux API with lower overhead such as zero-copy lockless network APIs. Nadav Har'El suggested that we can improve performance further with new APIs and modifying the runtime environment (JVM), which can benefit all unmodified JVM applications.

Beside the evaluations that can be found in the paper, Nadav Har'El also showed some unreleased experimental results. For the Cassandra stress test (READ, 4 vCPUs, 4 GB RAM), OSv is 34% better. For Tomcat (servlet sending fixed response, 128 concurrent HTTP connections, measure throughput, 4 vCPUs, 3 GB), OSv is 41% better. Finally, Nadav Har'El invited people to join the OSv open source project: http://osv.io/.

Because there is no protection between user space and kernel space, the first questioner worried that if something evil is done in the user space, this could cause some security issues. Nadav Har'El answered that one VM would only run a single application, so only the application itself would be affected, and no damage would be made to other applications or the underlying hypervisor.

Gleaner: Mitigating the Blocked-Waiter Wakeup Problem for Virtualized Multicore Applications

Xiaoning Ding, New Jersey Institute of Technology; Phillip B. Gibbons and Michael A. Kozuch, Intel Labs Pittsburgh; Jianchen Shan, New Jersey Institute of Technology

Xiaoning Ding addressed the Blocked-Waiter Wakeup problem (BWW) and proposed its solution, Gleaner. As the number of

vCPUs in a virtual machine keeps increasing, Xiaoning Ding said that the mismatch between vCPU abstraction and pCPU behavior could prevent applications from effectively taking advantage of more vCPUs in each VM. Xiaoning Ding explained that vCPUs are actually schedulable execution entities, and there are two important features of a vCPU: First, when a vCPU is busy, it may be suspended without notification; second, when vCPU is idle, it needs to be rescheduled to continue computation, not like the physical CPU that can be available immediately for ready computation. For the synchronization-intensive applications that use busy waiting lock, the first feature would cause the Lock Holder Preemption problem (LHP). The vCPU holding the spinlock may be preempted, which makes other vCPU spend a long time waiting for the lock. For those applications that use blocking locks, the second feature would cause the BWW problem: Waking up blocked threads takes a long time on idle vCPUs. Through experiments, Xiaoning Ding showed that waking up a thread on pCPU only takes 8μs, but waking up a thread on vCPU takes longer than 86μs, which is also variable. The major source of overhead and variation comes from the vCPU switch. So that's why BWW would increase execution time and incur unpredictable performance and reduced overall system performance.

Xiaoning Ding pointed out that the LHP problem has been well studied. So their team focused on the solution to the BWW problem. Generally, the goal is to reduce harmful vCPU switching, and there are two main methods to deal with this. First is resource retention: preventing an idle vCPU from being suspended by letting it spin instead of yielding hardware resources, although this may cause resource under-utilization. Second is consolidation scheduling: consolidating busy periods and coalescing idle periods on vCPUs. This method activates some vCPUs to avoid vCPU switching and suspends other vCPUs to save resources. However, the problem is that the active vCPUs may be overloaded. Some active vCPUs may undertake too heavy a workload because some workloads cannot be evenly distributed among active vCPUs. Gleaner basically takes advantage of both of these two methods. At the same time, Gleaner provides a solution to the overloading problem, gradually consolidating workload threads only if the following conditions are satisfied: vCPU utilization would not be too high after consolidation and workloads can be evenly distributed among active vCPUs. Also, Gleaner stops consolidation when the throughput tends to decrease. Xiaoning Ding concluded that the evaluations prove that Gleaner can improve application performance by up to 16x and system throughput by 3x.

Someone was interested in whether Gleaner could also perform well in other hypervisors like Xen. Xiaoning Ding replied that the current evaluation was only done in KVM, so further experiments may need to be done across different hypervisors. Xiaoning Ding was also asked why there is still some slowdown relative to bare-metal performance in the current experimental results. He answered that this slowdown may still be caused by the overloading problem, which cannot be entirely prevented.

HYPERSHELL: A Practical Hypervisor Layer Guest OS Shell for Automated In-VM Management

Yangchun Fu, Junyuan Zeng, and Zhiqiang Lin, The University of Texas at Dallas

Yangchun Fu explained that current cloud services or datacenters usually host tens of thousands of virtual machines, which require large scale and automated management tools. Traditional management tools are placed in the user space of the host OS and work with the management utilities installed in the guest OS. Although the management is centralized, each VM is required to install the client utilities, and the user-space tool also needs the admin password to access the VM, which is painful when dealing with large scale. Hence Yangchun Fu's team proposed the HYPERSHELL, a practical hypervisor layer guest OS shell tool for automated in-VM management, which only installs the management utilities at the hypervisor layer.

Yangchun Fu stated that the system call is the only interface to request OS service, so HYPERSHELL introduces the reverse system call to achieve the guest OS management. The main contribution of the reverse system call is bridging the semantic gap for the hypervisor layer program between the guest and host OS so that the hypervisor can interpret semantic information about the guest OS. The reverse system call technique would dispatch the system call from the host library space program. To differentiate the host and guest system call, Yangchun Fu's solution would add an extra value to the file descriptor, since it is just an index and has a limited maximum value. The system call is arranged to execute by hypervisor. The hypervisor along with a helper process inside the guest OS's user space would inject the transferred guest system call. Yangchun Fu explained that the helper process is created by the hypervisor layer program and would inject the guest system call right before entering the kernel space or exiting to the user space. And because the system call data is saved and exchanged in shared memory, many of the current guest OS management utilities can be directly reused in HYPERSHELL without any modification.

Yangchun Fu said they evaluated about a hundred management utilities and demonstrated that HYPERSHELL has relatively little slowdown and overhead on average compared to their native in-VM execution. More detailed information could be found in the paper. Yangchun Fu concluded that HYPERSHELL will circumvent all existing user logins and the system audit for each managed VM, so it cannot be used for security-critical applications unless special care is taken for these uses. HYPERSHELL requires both OSes running in the host OS and VM to have a compatible system-call interface. But, if further work can be done in additional system-call translation, the HYPERSHELL can work with more kinds of OSes.

Someone asked whether there is any extra overhead when HYPERSHELL is used for monitoring. Yangchun Fu replied that there is no extra overhead.

XvMotion: Unified Virtual Machine Migration over Long Distance

Ali José Mashtizadeh, Stanford University; Min Cai, Gabriel Tarasuk-Levin, and Ricardo Koller, VMware, Inc.; Tal Garfinkel; Sreekanth Setty, VMware, Inc.

Ali José Mashtizadeh explained that previous live virtual machine migration only happened between the machines that shared local shared storage, such as a cluster with a storage array, and only the memory is migrated. But faster networks and current VM deployment, which usually hosts VMs on tens of thousands of physical machines over sites far away from each other, have made necessary a reliable and efficient wide area virtual machine migration technique. Current ad hoc solutions are often complex and fragile, however, so Ali José Mashtizadeh and his team have proposed a solution called XvMotion, which is an integrated memory and storage migration system that does end-to-end migration between two physical hosts over the local or wide area. XvMotion offers performance and reliability comparable to that of a local migration and has been proven to be practical for moving VMs over long distances, such as between California and India, over a period of a year.

Ali José Mashtizadeh said that the architecture of XvMotion contains the live migration and I/O Mirroring modules in ESX. Between the source and destination, the Streams layer handles the large data transfer, including memory pages and disk blocks on top of the TCP network. The live migration module would be responsible for regular tasks as found in local live migration. The I/O Mirroring would record any changes during the copy, which is asynchronously implemented with Streams and the disk buffer. This process can reduce the impact on the source VM from high and unstable network latency. Ali José Mashtizadeh pointed out that in long distance memory migration, if the network transferring rate were slower than the VM's workload changing page rate, then there would be an inconsistency issue and convergence would not occur. In traditional local networks, the VM would be halted and transfer all remaining dirty pages during downtime. This solution is not acceptable, because long distance incurs long latency, which would lead to longer downtime.

XvMotion's solution is Stun During Page Send, which can inject latency in page writes operation to throttle the page dirtying rate to a desired level when it is faster than the network transmit rate. In terms of disk buffer congestion control, Ali José Mashtizadeh refers listeners to the paper for details. The evaluation of XvMotion showed it provides stable migration time and downtime (atomic switchover) less than one second even for latencies as high as 200 ms and data loss up to 0.5%. Downtime only has a small linear time increase with distance. In addition, the guest workload performance penalty is nearly constant with respect to latency, and only varies based on workload intensity.

GPUvm: Why Not Virtualizing GPUs at the Hypervisor?

Yusuke Suzuki, Keio University; Shinpei Kato, Nagoya University; Hiroshi Yamada, Tokyo University of Agriculture and Technology; Kenji Kono, Keio University

Yusuke Suzuki described GPUvm, a technique to fully virtualize GPU at the hypervisor. Currently, GPUs are used not only for graphics but also for massively data-parallel computations. GPGPU applications are now widely accepted for various use cases. However, the current GPU is not virtualized, and we cannot multiplex a physical GPU among virtual machines or consolidate VMs that run GPGPU applications. Yusuke Suzuki said that GPU virtualization is necessary. There are now three virtualization approaches: I/O pass-through, API remoting, and paravirtualization. I/O pass-through assigns a physical GPU to VM dedicatedly. The problem is that multiplexing is impossible. API remoting can multiplex GPUs because it forwards API calls from VMs to the host's GPUs. But API remoting needs host's and VM's API and its version to be compatible. The paravirtualization approach provides an ideal GPU device model to VMs. But each VM uses PV-drivers, which must be modified to follow the ideal GPU device model. The goal of GPUvm is to allow multiple VMs to share a single GPU without any driver modification. At the same time the performance bottlenecks of full virtualization are identified and optimization solutions are accordingly provided in GPUvm.

Yusuke Suzuki explained that there are three major components in a GPU: GPU computing cores, GPU channels, and GPU memory. The GPU channel is a hardware unit to submit commands to GPU computing cores. Memory accesses from computing cores are confined by GPU page tables. GPU and CPU memory spaces are unified. GPU virtual address (GVA) is translated into CPU physical addresses as well as GPU physical addresses (GPA). So the basic idea of GPUvm is to virtualize these three major components. The GPU memory and channels are logically partitioned to virtual ones. And time sharing is employed to make GPU cores shared by several VMs. As a result, each VM would have a set of isolated resources. GPUvm works as a hypervisor, which exposes a virtual GPU device model to each VM. Hence the guest GPU driver needn't be modified.

In GPUvm, the GPU shadow page table isolates GPU memory. GPU shadow channel isolates GPU channels. And GPU fair-share scheduler isolates performance on GPU computing cores. Memory accesses from GPU computing cores are confined by GPU shadow page tables. Since DMA uses GPU virtual addresses, it is also confined by GPU shadow page tables, which guarantees that DMA requested from one VM cannot gain access to CPU memory regions assigned to other VMs. For the GPU channels, mapping tables are maintained to partition the physical channels. Finally, for equitable use of GPU cores, the BAND scheduling algorithm [Kato et al. '12] is used, since the GPU command execution is non-preemptive and BAND is aware of this feature. Their evaluation showed that for long non-preemptive tasks, BAND can provide better fairness compared to some traditional

scheduling algorithms. Yusuke Suzuki said the raw full GPU virtualization may incur large overhead.

Besides describing the architecture of GPUvm, Yusuke Suzuki also introduced some optimization techniques. First, GPUvm allows the direct BAR accesses to the non-sensitive areas from VMs. This reduces the cost of intercepting MMIO operations. Second, GPUvm delays the updates on the shadow table, since this operation is not needed until the channel is active. Third, GPUvm provides a paravirtualized driver to further reduce the overhead of the shadowing table. The evaluation used Gdev (open-source CUDA runtime) and Nouveau (open-source device driver for NVIDIA GPUs) on Xen. The paravirtualized version performed best but is still 2–3x slower than Native execution. Also, GPUvm would incur large overhead in cases involving four and eight VMs without paravirtualization, since page shadowing locks GPU resources.

Yusuke Suzuki and an attendee discussed the pros and cons of GPU hardware virtualization and software virtualization. Yusuke Suzuki pointed out that changing scheduling cannot be supported in hardware virtualization. Someone also suggested methods like killing long tasks to achieve preemption, so that the traditional scheduling algorithm can also provide good fairness in GPU hardware virtualization.

A Full GPU Virtualization Solution with Mediated Pass-Through

Kun Tian, Yaozu Dong, and David Cowperthwaite, Intel Corporation

Kun Tian presented their work on GPU virtualization and began with GPU virtualization's three requirements: native GPU acceleration performance, full features with consistent visual experience, and sharing among multiple virtual machines. Among existing methods, API forwarding can share a single GPU for several VMs, but the overhead and API compatibility problem limits its performance and the features supported. Another method for VMs to use a GPU is direct pass-through. It provides 100% native performance and full features, but no sharing at all. To meet the above three requirements, Kun Tian proposed their team's solution called gVirt, which supports full GPU virtualization and combines mediated pass-through with a performance boost. gVirt can provide full-featured vGPU to VM and avoid VM modifying the native graphics driver. gVirt can also achieve up to 95% native performance and scale up to seven VMs.

Kun Tian said that gVirt is open source and the current implementation is based on Xen, codenamed XenGT, with KVM support coming soon. gVirt supports Intel Processor Graphics built into fourth-generation Intel Core processors. But Kun Tian mentioned that the principles behind gVirt can also apply to different GPUs, since most modern GPUs only have major differences in how graphics memory is implemented. gVirt is trademarked as Intel GVT-g: Intel Graphics Virtualization Technology for virtual GPU. Kun Tian said the main challenges of GPU virtualization are complexity in virtualizing, efficiency when sharing the GPU, and secure isolation among the VMs. In terms of efficiency when sharing the GPU, gVirt takes advantage of mediated pass-through, which passes through performance-critical operations but traps and emulates privileged operations. Trap-and-emulation can provide a full-featured vGPU device model to VMs and use the shadow GPU page table.

In terms of sharing, gVirt partitions the GPU memory and avoids address translation by employing address space ballooning. In terms of secure isolation, among many problems Kun Tian gave an example of the most important one: the vulnerability from direct execution. Although the direct command needs to be audited when using mediated pass-through, after gVirt has received, audited, and submitted the command to the GPU, the GPU may execute the modified command where something evil happens. The solution is smart shadowing: lazy shadowing the statically allocated ring buffer and write protection of the batch buffer, which is allocated on-demand. These two methods can prevent the modification of command's content or prevent the modified command from being executed. At last, Kun Tian said, they found the overhead mainly comes from the power management register's accesses operation, which is unnecessary in VM. So gVirt removed these operations and added more commands submitted in some benchmarks, which led to improved performance.

Someone asked about the progress of the project. Kun Tian said that many more small changes have been included in the newest version. He also provided the publicly available patches for further reference: (1) https://github.com/01org/XenGT-Preview-xen; (2) https://github.com/01org/XenGT-Preview-kernel; (3) https://github.com/01org/XenGT-Preview-qemu.

Best of the Rest I

Summarized by Cheng Li (chenglii@cs.rutgers.edu)

Naiad: A Timely Dataflow System

Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi, Microsoft Research

Derek Murray said that a new programming model is required for new applications. A timely dataflow system should be able to handle batch processing, stream processing, and graph processing applications. For example, how would one build a system to handle large social networks that minimizes latency for coordination?

Murray revisited a number of dataflows such as parallelism and iteration. Then he showcased and contrasted the two systems using batching and streaming applications. Batching systems require coordination but need to support aggregation. Streaming systems do not require coordination but aggregation is difficult.

Naiad is different because it is tightly coupled with the execution mode and uses batch processing, stream processing, and graph processing. The timely framework will update results in real-time, minimizing the latency through coordination of events. The timely system supports asynchronous and fine-grained synchronous execution. To achieve low latency, three techniques

were used: a programming model, a distributed progress tracking protocol, and system performance engineering were proposed.

The programming model is event based. Each operation corresponds to an event. Messages are delivered asynchronously. Notifications will support batching. These programming frameworks leverage a timely dataflow API to run programs in a distributed manner. To achieve low latency, Naiad proposes a distributed progress tracking protocol that assigns a timestamp to each event. Sometimes, an event may depend on its own output. So a structured timestamps-in-loops solution was proposed. One challenge of performance engineering in Naiad is to reduce micro-stragglers that negatively impact the overall performance.

The evaluation results showed that Naiad can achieve the design goals with low overheads. Murray demonstrated PageRank, interactive graph analysis, and query latency. His conclusion was that Naiad achieved the performance of specialized frameworks and provided the flexibility of a generic framework.

One question was whether the generic programming model, in providing a fair share in a multi-tenant environment, ran into resource competition problems. Murray answered no. Someone asked how to do fault tolerance. Murray said query and log the state.

Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior

Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, Armando Solar-Lezama, MIT CSAIL

Xi Wang used an example (digit overflow) to demonstrate that compiler optimization flags may optimize aggressively or discard some of the sanity checks and change the accuracy of the results. Then he used a table to show that it is a common error for widespread GCC versions.

The notion of undefined behavior refers to the behaviors that are not defined by the specifications. The original goal is to emit efficient code, but it allows the compilers to assume a program never invokes undefined behaviors. So this ambiguity causes many undefined behaviors. Therefore, there is a need for a systematic approach to study and control undefined behavior.

The methodology is to use a precise flag to annotate unstable code and compare the two versions with or without the unstable code. A Boolean satisfaction table is used to justify the behavior of two programs when unstable code is enabled or disabled. A framework, STACK, is proposed to identify unstable code. STACK makes the compute assumption as no undefined behavior. Then STACK will run the two versions and compare the diffs. The limitation is missing unstable code and false warnings. The presenter addressed these issues separately.

The evaluation shows STACK is robust to find unstable code and scales to large code bases. The presenter also made suggestions on how to avoid unstable code.

Storage

Summarized by Daniel J. Dean (djdean2@ncsu.edu)

vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment

Fei Meng, North Carolina State University; Li Zhou, Facebook; Xiaosong Ma, North Carolina State University and Qatar Computing Research Institute; Sandeep Uttamchandani, VMware Inc.; Deng Liu, Twitter

Fei Meng began by describing how SFC can be used to accelerate I/O performance, reduce I/O disk load, and reduce contention. Fei then discussed how current SFC management techniques, including static partitioning and globally shared SFC space. Both have issues. To address these challenges, Fei introduced vCacheShare, which can dynamically resize the cache as needed. He also described how vCacheShare takes both the long term and short term into account when deciding the cache size. The four modules of vCacheShare are: (1) a cache module, responsible for cache management; (2) a monitor, responsible for tracing cache usage; (3) an analyzer, responsible for analyzing cache usage based on the traces from the monitor; and (4) an optimizer, responsible for optimizing cache usage. Finally, Fei discussed the results of micro- and macrobenchmarks they ran that demonstrated the effectiveness of their tool.

The first question was whether the number of VMs affects the system. Fei answered that the number of VMs does not affect the system. Someone else asked whether context switches affect the system, to which Fei answered they do not.

Missive: Fast Application Launch From an Untrusted Buffer Cache

Jon Howell, Jeremy Elson, Bryan Parno, and John R. Douceur, Microsoft Research

In this talk, Jon Howell described how to quickly launch relatively large applications from an untrusted buffer cache. The authors based their work on the Embassies system, which uses an ultra-lightweight client to ensure applications are isolated and self-contained. By making the client small, however, a large amount of application data will need to be sent over the network, causing applications to take a long time to launch.

To address this issue, Jon described how they collected and analyzed 100 "best-of" applications in order to find ways to speed up the launch process. They found that there was a lot of commonality among the applications, which they could exploit to launch applications faster. Specifically, they used zar files and Merkle trees to only send what was absolutely necessary to launch the application in an efficient way. Some of the details Jon emphasized were how zarfile packing works. Specifically, large files are packed first and smaller files are packed in between, leading to very little wasted space. By using this approach, Jon then showed how it was possible to launch relatively large applications in hundreds of milliseconds.

Someone asked whether they tried to optimize anything. Jon answered that they had not yet optimized anything and wanted to first show a proof that the idea could be done.

A Modular and Efficient Past State System for Berkeley DB

Ross Shaull, NuoDB; Liuba Shrira, Brandeis University; Barbara Liskov, MIT/CSAIL

Ross Shaull described Retro, a system for snapshotting and past state analysis for Berkeley DB. He began by describing why it is useful to know the past states of the DB, using several examples such as auditing, trend analysis, and anomaly detection. The problem is that saving past states is difficult, and not all data stores provide the ability to do it. Ross then described Retro, a low overhead snapshot system for Berkeley DB.

Ross discussed how Retro was designed with simplicity in mind, making choices such as extending existing Berkeley DB protocols instead of creating new protocols. He then described how Retro snapshots are consistent, global, named, and application-declared. Retro works by tagging which pages to save, which pages are part of a query, and which pages to recover when a crash occurs. Ross also described how recovery with Retro works. Specifically, Retro saves pre-states during Berkeley DB recovery, then reads a page from disk and applies redo records to it. Finally, Ross showed that Retro is non-disruptive, imposing 4% throughput overhead. He also conceded, however, that Retro does require a separate disk for good performance.

The first question was how has this advanced the state of the art. Ross replied that the past-state system is integrated into a different system layer, which can be inserted into the database near the page cache.

SCFS: A Shared Cloud-backed File Systems

Alysson Bessani, Ricardo Mendes, Tiago Oliveira, and Nuno Neves, Faculdade de Ciências and LaSIGE; Miguel Correia, INESC-ID and Instituto Superior Técnico, University of Lisbon; Marcelo Pasin, Université de Neuchâtel; Paulo Verissimo, Faculdade de Ciências and LaSIGE

Alysson Bessani described SCFS, a shared cloud-backed file system that is designed to provide reliability and durability guarantees currently lacking in existing systems. Alysson began by describing the two main classes of shared storage: local software, which interacts with a backend (e.g., Dropbox), and direct access-based systems (e.g., BlueSky). While these systems work well, they do not make any reliability or durability guarantees and instead offer a best effort approach.

To address this challenge, the authors designed SCFS. Alysson discussed a key idea behind their approach: always write and avoid reading; in cloud-backed storage systems, writes are essentially free while reading is typically more expensive. To do this, they use a consistency anchor in order to make sure everything done locally is consistent with whatever is in the cloud. They use a consistency service to ensure the correct version of a file is obtained initially, they then perform all operations on the file locally, and finally push it to the cloud when the file is closed. Another point Alysson discussed was how SCFS can use multiple backends in order to ensure data is available even when faced with data-corruption or service unavailability. Finally, the experiments they conducted demonstrated the pros and cons of their approach under various modes of operation (e.g., blocking vs. non-blocking).

Someone asked whether they compared FS Cache to their work. Bessani answered that the files are only sent to the cloud when calls close. The questioner also wondered what files were cached, to which Bessani answered that all locally modified files were cached, with the master version being on the cloud.

Accelerating Restore and Garbage Collection in Deduplication-based Backup Systems via Exploiting Historical Information

Min Fu, Dan Feng, and Yu Hua, Huazhong University of Science and Technology; Xubin He, Virginia Commonwealth University; Zuoning Chen, National Engineering Research Center for Parallel Computer; Wen Xia, Fangting Huang, and Qing Liu, Huazhong University of Science and Technology

Min Fu described how to accelerate the restore and garbage collection process in deduplication-based systems. He began by describing how fragmentation is a major problem for these systems and how it can negatively affect garbage collection and restoration. Specifically, Min discussed how sparse and out-of-order containers cause problems.

To address this problem they have developed a history-aware rewriting algorithm that reduces sparse containers. Min said that the idea of taking history into account is based on the fact that two consecutive backups are very similar; the data contained in the previous backup is useful for the following backup. Min then described two optimization approaches to their new algorithm that can reduce the negative impact of out-of-order containers. Finally, the results shown demonstrated the effectiveness of their approach in terms of performance versus various commonly used approaches.

The first question was how the merge operation would be handled by their garbage collection algorithm. The answer Min gave was that they don't need to merge with their scheme.

Hardware and Low-level Techniques

Summarized by Jons-Tobias Wamhoff (jons@inf.tu-dresden.de)

The TURBO Diaries: Application-controlled Frequency Scaling Explained

Jons-Tobias Wamhoff, Stephan Diestelhorst, and Christof Fetzer, Technische Universität Dresden; Patrick Marlier and Pascal Felber, Université de Neuchâtel; Dave Dice, Oracle Labs

Jons-Tobias Wamhoff proposed that multithreaded applications should gain control over the frequency of the underlying processor cores such that they can expose their possibly asymmetric properties and improve performance. Traditionally, dynamic voltage and frequency scaling (DVFS) is used to save energy by reducing the voltage and frequency if there is only low load on the system and to boost a subset of the cores to speed up sequential bottlenecks or peak loads. Instead of relying on the transparent solution by the operating system and processor, he introduced a user-space library that allows programmatical control of DVFS.

In his talk, Jons-Tobias first gave an overview of DVFS implementations on current AMD and Intel x86 multicores and a

study of their properties regarding frequency transition latencies and power implications. The study shows when frequency scaling improves the efficiency and uses a benchmark that continuously tries to execute critical sections on all cores. The results indicate that blocking locks using the futex syscall are preferred over spinning locks if the critical section has a length of at least 1.5M cycles (500µs) to outweigh the overhead of halting the cores and boosting the frequency. With manual DVFS control, all cores can remain active at a low voltage while one core can boost its frequency. This allows reducing the break-even point of DVFS to boosted sections with at least 200k cycles (50µs). After presenting the DVFS cost, an overview of the TURBO library followed as well as a teaser for the use cases in the paper that apply the library to real-world applications.

After the talk, someone asked how frequency scaling is limited by accesses to the last level cache. Jons-Tobias replied that the L3 cache is not part of the core's frequency domain and can limit the performance when the instructions per cycle depend on the core frequency.

Implementing a Leading Loads Performance Predictor on Commodity Processors

Bo Su, National University of Defense Technology; Joseph L. Greathouse, Junli Gu, and Michael Boyer, AMD Research; Li Shen and Zhiying Wang, National University of Defense Technology

Joseph L. Greathouse started his presentation by asking how fast applications will run at different CPU frequencies. He highlighted that applications are affected by DRAM accesses, which hinder the scaling of performance to higher frequencies. Therefore, a good estimate of the memory time is required to be able to reason about the speed of the remaining CPU time. Many estimates are based on the last level cache misses, but those are not a good indicator because memory accesses can be processed in parallel. In such situations, the CPU time overlaps with the memory accesses and can still scale with the frequency.

The proposed solution focuses on leading loads, which are the first in a series of parallel cache misses to leave the CPU core. The remainder of the talk explained how leading loads can be estimated using existing performance counters with an average error of 2.7%. AMD processors maintain a miss address buffer, a list of L2 cache misses that will be served from the L3 cache or memory in the order they were requested. The event of a new entry in the first provision of the buffer demarks a new parallel memory access period. A hardware performance counter makes the event available. Together with the timestamp counter, this can be used to estimate the portion of the execution time that is spent waiting for data that is not available within the frequency domain (core, L1 & L2 cache).

Afterwards, someone noted that (1) the error is only slightly affected by the frequency, (2) no matching performance counter on Intel processors is known, and (3) the impact of the memory throughput becomes visible if the memory runs out of bandwidth and the latency increases.

HaPPy: Hyperthread-aware Power Profiling Dynamically

Yan Zhai, University of Wisconsin; Xiao Zhang and Stephane Eranian, Google Inc.; Lingjia Tang and Jason Mars, University of Michigan

Yan Zhai addressed the power accounting on servers at individual job granularity with the goal of allowing billing based on power and power capping. The focus of power accounting is on the processor because it is responsible for the biggest part of the power draw. Unfortunately, the simple approach of estimating the power draw linearly to the CPU usage does not work for hyperthreading systems because the processor cores are a shared resource.

The talk introduced a hyperthreads-aware power profiler, which first maps the socket power to the processor cores and then from the core's power to the hyperthreads. The solution is based on finding a factor that gives a ratio of the core's power to the active hyperthreads. This is done by weighting the cycles: The cycles for each hyperthread are captured and used to map the core's power to the hyperthreads. The approach allows reducing the prediction error to 7.5%.

After the talk, Yan Zhai clarified that the approach also works for power cores in large SMT systems and that finding the factor is based on samples that allow an adaption when the load or the application characteristics change.

Scalable Read-mostly Synchronization Using Passive Reader-Writer Locks

Ran Liu, Fudan University and Shanghai Jiao Tong University; Heng Zhang and Haibo Chen, Shanghai Jiao Tong University

Ran Liu started the talk with an overview of the synchronization evolutions to highlight that the mechanisms trade semantic guarantees for performance. By allowing readers and a writer to proceed in parallel, RCU removes the reader side memory barrier. Prior research reveals that the efficiency of reader-dominant synchronization would improve significantly if no barrier was required on the reader side. However, RCU adds considerable constraints to programming due to its weaker semantic compared to reader-writer locks.

While active locks keep the state always consistent using adequate barriers, passive reader-writer locks remove the barriers on the reader side by making the state only consistent if the writer becomes active. However, the writer can only wait for the readers to report their state but it cannot directly check it. The period a writer has to wait is bounded by enforcing the readers to report using IPI. The algorithm only works if TSO is guaranteed because it implicitly enforces that readers see the latest state from the writer. Ran Liu reported that passive reader-writer locks can be implemented and applied to the address space management in Linux with trivial effort. The evaluation showed scalable results similar to RCU while maintaining the reader-writer locks semantic.

There was no time for questions at the end.

Large Pages May Be Harmful on NUMA Systems

Fabien Gaud, Simon Fraser University; Baptiste Lepers, CNRS; Jeremie Decouchant, Grenoble University; Justin Funston and Alexandra Fedorova, Simon Fraser University; Vivien Quéma, Grenoble INP

Baptiste Lepers' talk was about the efficiency of large pages on NUMA systems. Since a TLB miss is expensive (43 cycles), applications that need large amounts of memory typically increase the page size from 4 KB to 2 MB to reduce the address translation overhead and have fewer TLB misses. Unfortunately, large pages may lead to a bad placement of memory on a NUMA system and hurt the performance by up to 43%. Existing memory management algorithms are not able to solve the performance decrease. The talk identifies two effects that explain bad performance: (1) hot pages, i.e., a single page that concentrates most of the memory accesses and creates contention (such a page is far more likely with 2 MB pages than with 4 KB pages); and (2) "page level false sharing," i.e., when different threads allocate distinct data that unfortunately end up being allocated on the same page—this is bad for locality if the two threads are on different nodes. These two effects can lead to a bad locality and high contention on the interconnect.

The performance bottlenecks are addressed by (1) splitting hot pages, (2) improving the locality by migrating pages to the core that accesses it most frequently, and (3) enabling 2 M pages only if the TLB miss rate is very high. The evaluation showed that the approach can lead to a performance improvement of up to 50% while introducing only 3% overhead.

Someone asked if this works for datacenter-scale applications. Lepers answered by stating the evaluation included benchmarks that use up to 20 GB memory but showed sometimes mixed results.

Efficient Tracing of Cold Code via Bias-Free Sampling

Baris Kasikci, École Polytechnique Fédérale de Lausanne (EPFL); Thomas Ball, Microsoft; George Candea, École Polytechnique Fédérale de Lausanne (EPFL); John Erickson and Madanlal Musuvathi, Microsoft

Baris Kasikci said his team's goal is to efficiently sample cold code because such code is not known a priori and is typically not well tested. Existing code instrumentation techniques are inefficient or do not scale: static instrumentation has high overheads and existing dynamic instrumentation frameworks do not work well for multithreaded applications, because they need to stall all program threads before instrumenting the program.

The proposed solution is based on leveraging breakpoints. The code is assigned one breakpoint per basic block that can be removed when the block is sampled, incurring no subsequent overhead. Insertion and deletion of a breakpoint are atomic in modern hardware and hence do not require synchronization. This way, there is no need for a separate program build (easier maintenance), and threads are better supported. The remaining challenge is the effective and efficient handling of the high volume of breakpoints that fire.

The bias-free sampling approach samples code independent of the execution frequency of its individual instructions, and it takes only a specific number of samples from all basic blocks. This way, tasks such as measuring code coverage or periodically sampling instructions can be performed with an overhead of only 1–6%.

During the discussion, Kasikci clarified that it is necessary to stall the threads when inserting or removing multi-shot breakpoints.

Distributed Systems

Summarized by Rik Farrow (rik@usenix.org)

Gestalt: Fast, Unified Fault Localization for Networked Systems

Radhika Niranjan Mysore, Google; Ratul Mahajan, Microsoft Research; Amin Vahdat, Google; George Varghese, Microsoft Research

Radhika Mysore worked on Gestalt when she was a grad student at UCSD. They used Lync, an enterprise communication system within MS that already has a monitoring infrastructure as one data source. But Lync needed an automated fault localization system, because manual localization took hours or days. They started with three existing tools: Score, Pinpoint, and Sherlock, but the diagnostic ranks of Score were terrible, better for Pinpoint, while Sherlock had a good diagnostic rank but took too long to complete. They also tried the same algorithms on an Exchange installation and found that Score was both extremely fast and very accurate. Sherlock was as accurate but still very slow.

Their first contribution was to establish a framework to explain why different algorithms behave differently for different systems, and they built Gestalt based on this framework. For comparing different algorithms, they found that each had a model of system operation, a state space explorer to generate root-cause hypotheses, and finally a scoring function for choosing the most likely causes. Radhika then explained models as ways of encoding systems organization: for example, a deterministic model that is a graph where all edges are equally likely, and a probabilistic model where probabilities are assigned to each edge. Given a model, the state space explorer traverses the model in an attempt to determine which edges may have lead to a failure. Then the scoring function chooses the most likely root-cause.

Radhika focused on one aspect that compounds successful localization: observation noise, or information that falsely reports success or failure of a transaction. This aspect helps to explain why certain algorithms performed poorly. When greedy set cover is used as a space explorer, it performs well when faced with noise, but is slow. Radhika then described how greedy set cover failed when there is noise with an example. Gestalt works better by including a noise factor—and by expecting the real culprit will explain noise—time observations, and fewer observations, and that is how Gestalt achieves better recall. Gestalt performed with high accuracy and low diagnostic time in their testing with data from real systems.

Steve Neil (Comcast) asked, if noise is critical, whether that is something that they determined by looking at all the data they

had. Radhika replied that they looked at a history of failures, looked at these observations, compared these observations with actual data, and came up with the actual value of noise. In addition, she chose noise for the presentation, when there were actually five issues that can confuse automatic fault detection. Neil asked whether this is automated. Radhika responded yes.

Insight: In-situ Online Service Failure Path Inference in Production Computing Infrastructures

Hiep Nguyen, Daniel J. Dean, Kamal Kc, and Xiaohui Gu, North Carolina State University

Hiep Nguyen explained that they examined an online service, a VCL lab with 8000 users, similar to EC2, where students can make requests for VMs. They focused their study on the reservations servers. There were many non-crashing failures in online services, as many as 1813 in one year, which often go unnoticed, like HTTP server threads dying. On top of this, replicating these failures offline is difficult because they are lacking the correct environment, don't want to use record and play, or perform diagnosis directly on a production server. But production environments provide lots of clues—inputs, configuration, logs, and system call traces—that they can use to limit search scope. Finally, they can use dynamic VM cloning to create shadow components so that they can diagnose failure on non-production systems.

Analysis is still difficult because they are using a binary-based approach, want to have low overhead (no intrusive recording), and are analyzing both compiled and interpreted programs. Their solution is to use guided binary execution exploration, which leverages the production environment data and runtime outputs as guidance to search the potential failure paths. They allow the dynamically created clone to receive input data and perform reads, but no writes, to prevent side effects on the production system. The guided binary execution exploration combines both input and console logs as constraints in the search for the cause of a fault. They also include system call records to guide the search. Their existing implementation currently supports Perl and C/C++ programs, but with a modified Perl interpreter, and uses the Pin tool for C/C++ programs. Combining all three (input, console log, and system calls) provides the best method to uncover the root cause of faults.

There were no questions.

Automating the Choice of Consistency Levels in Replicated Systems

Cheng Li, Max Planck Institute for Software Systems (MPI-SWS); Joao Leitão, NOVA University of Lisbon/CITI/NOVA-LINCS; Allen Clement, Max Planck Institute for Software Systems (MPI-SWS); Nuno Preguiça and Rodrigo Rodrigues, NOVA University of Lisbon/CITI/NOVA-LINCS; Viktor Vafeiadis, Max Planck Institute for Software Systems (MPI-SWS)

Cheng Li began by saying that developers often choose to use replication to speed up performance, but then need to decide when strong consistency, which weakens performance, is required. Three years ago, they wrote RedBlue consistency (OSDI '12), which builds replicated systems that are fast and correct. Blue means local and fast but with weak consistency, and red operations are globally slow but strongly consistent. To choose between red and blue, you choose red for operations that are not commutative and may break invariants, or you can use the faster blue. You want to maximize the blue state by encoding the side effects. Cheng Li used the example of making deposits and receiving interest in parallel. They created a tool, called Sieve, which classifies side effects into fast/weak and strong/slow operations.

They examined several example applications and noticed that most are divided into two tiers, the application servers and the database. They decided to use commutative replicated data types (CRDT). They transformed each database statement into one or more database transactions. Programmers only need to annotate the schema with a CRDT annotation to have encoded side effects into shadow operations. Cheng Li next explained how to classify operations accurately and efficiently. Sieve statically defines the weakest precondition for the corresponding shadow operation to be invariant preserving. At run time, Sieve classifies shadow operations by evaluating the corresponding weakest precondition. By using path analysis, they can determine which paths might lead to invariants by creating templates.

The programmer's notations guide the path analysis. When evaluated, Sieve using programmer annotations was almost as accurate as manually choosing weak and strong consistency issues, and incurred a very small hit to performance.

Someone from Google asked how much memory their technique added, and Cheng Li answered that they hadn't checked that. Someone else asked about selecting which types of CRDT they should use. Cheng Li answered that they have a table about how to choose CRDT types, as that research had already been done. There was a third questioner who was cut off by the session chair.

Sirius: Distributing and Coordinating Application Reference Data

Michael Bevilacqua-Linn, Maulan Byron, Peter Cline, Jon Moore, and Steve Muir, Comcast Cable

Jon Moore began by explaining that reference data means a read-only relationship with the data, and also that the rate of update to the data is not very high. For Comcast, reference data means TV and movie metadata, and, with main memory capacity growing, the authors hoped to be able to fit all the reference data into memory. But there is an impedance issue, because the application wants the data and the data is stored in a database. Object-relational mappers can be used to perform the conversion, and then application developers are dealing with data structures, algorithms, unit tests, and profilers.

They use the system of reference to publish updates to the version stored in RAM, which does not have to be that fresh. They created Sirius to do this, with just two operations, put and delete, using Paxos for accuracy and a transaction log for persistence. The application, rather than the database, handles these updates. On the read path, applications read directly from

the data structures in RAM, meaning that they have eventual consistency. They have a set of ingest servers, and client servers just pull updates from the ingest servers. Since the data includes a version, they use compaction to compress past transactions to the most recent value for each key. They use Scala and work from a paper (Paxos made moderately complex) to implement this; Jon displayed the code they used.

Sirius is currently being used at Comcast and has been running in production for almost two years. Since they want their programmers focused on the user experience, not the plumbing, this has been very important for them. The library handles persistence and replay. Sirius is available as open source at comcast.github.io/sirius.

Fred Douglis (EMC) asked about related work and Jon answered that the paper does include a long related-work section. As good engineers, they wanted to build on the shoulders of giants. There is a lot of related work, but most is in external processes. They did look around, but a key difference is not just holding data in memory, but rather the convenience to developers. Dave Presotto (Google) wondered why they ended up with Paxos, given their two-layer structure. Jon replied that their work predates Raft [the next paper], and they were looking at lots of work on distributed databases. Paxos was also easy to reason about. Someone pointed out that given their constraints, it was easy to see how they came up with this design. But with a higher update rate, this wouldn't be applicable. Jon replied that he absolutely agreed with the questioner.

In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout, Stanford University

Awarded Best Paper!

Diego Ongaro presented Raft as a replacement for Paxos. Consensus requires an agreement about shared state, and while Paxos is synonymous with consensus, it is also hard to understand and hard to implement. When they tested CS students, most tested better using Raft and would prefer to use it.

Raft is broken into three parts: leader election, log replication, and safety. The leader takes commands from clients and appends them to its log, then the leader replicates its logs to other servers. The leader sends out RPCs with updates and gets back replies from clients. The leader gets elected when the previous leader times out. Each server uses a randomized timeout to prevent split votes on leader elections.

The leader's job is to send out logs, and clients maintain two indices: match and next indices. The next index is where the next update goes, and match index is the latest update. The leader doesn't mark an update committed until a majority has responded to a log replication update. If a client server has old data, it accepts updates to overwrite that old data from the current leader. When a candidate server starts an election, but has an out-of-date log, no other servers will vote for it. So safety means the server with the latest log will always win the election.

Consensus is widely regarded as difficult, and Raft is easier to teach in classrooms, with dozens of implementations available on their Web site or at raftconsensus.github.io.

Fred Douglis said that this was a great talk and should have won best presentation, too. Diego pointed out that they could watch the student study on YouTube. Fred then asked if you could wind up with more divisions and split votes. Diego replied that you might need to scale the timeouts to be wider, to prevent split votes from occurring. Someone noted that for reverting logs, they must keep a lot of version information, and wondered whether this created a lot of overhead compared to Paxos. Diego responded that it depends on which Paxos variant you use, but the overhead is comparable. Someone else asked about log compaction, and Diego said that Raft uses a snapshotting approach; it doesn't snapshot the tail, just committed prefixes. Garth Gibson (CMU) wondered why they hadn't tried using Emulab for large-scale testing. Diego replied that the motivation for his work was RAMCloud, which is why he wasn't focused on wide area issues. In the worse case, they may have to change the leader algorithm. Garth Gibson responded that in the worse case, timeouts mean not making progress. Diego replied that Paxos takes 10 message types to do the same thing. A questioner wondered whether they had formalized the algorithm to prove its correctness. Diego said he had, and there was more work ongoing.

Networking

Summarized by Lalith Suresh (lsuresh@inet.tu-berlin.de)

GASPP: A GPU-Accelerated Stateful Packet Processing Framework

Giorgos Vasiliadis and Lazaros Koromilas, FORTH-ICS; Michalis Polychronakis, Columbia University; Sotiris Ioannidis, FORTH-ICS

Giorgos Vasiliadis presented GASPP, a framework for leveraging GPUs to perform network traffic processing. The premise of the work is that network packet processing is both computationally and memory intensive, with enough room for data parallelism. However, this presents many challenges, such as the fact that the nature of traffic processing presents poor temporal locality since each packet is mostly processed only once.

GASPP presents a modular and flexible approach to expressing a broad range of packet-processing operations to be executed on the GPU. It presents a purely-GPU-based technique for flow state management and TCP stream reconstruction. Since real traffic is very dynamic with different rates and varying packet sizes within and across flows, it becomes a challenge to ensure that GPU threads are sufficiently load balanced and occupied. Thus, GASPP also implements an efficient packet-scheduling mechanism to keep GPU occupancy high. In the evaluation, the authors present the tradeoff between latency and throughput, which is inherent to the design of GASPP.

Steve Muir (Comcast) asked whether the authors performed any comparisons versus Packet Shader. Giorgos answered that they haven't done so yet, but expect GASPP to outperform

Packet Shader. Someone from IBM Research asked how Intel's DPDK compares to GASPP. Another author answered that the main difference is that Intel DPDK uses polling and keeps the CPU cores utilized (potentially up to 100%) with the benefit of low-latency packet processing, whereas GASPP offloads packet processing to the GPU. Garth Gibson (CMU) asked how fast the CPU implementation used as a baseline was, and whether any low-level hardware features were used. Giorgos responded that only a single CPU core was used for the CPU-based baseline, but the point they wanted to make was to use both the CPU and GPU together to perform packet processing.

Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks

Dan Levin, Technische Universität Berlin; Marco Canini, Université catholique de Louvain; Stefan Schmid, Technische Universität Berlin and Telekom Innovation Labs; Fabian Schaffert and Anja Feldmann, Technische Universität Berlin

Dan Levin presented Panopticon, an architecture to allow operators to reap the benefits of software-defined networking (SDN) without having to replace all of their legacy network switches with SDN-enabled switches. He argued that while SDN presents operators with a plethora of benefits, performing a full fork-lift upgrade to SDN is impractical for most network operators.

Panopticon thus presents operators with a solution that allows them to incrementally upgrade a network to a partial-SDN network, and then treat this as a logical SDN. The gist of the idea is to replace some legacy switches with SDN switches and then have all the traffic in the network cross at least one SDN switch using VLANs, which then presents a vantage point for controlling the network as an SDN. The Panopticon planning tool takes as input the network topology, estimates of how much traffic is to flow through the network, and performance constraints (such as bandwidth requirements). It applies these inputs against a planning strategy, and then provides an output hybrid SDN deployment that satisfies the necessary constraints. The authors evaluated the work through simulations, emulations, and a real testbed. Their simulations run against the topology of a large enterprise network demonstrate that with upgrading as few as 10% of distribution switches to SDN-capable switches, most of the considered enterprise network can be operated as a single logical SDN.

Steve Muir (Comcast) asked how one positions Panopticon with respect to the trend towards the use of SDN overlays and soft-switch technologies such as OVS and VXLAN. Dan answered that if and when a network can be managed as an SDN by deploying soft switches on hypervisors at servers, then it should be done. But he repeated the point that there are many legacy networks where that cannot be done, as their survey of enterprise networks has shown, and what they are stressing in their work is how to reason about the network during the transition phase to an SDN. Nick Feamster (Georgia Tech) asked how related is the underlying physical network to the logical network when attempting to assert different guarantees. Dan answered that that is indeed a challenge and it is difficult to make strong guarantees when presented with a logical SDN as in Panopticon, which is why they say they can reap the benefits of a nearly full SDN as opposed to a full SDN.

Pythia: Diagnosing Performance Problems in Wide Area Providers

Partha Kanuparthy, Yahoo Labs; Constantine Dovrolis, Georgia Institute of Technology

Partha Kanuparthy presented Pythia, a system for automatic and real-time diagnosis of performance problems in wide area providers, which. Wide area providers are a set of sites that are deployed in different geographic regions connected by wide area links (such as ISPs or content providers). In practice, they are connected by wide area paths with transient providers, which are essentially black boxes. In these scenarios, network upgrades or changes to the traffic matrix can introduce performance problems.

Wide area providers use monitoring infrastructure that essentially runs measurements (such as ping). This infrastructure can provide a time series of end-to-end measurements of delays, packet reorderings, and network paths being used. Pythia, leverages such infrastructure for near real-time network diagnosis, problem detection, and localization by having lightweight agents run at the different monitors in the network. At the heart of Pythia is a pathology-specification language, which would allow operators to add and remove definitions of what constitutes a performance problem, allowing incremental diagnosis deployment. Each pathology is expressed as a mapping of an observation to a logical expression constituting a set of symptoms. Using this specification, Pythia generates diagnosis code as a forest of decision trees. The system then matches the recorded observations from the monitoring infrastructure (such as delays, losses, reorderings) against the decision trees in order to diagnose any detected problems, wherein a problem is defined as a significant deviation from a baseline. An interesting discussion was also presented on how bugs with the monitors themselves complicate the diagnosis of short-lived problems, since the measurements themselves are potentially contaminated.

There were no questions after the talk.

BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks

Srikanth Sundaresan, Sam Burnett, and Nick Feamster, Georgia Institute of Technology; Walter de Donato, University of Naples Federico II

Sam Burnett presented their experience deploying BISmark, a world-wide measurement infrastructure comprising modified home routers. The objective of this infrastructure was to study questions regarding whether or not ISPs are performing as advertised, how home network usage varies across different parts of the world, and how network troubleshooting can be improved. This presents many challenges, since studying home networks is difficult because of network address translation, the fact that these networks are largely unmanaged and unmoni-

tored, and, lastly, the need to involve actual home users in order to deploy such an infrastructure.

To reliably and consistently measure how fast an ISP is or how a wireless access point affects a user's performance, the home router was a natural candidate to be a vantage point. This allowed the authors to study what traffic was coming from the home network, what devices were causing this traffic (mobile phones, home entertainment devices, laptops?), and so understand where the bottlenecks were. BISmark home routers were deployed in more than 30 countries. Sam then discussed the practical challenges involved in making such a project work, since these home routers are on the critical path of a user's network. This challenge extends to issues regarding automatic updates to the software on the routers and ensuring that the routers are running even when the research team cannot access the router. Although the advantage is that the home router is an ideal vantage point for the kind of problems the authors set out to study, there are disadvantages as well. Users naturally have privacy concerns. Furthermore, establishing trust also proved to be difficult. Sam lastly also discussed interesting statistics regarding the deployment effort, including attrition rates, the time it took for each user to turn on their router after receiving it, and so forth.

Steve Muir (Comcast) asked the degree to which path measurements would have sufficed to study the kind of problems the BISmark project was targeting. Sam answered that that there are classes of problems which you cannot see using path measurements. Someone asked whether there was any relationship between the discussed trust issues and human behaviors that can be inferred from studying the usage patterns. Sam acknowledged that as an issue.

Programmatic Orchestration of WiFi Networks

Julius Schulz-Zander, Lalith Suresh, Nadi Sarrar, and Anja Feldmann, Technische Universität Berlin; Thomas Hühn, DAI-Labor and Technische Universität Berlin; Ruben Merz, Swisscom

Julius Schulz-Zander discussed Odin, a software-defined networking (SDN) framework for WiFi. The motivation for the work is that most of the benefits of SDN have been geared towards wired networks and have not benefitted WiFi as much, whereas WiFi networks are becoming increasingly more complex to manage. In order to design an SDN for WiFi and to programmatically manage WiFi networks, new abstractions need to be designed.

Core to how Odin functions is the light-virtual access point (LVAP) abstraction. An LVAP is a per-client virtual access point, which is spawned on physical access points. Every client is assigned an LVAP when it attempts to connect to the network. LVAPs can be migrated quickly between physical access points such that clients do not have to reassociate to the network and do not notice any breakage at the link layer. Using this mechanism, an Odin-managed network can control clients' attachment points to the network. A logically centralized controller manages LVAPs and exposes the programming API with which network applications can orchestrate the underlying network. Using LVAPs, a network can be divided into slices, where each slice is defined as a set of physical APs, network names, and network applications that operate upon it. An application can control only those clients that are connected to the network names of the slice that it belongs to, and thus, applications cannot control LVAPs from another slice. Using these building blocks, the authors built six typical enterprise services on top of Odin.

Nick Feamster (Georgia Tech) asked how Odin compares to commercial offerings such as those from Meru, Cisco, and Meraki. Julius responded that Odin exposes more low-level hooks to write applications such as mobility managers than the commercial offerings do.

HACK: Hierarchical ACKs for Efficient Wireless Medium Utilization

Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp, University College London

Awarded Best Paper!

Lynne Salameh discussed how they overcame a limitation with WiFi's medium acquisition overhead and its ramification on TCP's end-to-end throughput. Since every two data packets in TCP require one TCP ACK, this overhead restricts performance as data rates increase, even in the presence of 802.11 frame aggregation and link-layer block ACKs.

The proposed cross-layer solution is named TCP/HACK (Hierarchical ACKnowledgment), which eliminates medium accesses for TCP-ACKs in unidirectional TCP flows by encapsulating TCP-ACKs within WiFi ACK frames. An important design consideration here is that devices using TCP/HACK should coexist with stock 802.11 devices. Furthermore, block ACKs have a hard deadline in that they must be sent within the short interframe spacing duration (SIFS). Given that TCP ACKs may not be ready in time, the block ACKs cannot be delayed since other senders will then acquire the medium. Since TCP ACKs do not have hard deadlines themselves, the TCP ACKs can be appended to the next link layer ACK. Lastly, since a client does not know whether there will be an ACK to send out soon, the access point notifies clients if the access point has any packets destined to them in its transmit queue. This allows clients to not have to guess whether there will be any ACK to be transmitted soon. An implementation of the technique was tested on a software radio platform as well as using simulations with ns-3.

Someone asked whether there are any metrics that worsen when using TCP/HACK. Lynne answered that aggregation makes TCP ACKs bursty anyway, and if you delay TCP ACKs to be encapsulated within the next link layer ACK, you will increase the round-trip time (RTT). Garth Gibson (CMU) asked about the ramifications of breaking modularity, since TCP/HACK is tied to a lower level protocol. Lynne answered that there is always a danger in breaking the layering structure. The final question was on whether there was a performance cost incurred from the

fact that 802.11 ACKs have to be sent at the basic rate of 1 mbps. Lynne responded that 802.11n ACKs are sent at higher data rates.

Best of the Rest III

Summarized by Dimitris Skourtis (skourtis@soe.ucsc.edu)

Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation

Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara, University of Waterloo

The manufacturing of digital integrated circuits (ICs) is being outsourced to multiple teams and external foundries in different locations. Such outsourcing can lead to security threats as soon as any of those foundries inserts a malicious modification to the IC.

The presenter described a threat model where an insider at an external foundry makes a malicious modification. The author then described an example of an attack, where the malicious party attaches a hardware trojan allowing the attacked to be treated as a super user. Next, the presenter gave an example of a modified circuit, where a gate and a trigger were added to allow the malicious party to control when the attack happens. The presenter mentioned that if a malicious modification happens, all IC instances will carry that. Also, whereas with software viruses a patch could be released, this is not true for hardware.

Next, the presenter described how the above problem can be solved by obfuscating the logic of the IC, that is, by hiding certain wires from the view of the third parties. In practice, the IC is split into two or more tiers. The top tier is fabricated in-house and implements the wires that have to remain hidden. The obfuscated circuit is outsourced and is fabricated on other tiers.

The presenter next described a formalization of the level of security provided by circuit obfuscation. In particular, the presenter defined the k-secure gate as one that is indistinguishable from at least k-1 other gates in the circuit. If all gates are k-secure, then the circuit provides k-security. This makes it hard for the attacker to identify a particular gate, because they would have to attack k gates as opposed to a single one.

Next, it was mentioned that it is computationally expensive to find the minimum number of wires that can be hidden while guaranteeing a k-secure circuit. There is a tradeoff between the number of hidden wires and the amount of security. A greedy algorithm was then presented to manage that tradeoff, and was shown to be more effective than randomized selection.

Bill Walker (Fujitsu) asked about detecting malicious modifications. The presenter noted that attackers can disable the attack during testing so that it remains undetected and only be activated afterwards.

Control Flow Integrity for COTS Binaries

Mingwei Zhang and R. Sekar, Stony Brook University

Mingwei first introduced control-flow integrity (CFI), a low-level security property that raises a strong defense against many attacks such as return-oriented programming. Previous work requires compiler support or symbol information to apply CFI. Instead, Mingwei mentioned that their work applies to stripped/COTS binaries, with comparable performance to that of existing implementations.

Mingwei talked about the key challenges: disassembling and instrumenting the binary without breaking the low-level code, and applying their technique to libraries. With respect to disassembling, Mingwei mentioned they are using a mixture of linear and recursive disassembling to mark gaps between pieces of code.

To maintain the correctness of the original executable as well as all the dynamically loaded libraries, they maintain a global translation table (GTT), which gets updated as modules are loaded. Update to GTT is performed by a modified dynamic linker and, in particular, the loader (ld.so).

To evaluate the correctness of their implementation, they successfully applied their method to binaries of over 300 MB (240 MB being libraries). Moreover, Mingwei presented benchmarks (SPEC) to evaluate the runtime overhead (4.29% for C programs), as well as the space (139%) and memory (2.2%) overhead.

Zhiqiang Lin (UT Dallas) asked whether they had encountered any false positives. Mingwei answered that so far they had not, but if there were any, they would discover them.

HotCloud '14: 6th USENIX Workshop on Hot Topics in Cloud Computing

June 17–18, 2014, Philadelphia, PA

Summarized by Li Chen, Mohammed Hassan, Robert Jellinek, Cheng Li, and Hiep Nguyen

Note: The first two sessions of HotCloud '14 were joint sessions with HotStorage '14, and the summary of the keynote can be found in the HotStorage '14 summary on page 91.

Systems and Architecture

Summarized by Li Chen (lchenad@ust.hk)

Academic Cloud Computing Research: Five Pitfalls and Five Opportunities

Adam Barker, Blesson Varghese, Jonathan Stuart Ward, and Ian Sommerville, University of St Andrews

Adam Barker described five pitfalls and five opportunities in academic cloud research in this talk. He argued that academia is pursuing the wrong class of problems and should instead conduct research with higher risk. The core of the problem is the scale of low-level infrastructure that academia has access to is limited, and therefore the research conducted is of lesser value to the cloud computing community.

The first pitfall lies in infrastructure at scale. With more than hundreds of thousands of servers in big cloud computing services, academics can only recreate a small subset of the network,

and cannot replicate the scale. Therefore research may have less value if the researcher does not have access to or partnership with large clouds. The second pitfall is with abstraction, a key feature of cloud computing. Academics see no value in "black box" abstractions, and often need to reimplement the low-level infrastructural components for comparison or prototypes, without support from cloud providers. The third pitfall is with unreproducible results from network simulators, simulations with real-world trace data, and custom evaluation setups. Reproducing the results in papers using these evaluation schemes is nearly impossible. The fourth pitfall is about rebranding cluster/grid computing research as cloud computing. Research on lower levels cannot be tested by academic peers, and research on higher levels may actually have a longer-term effect. The last pitfall is about industrial relations. Current research programs provided by the industry do not address the problem of not being able to access low-level infrastructure.

Researchers should exploit the opportunities in user driven problems. The properties of cloud computing can help solve a number of problems in other domains such as scientific problems. Minimizing cloud resource usage given user derived requirements is also an interesting area. Programming models other than MapReduce should also be investigated, because MapReduce does not fit for all computation tasks. Debugging large-scale applications is very difficult due to the inherent complexity, scale, and high level of abstraction. This area does not receive enough attention from academia. The fourth opportunity lies in Platform-as-a-Service environments. Building environments on multiple cloud infrastructure and providing a high-level interface for users pose interesting challenges. Elasticity is the last opportunity that Adam pointed out. Dynamic provisioning is what differentiates cloud computing from cluster/grid computing.

The first questioner pointed out that there is a huge concern with intellectual property (IP) issues and legal issues. Adam replied that the industry does not feel confident about sharing their facilities because of IP issues. It's really a chicken-and-egg problem: Academia does not have the necessary IP to guarantee deliverables, and industry does not want to share for the same reason. A unified model that resolves these issues in the industrial-academic relationship may be necessary. Another attendee stated that datacenters are drastically different when scaled up, so raising the level of abstraction actually hinders the improvement that can be made by academia. Adam replied that cloud computing and datacenter networking are not the same and require different levels of abstraction. It is important to question all layers in improving datacenter performance, and for cloud computing, a higher abstraction level in fact provides academics with more freedom.

Towards a Leaner Geo-distributed Cloud Infrastructure

Iyswarya Narayanan, The Pennsylvania State University; Aman Kansal, Microsoft Corporation; Anand Sivasubramaniam and Bhuvan Urgaonkar, The Pennsylvania State University; Sriram Govindan, Microsoft Corporation

Aman Kansal started by reviewing the factors affecting the capacity implications of geo-distribution. Latency is the most compelling argument for geo-distribution, as users all over the world would like to be serviced by the nearest datacenters. Another advantage of geo-distribution is failure recovery in case of disasters, but the availability gains come at the cost of excess capacity. Geo-distribution can also exploit regional differences in energy cost.

Aman emphasized the problem of excess capacity, and continued to examine what is the least capacity required. He formulated a linear programming problem with the goal of minimizing the sum of capacities in geo-distributed datacenters. The constraints include latency and capacity to service demand, before and after failures. Aman also identified the trade-off between latency requirement and capacity requirement—tighter latency constraints lead to higher capacity requirements. He showed an interesting result that the excess capacity required by latency and availability jointly is similar to that of latency alone. He also pointed out that routing to the nearest datacenter is not always efficient, especially after a disaster.

Aman went on to describe the open challenges in two aspects: infrastructure and software design. For infrastructure, the previous optimization problem needs to be further examined to consider more factors. Another issue is the fine-grained control of latency and availability for different applications. Lastly, spatial-temporal variations of failures and demands should be exploited to achieve better capacity provisioning.

For software design, Aman emphasized request routing to ensure that the demand is routed to the correct datacenter for efficient capacity provisioning. Placing copies of states and data for efficient user access in geo-distributed infrastructure is another interesting challenge. It is also important for the software to automatically scale the computation with the demand. In the end, Aman came to virtualization, and mentioned that the applications in the cloud should be able to exploit the flexibility of geo-distributed virtualized datacenters.

The first questioner asked: If distribution of clients with different latency classes will affect their formulation, what would be the impact? Aman replied that adding more latency classes can be addressed by small modifications to the formulation, and that they plan on studying the impact in future work. The second questioner asked how their geo-distributed model is affected when the number of servers increases or decreases. Aman answered that depends on the capacity of the infrastructure and how the demand grows over time. In practical cases, land is more important, so the number of servers will not vary significantly. A third person asked about data consistency in distributed data-

centers. Aman replied that their model makes the assumption that the consistency is handled already.

A Way Forward: Enabling Operating System Innovation in the Cloud

Dan Schatzberg, James Cadden, Orran Krieger, and Jonathan Appavoo, Boston University

Dan Schatzberg pointed out that the OSes used in the cloud are often general purpose and not optimized for the cloud. A general purpose OS supports multiple users and applications concurrently, while entire virtual machines in the cloud are often dedicated to a single application. Dan argued for a reduced role for the OS in cloud computing and presented the MultiLibOS model, which enables each application to have its own customized OS.

Dan started by reviewing the unnecessary or relaxed requirements of general purpose OSes in the cloud: support for multiple concurrent users, resource balancing and arbitration, and identical OS (symmetric structure) for all the nodes.

Dan described the MultiLibOS model as combining a general purpose OS with a specialized OS. With this model, applications have flexibility in choosing their own OS's functionalities, from a full legacy OS to a lean customized library. In this way, providing an application with a feature is simple and intuitive, as one need not to go through the labyrinth of a legacy OS. Dan also noted that MultiLibOS makes application and hardware specialization easy, and allows for elasticity and full backward-compatibility.

Dan discussed a few research questions for MultiLibOS. Library development has many known issues, such as configuration, compatibility, "versionitis," fragmentation, reliability, and security. Dan suggested language-level techniques to deal with these issues and noted the importance of efficiently reusing libraries when customizing for different applications; otherwise, a major advantage of MultiLibOS is lost. Lastly, the improvement of a specialized OS needs to justify the cost of development.

The first questioner asked about Dan's intuition of how this was going to work in a virtualized environment. Dan replied that depends on how isolation is implemented in physical hardware; they think it is a good match for virtualized settings. Another attendee wondered whether there's enough headroom to make this work well. Dan replied that their preliminary results show there is a gap, and their design does make improvement to decrease this gap. The final questioner wondered how this is different from RAMCloud. Dan answered that it is along the same line of research, but they focus on giving every application its own customized system.

Software Defining System Devices with the "Banana" Double-Split Driver Model

Dan Williams and Hani Jamjoom, IBM T. J. Watson Research Center; Hakim Weatherspoon, Cornell University

With a botany analogy, Dan Williams showed us that there can be a clean separation of Spike (backend driver) and Corm (hardware driver) in the virtualized cloud, and that the spike and corm do not have to be on the same physical machine.

Dan first identified the incomplete decoupling of system devices in the cloud. The virtual devices are dependent on the physical hardware, which limits the flexibility of the cloud resource management. The split driver model in Xen, while enabling flexibility to multiple access to hardware, fails to provide location independence. To design a generic, software-defined mechanism for device decoupling, he proposed the Banana Double-Split Driver model (Banana for short).

Banana splits the backend driver in Xen into two parts, Corm and Spike. Corm handles multiple accesses to the hardware, and Spike handles the guest OS. Corm and Spike are connected by wire that can be switched in local memory or network connections. Wires are controlled by the Banana controller, which is software-defined and can create on-the-fly reconfigurations.

Dan demonstrated the Banana model by providing an alternative approach to virtualize NICs in Xen, noting that Xen can currently support device-specific complete decoupling of NICs. The management and switching of wires is achieved by integrating the endpoint controller with the hypervisor. Dan mentioned that they augmented the existing Xen live migration mechanism to enable migration of wires and endpoints.

The experimental setup showed the Banana Double-Split model works, but the overhead is large. It is exciting to see that they can live migrate VMs from local cloud to Amazon EC2 without a complex network setup. Dan showed that VM migration is simplified with Banana, but the downtime is increased.

The first questioner pointed out that the guest VM does not have as much detail about the hardware driver. Dan replied that if you want a general framework/API, you will lose some flexibility. The second questioner pointed out that their design requires a taxonomy of all the types of hardware, and wondered whether they had done this work. Dan answered that they had focused on the dependency issues, and their proof-of-concept improves on NICs for now. The final question was about the design's sensitivity to different devices. Dan replied that they had designed something general for all devices. Different devices need to be treated differently, but the authors feel that their model is the way to do it.

Building a Scalable Multimedia Search Engine Using Infiniband

Qi Chen, Peking University; Yisheng Liao, Christopher Mitchell, and Jinyang Li, New York University; Zhen Xiao, Peking University

In this talk, Qi Chen delivered a key insight on how to scale multimedia search in datacenter networks: With low-latency networking, computation time is reduced by using more round-trips to perform searches in a large media collection.

Vertical partitioning is known for its potential scalability for multimedia search engines, yet the large number of indexed

features results in huge communication cost per search query. Therefore it is impractical to implement on Ethernet.

But with high performance networking technologies like Infiniband, vertical partitioning becomes viable, as the round-trip time is only a few microseconds, as opposed to hundreds of microseconds in Ethernet. In this work, they demonstrated the practicality of vertical partitioning by building a distributed image search engine, VertiCut, on Infiniband.

Qi described the two key optimizations in VertiCut. First, VertiCut performs an approximation of k-nearest-neighbor search by stopping early (after getting enough good results). This helps to reduce the number of hash tables read per query. Second, VertiCut keeps a local bitmap at each server to avoid looking up non-existent keys.

For the evaluation, Qi mainly described the comparison with traditional horizontal cutting, dispatch, and aggregate schemes. Qi showed that, with higher network cost (in terms of bytes sent per query), vertical cutting is faster than horizontal cutting on Infiniband. Qi also discussed the effects of the optimizations, concluding that the first optimization results in an 80x speed-up, and the second 25x.

The first questioner asked how the data is stored. Qi answered that their workload is stored in a single DHT table, not on servers. The next questioner noticed a similarity to a previous work on optimal aggregation of middleware and wondered whether they plan to extend their applications. Qi said that in addition to multimedia search, they will have more types of applications in the future. Finally, someone asked whether they have a formal treatment to deal with LSH randomness. Qi said that they have analysis to back up the early stops, and have other approximation methods that they are evaluating.

Mobility and Security

Summarized by Mohammed Hassan (mhassanb@masonlive.gmu.edu)

POMAC: Properly Offloading Mobile Applications to Clouds

Mohammed A. Hassan, George Mason University; Kshitiz Bhattarai, SAP Lab; Qi Wei and Songqing Chen, George Mason University

Mohammed Hassan showed how computation-intensive mobile applications can be offloaded to the cloud more efficiently. He presented a framework that proposed a transparent approach for an existing mobile application to be offloaded. In addition, the authors suggested when the computation should be offloaded and when it should be executed on the mobile device.

Hassan explained that mobile applications are getting more and more resource hungry, but mobile devices are constrained by a limited power supply and resources. Offloading computation to the cloud can mitigate the limitations of the mobile devices. But the current research either requires the applications to be modified or requires a full clone image running on the cloud to be offloaded. Hassan claims that their first contribution is to provide a transparent mechanism for the existing applications to be offloaded without modification. On the other hand, Hassan also emphasized the timing of the offloading decision, which depends on the network bandwidth and latency between the mobile device and the server, and on the server-side load as well. To make the offloading decision more efficiently, Hassan showed that a learning-based classifier would make a more accurate decision for offloading.

Chit-Kwan Lin (UpShift Lab) asked how the bandwidth and latency between the mobile device and the server is measured. Hassan replied that they are monitoring previous values and using a moving average to predict the future bandwidth and latency. He also explained that bandwidth and latency can be well predicted by monitoring the network the mobile is connected to. Michael Kozuch (Intel Labs) suggested that considering remaining battery power in making the offloading decision can help more. Phillip Gibbons (Intel Labs) asked whether the energy consumption is considered here for making the offloading decision. Hassan replied that in future work they are planning to consider the tradeoff between energy consumption and response time for making offloading decisions.

Mobile App Acceleration via Fine-Grain Offloading to the Cloud

Chit-Kwan Lin, UpShift Labs, Inc.; H. T. Kung, Harvard University

Chit-Kwan Lin proposed a novel compression technique that can boost mobile device performance by offloading. Lin included some promising findings about the performance gain of the offloaded performance.

Lin presented the importance of cloud computing for emerging resource-intensive mobile applications. While offloading can augment the computation power of the mobile devices, the bandwidth and latency between the mobile devices and cloud impacts the offloading overhead. With these circumstances, fine-grained offloading may provide more performance gain.

To offload mobile computation, it is necessary to have a replication of the application in the cloud side to execute the offloaded application there, and to synchronize the server-side replication's memory blocks. Lin showed a novel technique to minimize the synchronization data transfer overhead by compression. In short, the change in the mobile device's state is compressed and sent to the server side. The server side synchronizes by uncompressing the changes and thus updating itself. In this way, offloading can be done without object marshaling and with less overhead. At the end, the presenter demonstrated the effectiveness with a handwriting recognition application.

Someone asked how change is sent to the server. Lin responded that the changes are sent continuously. Another person asked how the server side was executing the offloaded application. Lin said that same exact application was running off the server side. Ymir Vigfusson asked about overhead if there are lots of writes and the mobile device state changes a lot. Lin replied that in

that case the compression technique might not help that much because there would be a lot of overhead.

Leveraging Virtual Machine Introspection for Hot-Hardening of Arbitrary Cloud-User Applications

Sebastian Biedermann and Stefan Katzenbeisser, Technische Universität Darmstadt; Jakub Szefer, Yale University

Sebastian Biedermann proposed an architecture to improve security settings of network applications in a cloud computing environment. Biedermann proposed a technique to locate and access memory locations of another VM for runtime analysis of applications on VM.

Biedermann introduced his presentation by citing related work, "hot-patching," which enables runtime analysis of another VM from the virtual machine introspection (VMI) by accessing the VM's memory. Hot-hardening is a similar approach that continuously and transparently improves the security-related configuration of running apps in a VM. At first the security or configuration setting (it may be a file in the memory or storage) of the target VM is identified and located. Then the VM is cloned for inspection. After the VM is cloned, the settings of the cloned VM are replaced or written with a different configuration to see its impact on applications. Finding an application's settings file in the VM's memory or storage is challenging. Biedermann showed that the setting file can be found by searching for certain settings' patterns in the VM's memory. Biedermann finally showed the framework's effectiveness with some real-world applications (e.g., MySQL and OpenSSH).

The first questioner asked how the configuration files were detected. Biedermann answered that they were using some heuristics to look for the setting's pattern. The second questioner wondered about the latency of VM cloning. Biedermann replied that it takes only few seconds to clone, which is acceptable. The last questioner wanted to know how the settings changes are injected in the cloned VM. Biedermann replied that it was done by changing the memory/page of the target VM.

Practical Confidentiality Preserving Big Data Analysis

Julian James Stephen, Savvas Savvides, Russell Seidel, and Patrick Eugster, Purdue University

Julian James Stephen proposed a framework to encrypt data for MapReduce work in the cloud. Security and data confidentiality are big concerns for cloud computing, where users have to trust third-party cloud providers with private data. The proposed framework showed that computation can be conducted in the cloud over the encrypted data while the server side is not aware of the actual content.

Stephen started his presentation by stating that the cloud has a big potential for carrying computation, but it also comes with the potential for security breaches like a data leak. But data can be encrypted with fully homomorphic encryption (FHE) so that the server side may carry the computation without knowing the actual content. But FHE is associated with high overhead, while partial homomorphic encryption (PHE) can keep the encryption overhead acceptable and is capable of performing certain operations. Stephen proposed a framework, Crypsis, that transfers Pig Latin script for MapReduce to accept encrypted data for computation. With an example, he demonstrated how a simple Pig Latin script can be transferred to work with encrypted data. Here the data is encrypted by a different encryption technique, and the original operations are transformed to operate on encrypted data by a user defined file (UDF). Stephen then compared the time to execute original and encrypted scripts and observed a three times overhead for the encrypted operations. The presentation also included some limitations: namely, the proposed framework does not support iterations; the UDF has to be defined by the user; and although the data is encrypted, the data access pattern is exposed when computation is carried in the cloud.

In the question and answer session, Phillip Gibbons (Intel Labs) asked how encrypted data is read on the client side. Stephen answered that the client side decrypts the data to find the result.

Keynote Address

Summarized by Cheng Li (chengli@mpi-sws.org)

Programming Cloud Infrastructure

Albert Greenberg, Director of Development, Microsoft Azure Networking

Albert Greenberg presented the framework they built inside Microsoft to allow developers to easily manage the large-scale cloud system. He started his talk by showing that the cloud system has grown very fast in Microsoft. In the past four years, computation and storage have doubled every six months, and a significant number of customers have signed up to use the Azure services. In addition, applications are not running in separated environments; instead, they are concurrently sharing resources within a single datacenter or across multiple datacenters.

All these trends urgently require an efficient and easy-to-use management application, which should provide an unambiguous language for architects to describe intent, codify design, and generate full details of the design, and allow different applications to consume data. To achieve this goal, Albert's team proposed NetGraph, an abstract graph model of network, to specify arbitrary network topology and state in an XML-like fashion. To be more specific, he showed a few adaptations of the graph model: physical network graph, data plane graph, control plane graph, and even the overlayed network graph.

Without the graph model, in the conventional buildout scenarios, a group of engineers played a very important role in transforming the high-level design into a deployed and configured system. PDFs and spreadsheets that described the design and are often in vendor-specific formats were exchanged among them to justify and debug the design. The obvious drawback of this approach is that it is really impossible to automate. To ease the developers' work and make the design highly dependable, they built a network graph generator and a network graph service to make the best use of the graph model. Developers could use

the reusable and extensible plugin modules in the generator to enforce the design principles. Additionally, the generator could produce both human-readable and machine-readable description files. The graph service stores the detailed design information in an in-memory graph database, and offers APIs for fetching and updating the information.

In addition to the automation, Greenberg mentioned the problems they found while managing the large-scale systems, such as unexpected states, interference, dependencies, and so on. To resolve these problems, they chose a state driven approach, where they could model dynamic network changes in a network state service (NSS). NSS knows all target states (good states) and all observed states (maybe bad), and periodically checks the difference between these two types of states to figure out and reconcile unexpected behaviors.

Following the talk, many interesting questions arose. The first was about the purpose of maintaining versioned objects. Greenberg replied that all objects in the graph database are versioned since they have to be able to roll back. The second question concerned the quiescent point before applying updates. Greenberg said that it is not practical to identify the quiescent point. Instead, they could roll back if any mistakes had been made. They also try to make the updates fast and incremental. The last question was about access control (ACL). Albert pointed out that ACL can conflict and overlap, so they designed a few automated methods to constantly check ACL (e.g., set comparison) and flag conflicts for an admin to investigate/resolve.

Diagnostics and Testing

Summarized by Hiep Nguyen (hcnguye3@ncsu.edu)

A Novel Technique for Long-Term Anomaly Detection in the Cloud

Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal, Twitter Inc.

Owen began by noting that existing work in anomaly detection does not work well when dealing with long-term anomalies such that just using the median is not good due to pronounced trend. Owen described the observation with Twitter production data that the underlying trend often becomes prominent when they look for a longer time span (i.e., more than two weeks) using time series data.

Owen then described the experience with exploring two approaches to extract the trend component of a long-term time series using STL Trend (seasonal, trend, and irregular components using Loess) and Quantile Regression. Neither of these two worked well in their experiments. He then introduced a technique called Piecewise Median to fix the limitations of these approaches. This technique computes the trend as a piecewise combination of short-term medians.

Someone asked whether this technique is used in the real production system and how the author would apply it. Owen said they tested the technique with the production data, and they are working with the team to deploy it in a real production system. Another questioner asked whether the authors consider the medians of nearby windows. Owen said it would be definitely helpful to consider those medians.

PerfCompass: Toward Runtime Performance Anomaly Fault Localization for Infrastructure-as-a-Service Clouds

Daniel J. Dean, Hiep Nguyen, Peipei Wang, and Xiaohui Gu, North Carolina State University

Daniel started with describing the common problem with multi-tenant cloud systems where the observed problem may come from external sources such as resource contention or interference or because of the software itself. If the admin can determine whether the performance anomaly is from an external fault, system administrators can simply migrate the VM to another physical node to fix the problem.

Daniel then introduced a system named PerfCompass that uses a system call trace analysis technique to identify whether the root cause of a performance anomaly is an external fault or is an internal fault. The main idea of the technique is based on the observation that the external fault will have a global effect on the application, meaning most of the threads will be affected. On the other hand, if the performance anomaly is caused by an internal fault, only a subset of the threads is affected. Finally, he described the results on testing the system with a set of real internal faults and typical external faults, showing that the system performed well with those tested faults.

John Arrasjid (VMware) asked whether the authors consider the arguments of system calls. Daniel said that it would be definitely helpful to consider that. He also mentioned that the way the system does segmentation helps in grouping system calls with similar arguments. Someone from VMware commented that the authors may want to look at the console log because it might be difficult to enable system call tracing in the real production systems. Daniel said that not all applications generate a console log, and system call tracing is lightweight.

MrLazy: Lazy Runtime Label Propagation for MapReduce

Sherif Akoush, Lucian Carata, Ripduman Sohan, and Andy Hopper, University of Cambridge

Sherif described MrLazy, a system that relies on lineage (i.e., origin information for a piece of data) to ensure that potentially sensitive data is checked against sharing policies applied to the data when it is propagated in the cloud. The motivation for the work is that existing work has various deployment challenges and runtime overhead issues. Sherif stated that checking data within a given trust domain continuously is not necessary and is the main source of the overhead. MrLazy delays the enforcement of data dissemination policies to the point where data crosses a trust boundary.

Sherif then described the results that the authors performed on a MapReduce framework. The results showed that MrLazy can significantly improve the job running time.

There were no questions.

Mechanisms and Architectures for Tail-Tolerant System Operations in Cloud

Qinghua Lu, China University of Petroleum and NICTA; Liming Zhu, Xiwei Xu, and Len Bass, NICTA; Shanshan Li, Weishan Zhang, and Ning Wang, China University of Petroleum

No presentation, paper only.

The Case for System Testing with Swift Hierarchical VM Fork

Junji Zhi, Sahil Suneja, and Eyal de Lara, University of Toronto

Junji started by stating that software testing is challenging because there are a lot of test cases that need to be executed, which may take a very long time if performed sequentially. He then gave an example of testing MySQL software. Junji then motivated his work with the observation that multiple steps during testing are shared, the test cases share the same code base, and a lot of test cases need to reuse the state of another test case; the authors concluded that if they could reuse the state of test cases, they could speed up the testing process.

Junji then described the idea of using VM fork to clone the VM that has the state of the finished test case available to use for multiple other test cases, thus allowing reuse and parallel testing. He went on to describe how this would improve the testing time in the MySQL example.

Someone asked whether the authors needed to assume that test cases are deterministic and whether they had any thoughts on applicability to non-deterministic test cases. Junji said that they need to assume test cases are deterministic. Someone else asked whether they can use OS fork instead of VM fork. Junji replied that running multiple processes might end up not working because of resource-sharing.

Economics

Summarized by Robert Jellinek (jellinek@cs.wisc.edu)

BitBill: Scalable, Robust, Verifiable Peer-to-Peer Billing for Cloud Computing

Li Chen and Kai Chen, Hong Kong University of Science and Technology

Li Chen presented work on BitBill, a system that ensures verifiable accounting of billable events in the cloud. He noted that more companies are using cloud computing, but that verifiable billing is still an issue for both providers and tenants.

In particular, providers may have trouble accounting precisely for all resource usage, which may in fact be detrimental to them since they can undercharge. Tenants, on the other hand, cannot perform an audit to verify that they are being billed correctly, because the actual physical resource consumption is behind a layer of abstraction. Tenants cannot trust providers under the current model.

Chen said that the lack of trust is currently impeding a wider adoption of cloud computing, and he presented several trust models: two existing models, and the authors' proposed model. The first model is that of unconditional trust, which is currently used in practice by commercial cloud providers. In this model, the tenant trusts the provider to accurately record tenants' resource usage and to bill accordingly. The tenants have no way to verify that they are billed accurately. The second model—the third-party trust model—uses a trusted third party to verify that resource accounting and billing are accurate. One problem with this model is that it introduces a central point of failure: the third party. Furthermore, the third party must itself have enough resources to perform accurate resource accounting, which could turn out to be a bottleneck.

Chen then introduced a third model, the authors' public trust model, where trust is distributed across all nodes in a network. The nodes maintain a single global history of billable events, which the authors implemented using a peer-to-peer (p2p) network to maintain resource accounting information across all participating nodes. Here, the only assumption is that the majority of nodes in the network are honest (i.e., will not introduce false events to the global log, or omit true ones). This is reinforced by the fact that all nodes share the same physical resource pool, and so one primitive resource, such as a CPU cycle on a single core, cannot be billed to two tenants.

The authors' implementation of this public trust model uses the Bitcoin-like solution to the Byzantine Generals Problem to ensure they have a trustworthy distributed log of billable events, even in the presence of untrustworthy individual nodes. Here, every billable event is broadcast to all nodes and is signed by the announcing party. To avoid false announcements, they use a simpler version of the proof-of-work (PoW) technique used in Bitcoin, where any announcing party must solve a computationally intensive problem to send along with the announcement. These PoW problems are NP-hard, so that they are easy for nodes to verify but hard for them to forge, ensuring that double billing announcements do not occur.

Chen then briefly explained the implementation of BitBill, which uses a Merkle tree so that every non-leaf node is labeled with the hash of the labels of its children nodes. Once a node finishes the PoW problem, it broadcasts its block to all other nodes, which then verify that block and use it to construct the next block. This yields the important property that the existence of an item in the log means that a network node has accepted it, and the blocks subsequently added to the log further affirm its validity. Ties are broken such that a given node works on the longest chain it sees, and nodes add any blocks they've missed by pulling them from future announcements they receive.

Chen noted that in their evaluation so far, BitBill appears to be much more scalable than the third-party-verifier model, and they are continuing evaluation. He then discussed deployment, resource monitoring, and security, saying that BitBill can be distributed by providers for users to install as a package or included in the user's VM, that BitBill can be used as the basis to extend existing work on verifiable resource accounting, and that due to the PoW approach, BitBill is secure as long as the majority of participating nodes are honest.

Michael Kozuch (Intel Labs) asked how often verification needs to happen, and what a standard policy would look like. Chen answered that it would depend on how the provider would want to charge the tenants, and that sampling and verification could happen at varying granularities.

A Day Late and a Dollar Short: The Case for Research on Cloud Billing Systems

Robert Jellinek, Yan Zhai, Thomas Ristenpart, and Michael Swift, University of Wisconsin-Madison

Robert Jellinek presented work on cloud billing systems, focusing on existing systems' lack of transparency, long update delays, unpredictability, and lack of APIs.

Jellinek began by noting that despite the fact that much attention has been paid to performance, reliability, and cost studies of the cloud, there has been no study of the billing systems themselves. The predominant pay-as-you-go pricing model relies upon complex, large-scale resource-accounting and billing systems that are not fully understood by cloud computing consumers.

The main question the authors considered was how one can track resource usage in real time and at fine granularity while maintaining accuracy and not hurting performance. They investigated Amazon Web Services (AWS), Google Compute Engine (GCE), and Rackspace public cloud. Jellinek noted that they were able to reverse-engineer the timestamps corresponding to various billing events, uncover several bugs in the providers' billing systems, detect systematic undercharging due to aggregation or caching, and characterize the performance of billing latency, which turned out to be substantial across all platforms.

Jellinek then described the methodology for their measurement study, which involved instrumenting providers' API calls to collect timestamps of all important instance lifetime events, largely by polling the APIs for instances' state. They would then launch an instance, execute a workload to test compute-time billing thresholds, storage, or network usage, fetch instance-local data related to the workload in question, terminate the instance, and then poll providers' various billing interfaces to check for updates. Billing latency, which they define as the time between when a resource is consumed and when the corresponding charge becomes available on a given billing interface, is recorded when a billing update is registered.

Jellinek then described various billing interfaces, including the Web-based GUI interfaces available for all three providers, and the additional CSV interfaces and Cloudwatch monitoring service available on AWS. Collecting information from the GUI interfaces required screen scraping, and none of the interfaces were particularly user-friendly. No providers offered billing APIs.

The authors found that billing updates would not necessarily occur atomically and that they occurred with high and unpredictable latency. Among other things, this made experiments difficult, since it was necessary to wait for longer than the greatest observed latency to be sure all updates had been registered. AWS, GCE, and Rackspace updated with average latencies of 6 hours 41 minutes, 22.5 hours, and 2.2 days, respectively, and with high variance. This shows that billing updates are both slow and unpredictable, which he claimed is bad for consumers who wish to optimize their deployment decisions.

Jellinek then described their experiments to measure when billing for an instance begins and ends, noting that this is ambiguous since most providers are not specific enough in their documentation or in the timestamps they provide. This means that, if a user thinks she has only run an EC2 instance for 3590 seconds, she may in fact get charged for two hours of usage, depending on how she measures an instance hour. The authors found that, despite the fact that they were able to determine what timestamps correspond to the start and end of billing for the three providers, they were not able to measure this precisely. This is due to the semantic gap between the providers' knowledge of their billing timestamps and the customers' knowledge. If the provider does not report its record of the relevant timestamps, a customer cannot know them precisely since they have to poll the provider's APIs for updated instance-state information. This is subject to jitter from variable network latency, server response time, and polling granularity. He then described a bug they found in EC2 that would yield two minutes on average of free compute time under certain conditions relating to when the instance was terminated.

In the rest of the talk, Jellinek described results on storage and network tests. The authors found a bug in Rackspace persistent storage volumes that led to overcharges when volumes became stuck in an intermediate stage, unusable but still being billed. He then noted that the authors found that billing for IOPS in EC2 was subject to a substantial amount of aggregation on sequential reads and writes, which leads to underbilling for the customer. While this may seem good, the downside is that billing for IOPS in EC2 is still opaque and ultimately unpredictable to the customer. Finally, he noted the authors' discovery that billing for networking is also systematically slightly undercharged in EC2, and that they discovered a bug in Rackspace's network billing that led to more severe but less common undercharges.

In concluding, Jellinek suggested that providers should offer a billing API in which they expose key parts of their internal billing-related data and metadata (billing start/stop timestamps, network and storage billing data, etc.). He closed by noting that important future work could be done to better understand the tradeoffs inherent in implementing transparent, real-time billing interfaces, and how we could optimize billing interfaces, and the underlying resource-accounting mechanisms, in light of these tradeoffs.

An attendee from IBM asked whether they had tried testing from different locations other than from the university. Jellinek responded that they had not, but that that was definitely a good idea to pursue in verifying the results.

A second attendee asked whether they had tried creating a small cloud environment and polling it to see whether the actual resource consumption by the guest OS matched the cloud environment's measurements. Jellinek responded that they had not done that, but that it sounded like another good idea to pursue to verify their results.

A representative from VMware asked whether the authors had considered viewing billing as a statistical process, rather than as a process of exact resource accounting. Jellinek responded that, from the conversations he's had, it seemed that cloud providers are aware that exact resource accounting is a hard engineering problem that requires a significant amount of engineering effort and hardware. In practice, it is definitely conceivable that behind the scenes there is a certain amount of sampling and rounding down, such that any inconsistencies are in favor of the consumer, but ultimately allow the provider to conserve costs associated with exact resource accounting. This is speculation though, since to really know, one would have to understand the underlying mechanisms, which are proprietary.

A Case for Virtualizing the Electric Utility in Cloud Datacenters

Cheng Wang, Bhuvan Urgaonkar, George Kesidis, Uday V. Shanbhag, and Qian Wang, The Pennsylvania State University

Cheng Wang presented work on virtualizing the electric utility in cloud datacenters. He began by discussing how expensive it is to power a datacenter; the cost of building the IT infrastructure is often comparable to building the power infrastructure that is needed to keep the servers powered. The same is true for the utility bill that is used to power the servers each month. These are both on the same order as the IT investment itself.

Wang then discussed how a datacenter currently recoups operating expenses from tenants, and how it should actually be done. Today, operating expenses are recouped by charging for virtualized IT resources such as compute time, storage, and network resources. However, electricity is billed in a very different way. One common way it's billed is "peak-based pricing," which differs from how we consume electricity at home. Home consumers spend a certain amount per kilowatt-hour (kWh) of electricity consumed, and that's it. But for large consumers such as datacenters, they pay this charge as well as an additional charge that is connected with their pattern of consumption. Essentially, they pay more if their consumption is more bursty. So they may pay $0.05/kWh for usage up to some point, but would then pay $12/kWh for peak power consumption drawn above some wattage at a given point in time. The takeaway, says Wang, is that there is a peak-to-average pricing ratio of 3:1, and this ratio affects the economics of cloud computing. The question is how this gets passed on to the consumer.

Wang claims that it is passed on to consumers unfairly, in a way that does not accurately reflect cloud consumers' share of the peak-power consumption costs incurred by the cloud provider. In particular, he noted two shortcomings: a lack of fairness in how tenants get charged and a loss of cost-efficacy for both cloud tenants and providers.

To understand the unfairness, Wang encouraged the audience to consider two tenants that consume the same amount, but where tenant T1 has low variance, and T2 has extremely high variance, including consumption at peak times. In the current model, both tenants are charged the same amount because they pay fixed prices for virtualized compute resources, but T2 imposes a higher cost on the cloud provider than T1, because T2 contributes to peak-power demand that is three times more expensive than non-peak power.

The solution, Wang claims, is to virtualize the utility so that the energy costs a tenant incurs are passed on to them and not redistributed unfairly across all tenants. In essence, this means passing on the pricing structure of electricity to tenants themselves, so that these prices reflect the value the tenants derive from using that power. Wang related this to building exokernels and letting applications carry out their own resource-management solutions. Here, with a virtualized utility, tenants will be incentivized to use their resources more efficiently and will manage their usage more carefully based on those new incentives.

In practice, Wang says that this approach should be used with large, long-lasting tenants. It will be more difficult for them to take this extra factor into account and to optimize for cost, but will ultimately let them feel like they are really operating within the datacenter, with all its associated concerns, and provide a more equitable distribution of costs.

Phillip Gibbons (Intel Labs) noted that the main challenge seemed to lie in the peak pricing model itself. Passing on prices according to that structure means that you never want to be the customer who contributes to peak power, but you want to be right after them. Gibbons said that this seems like an artificial artifact of that pricing model. Wang responded that peak power is just determined by the behavior of the consumer, not time of day or anything else. Gibbons responded that it's so easy to game the system then, by just avoiding contributing to peak power consumption.

An attendee from Boston University noted that Wang had made a comparison to exokernels, and that one of the main challenges exokernels faced was that of aggregation: When you lower the level of abstraction, it makes it harder to perform aggregation. He asked whether cloud computing would similarly lose out on the benefits of aggregation if this layer of abstraction is removed. Wang replied that he did not suggest that the existing interface should be replaced but, rather, augmented. In his proposed interface, tenants would access their normal interface but also see metrics about how much they're contributing to peak power consumption.

HotStorage '14: 6th USENIX Workshop on Hot Topics in Storage and File Systems

June 17–18, 2014, Philadelphia, PA

Summarized by Rik Farrow, Min Fu, Cheng Li, Zhichao Li, and Prakash Narayanamoorthy

Keynote Address

Summarized by Rik Farrow (rik@usenix.org)

The Berkeley Data Analytics Stack, Present and Future

Michael Franklin, Thomas M. Seibel Professor of Computer Science, University of California, Berkeley

Franklin began by explaining that, in the Algorithms, Machines, and People Lab (AMPLab), they have been building the Berkeley Data Analytics Stack (BDAS), pronounced Bad Ass. BDAS is composed of many elements that were introduced at past HotCloud workshops, and Franklin told us he would walk us through the stack, what it is and why they built it. The reason for BDAS is that there are cascades of data being generated: logs, user generated, scientific computing, and machine to machine (M2M) communication. Instead of defining big data, Franklin provided the example of Carat, an application that collects data on apps and power use on smartphones, sends it to be processed using AWS and a BDAS framework called Spark, and then provides personal recommendations to the users of the apps about any energy hogs they may be running. What's interesting about big data is that you can see things that you can't with less data.

In order to make a decision, you have the envelope of time, money, and answer quality. You want to stay within that envelope, and the first thing researchers and programmers do is increase performance. When they hit the wall, they can trade off for less quality, or pay more for better quality. Another way to think about this is via algorithms, machines (warehouse computing), and people. In AMPLab, they want to use these resources to solve the big data problem.

MapReduce is a batch processing algorithm that proceeds through grouping and analysis, but there are a lot of other things that people do with databases. MapReduce can be specialized for streaming, working with graphs, or targeted for some other design point. The BDAS approach is to generalize, rather than specialize, MapReduce by adding general task DAGs (directed acyclic graphs) and data sharing, making streaming, SQL, and machine learning not just possible but faster than the specialized versions of Hadoop MapReduce. Spark, the BDAS core execution engine, is smaller than Hadoop, Storm (stream processing), Impala (SQL), Giraph (Graph), and Mahout (ML). And even with other modules added to handle machine learning, graph processing SQL, and streaming, Spark is still smaller than any of the other popular tools that can do just one of these activities. Like these other tools, Spark is open source, which meant, among other things, that students had to decide to produce quality code instead of producing more papers.

Franklin went on to describe several other projects, starting with MESOS, a system that allows sharing a cluster with different frameworks, like Hadoop, Storm, and Spark. Tachyon is an in-memory, fault-tolerant storage system that can be shared across different frameworks. Spark is now Apache Spark, and Hadoop may fade away, replaced by Spark or something else, not bad for a student project (Matei Zaharia's, who wrote about his creation for *;login:*).

RDD (Resilient Distributed Datasets), a key part of Spark, came out of a desire to improve the performance of Hadoop for machine learning. RDD caches results in memory rather than on disk, as Hadoop does, taking disk processing out of the critical path. RDDs maintain fault tolerance by including the transformations needed to recreate immutable stores of data. RDD also works well for SQL (Shark), which allows Hive queries to run without modification 10x to 100x faster. SparkSQL is inside of Spark 1.0, and Shark will be ported to run within Spark. BlinkDB provides a SQL interface that provides approximate answers, the benefit being speed by using sampling and displaying the error range. Future work will add the ability to perform online transaction processing (OLTP), which will require modifications to the way that RDD works to support frequent, concurrent updates. Graph processing (GraphX) is another ongoing project.

Franklin ended his talk with reflections and trends. While "Big Data" has the word "Big" in it, the real breakthrough isn't scalability—it's really about flexibility. With a traditional database, you begin a process called ETL (extract, transform, load), and import the data in a vault where you get a promise that your data will be reliably stored. In this type of database, there is one way in and one way out, but the price you pay is you lose access to that data except via SQL. In Hadoop, you split that up into storage and multiple methods of accessing that data. Another type of flexibility is that there is no schema: data can be unstructured. It can be structured (SQL schema), semi-structured (XML), and unstructured (Hadoop and others).

Also, in big data, people have ignored single node performance. That needs to change, because for small clusters, a single node is more efficient: Distributed systems are hard. In the AMPLab, they want to make BDAS work better for uses that require random write and random read, neither of which Spark and RDD are good at. These are the directions AMPLab is going.

Franklin finished a bit after his allotted time, and so Q&A was limited to a single question. Steve Muir (Comcast) pointed out that Franklin didn't talk about programming languages or traditional systems stuff, and wondered whether that has been a difficult change. While the Enterprise has adopted Java, Spark was written in Scala. Are there benefits from abandoning C++? Franklin replied that Steve is right, particularly with single node performance. Where you need to pay attention to low-level stuff is when you start benchmarking. Cloudera Impala (SQL) is written in C++. But there are things you can do to avoid JVM issues.

Money, Batteries, and Shingles

Summarized by Min Fu (fumin@hust.edu.cn)

qNVRAM: quasi Non-Volatile RAM for Low Overhead Persistency Enforcement in Smartphones

Hao Luo, Lei Tian and Hong Jiang, University of Nebraska, Lincoln

Hao Luo argued that since smartphones equipped with irremovable batteries have become more popular, it is time to rethink the memory volatility in smartphones. Luo proposed qNVRAM to reduce performance overhead without decreasing persistency level to less than traditional journaling and double-write persistency mechanisms.

Luo first introduced existing persistence enforcement mechanisms (including journaling and double-write schemes) in smartphones, which result in significant overhead due to additional I/Os. Luo then introduced four failure modes in Android smartphones, including application crashes, application hangs, self-reboots, and system freezes. All four modes could result in loss of application data. Given that more and more smartphones are equipped with irremovable batteries, the DRAM can be considered as a quasi NVRAM. Luo then proposed qNVRAM, an easy-to-use memory allocator. When one of the four failure modes happens, the application data in the qNVRAM pool can be restored. qNVRAM significantly speeds up the insert, update, and delete transactions in the SQLite use case.

Someone asked how to ensure data integrity in physical memory. Luo answered that ECC is implemented in the kernel and checksums are used in the database. Xiaosong Ma (Qatar Computing Research Institute) asked about the energy consumption. Luo answered that qNVRAM can reduce energy consumption. Someone asked, what if I dropped my phone on the floor? Luo answered that this rarely occurs. Dai Qin (University of Toronto) asked whether the data would be lost if the battery has died. Hao's answer was no.

Novel Address Mappings for Shingled Write Disks

Weiping He and David H.C. Du, University of Minnesota, Twin Cities

Weiping He proposed several novel static logical block address to physical block address mapping schemes for in-place update Shingled Write Disks (SWD). By appropriately changing the order of space allocation, the new mapping schemes improve the write amplification overhead significantly.

He started by describing SWD. He explained that in-place SWD requires no garbage collection and complicated mapping tables of out-of-space SWD, but suffers from the write amplification problem. He observed that a simple modification of the writing order of the tracks can reduce the write amplification, such as writing tracks 1 and 4 first. He then presented three novel mapping schemes, including R(4123), 124R(3), and 14R(23). These mapping schemes could improve update performance significantly when SWD space usage is less than 75%.

The first question was whether there are any workloads that revert the advantage of the new address mapping schemes. He replied that general workloads won't revert the advantage. The second question was whether the new address mapping schemes are designed to take advantage of temporal localities. He's answer was no. Nitin Agrawal (NEC Lab) asked about the age of the disk model used in the experiments. He replied that it's about 10 years old but is the newest they can get. Lots of researchers are still using it.

On the Importance of Evaluating Storage Systems' $Costs

Zhichao Li, Amanpreet Mukker, and Erez Zadok, Stony Brook University

Zhichao Li argued that evaluating storage systems from a monetary cost perspective becomes increasingly important. Li built a cost model, and evaluated both tiering and caching hybrid storage systems.

Li started by describing two kinds of hybrid storage systems: tiering and caching architectures. Li said performance alone is not enough to evaluate a hybrid system and dollar cost matters. An empirical TCO (total cost of ownership) study is also lacking when systems deploy SSD. Li then presented a cost model for hybrid systems, including upfront purchase as well as TCO. Li compared the two architectures of hybrid storage systems in terms of monetary cost. The results are workload-dependent. Li also said the cost model has several limitations, such as not including computer hardware, air-conditioning, and so on.

Three people asked questions about the cost model, including someone from Red Hat, Xiaosong Ma (Qatar Computing Research Institute), and Peter Desnoyers (Northeastern University).

A Brave New World (of Storage System Design)

Summarized by Zhichao Li (lzcmichael@gmail.com)

Towards High-Performance Application-Level Storage Management

Simon Peter, Jialin Li, Doug Woos, Irene Zhang, Dan R. K. Ports, Thomas Anderson, Arvind Krishnamurthy, and Mark Zbikowski,University of Washington

Simon Peter proposed a novel architecture to move the operating system storage stack off the data path for optimized performance. The idea is based on the observation that the operating system storage stack is becoming the bottleneck in the I/O path.

Simon began the presentation by stating that file system code is expensive to run. He illustrated the transition from today's storage stack to their storage architecture where the storage stack (block management and cache) is moved to user-level. Simon then discussed the proposed architecture in more detail. In their storage hardware model, the kernel manages virtual storage devices and virtual storage areas (VSA). The VSA maps from virtual storage extents to physical storage extents, and it is guaranteed that there is at least one VSA per application. The VSA also handles the global file name resolution and uses persistent data structures for high-level APIs.

They implemented a case study system using FUSE based on the idea illustrated above. Evaluation against Redis showed

that their system cut SET latency by 81%, from 163 µs to 31 µs. Simon then summarized their study by stating that leveraging application-level storage eliminates I/O bottleneck and achieves a 9x speedup compared with Redis, and it scales with CPUs and storage hardware performance.

Someone from VMware asked for the statistics of context switches in the system. Simon replied that there was basically no context switch since only library calls were involved. The attendee then asked whether it is possible to just modify OS for the same purpose. Simon said no and stated that doing so increases the attack space within the kernel and will only make the complex system even more complex. Geoff Kuenning (Harvey Mudd College) asked how the system scales when there are millions of files being accessed. Simon replied that it is possible that some applications will slow down, but other applications can access millions of files efficiently. Peter Desnoyers (Northeastern University) asked what the authors think of customizing OS functionality for different applications either in kernel or in user-level. Simon replied that it is hard to answer and continued by stating that user space is easy to experiment with and working with the kernel is complex, and so they choose to go with user-level. Steve Swanson (UCSD) asked about the fundamental difference between file access and block access. Simon replied that this is a good question and files have names associated with them. Margo Seltzer (Harvard School of Engineering and Applied Science and Oracle) asked whether Simon could comment on Exokernel since Exokernel appears similar to Arrakis. Simon replied that the difference lies in the fact that hardware is now different, which matters more for storage.

NVMKV: A Scalable and Lightweight Flash Aware Key-Value Store

Leonardo Mármol, Florida International University; Swaminathan Sundararaman and Nisha Talagala, FusionIO; Raju Rangaswami, Florida International University; Sushma Devendrappa, Bharath Ramsundar, and Sriram Ganesan, FusionIO

The idea that Leonardo Mármol presented for a flash aware key-value store is to examine the Flash Translation Layer (FTL), instead of the upper-level key-value software, to leverage SSD in an optimal way.

Leonardo began by introducing key-value stores and then discussing the limitations of existing solutions—for example, better performance only on HDD and older SSDs, requiring compaction/garbage collection and introducing a write amplification problem, from 2.5x to 43x in one example. Leonardo then took a look at FTL, which manages data in a way similar to a key-value system, and proposed to move almost everything (except the key-value hashing mechanism) to the FTL for optimal efficiency. This is a new approach by cooperative design with FTL to minimize auxiliary write amplification, maximize application level performance, and leverage FTL for atomicity and durability by extending the interface. Leonardo then discussed the classes of key-value store: disk optimized and SSD optimized.

Leonardo next talked about the design: Sparse address mapping (LBA = hash(key)) leads to FTL sparse mapping, and translates logical to physical addresses. This is made possible by the extended FTL interface (i.e., atomic write and atomic trim; iterate, query an address). Leonardo further stated that hashing and collision is achieved by polynomial probing: Their software tries eight positions before failing. In their evaluation, microbenchmark results are generally positive and beat LevelDB even at low thread counts and without FS cache; the YCSB benchmark shows that their system beats LevelDB in all conditions as well. Leonardo concluded by proposing FTL cooperative design for simple key-value store design and implementation for high performance and constant amounts of metadata.

Michael Condit (Red Hat) asked why LBA and PBA are two to three times larger in space. Leonardo replied that it is because of a more efficient caching implementation. Margo Seltzer asked why they only compared with LevelDB when there are lots of other available key-value stores. Leonardo replied that there is no particular reason why they chose LevelDB and it is future work to compare against other key-value stores. Margo also commented that there is paper from FAST '14 that looks into the cooperative file system design with SSD, and suggested Leonardo look into that. Leonardo agreed. One attendee from VMware asked about operations for key lookup. Leonardo replied that it depends on the hashing and the key being looked up. In most cases, Leonardo continued, there is only one I/O for key lookup, and under other cases, it may need multiple operations. They have set a limit of eight lookups before giving up.

FlashQueryFile: Flash-Optimized Layout and Algorithms for Interactive Ad Hoc SQL on Big Data

Rini T. Kaushik, IBM Research—Almaden

Rini Kaushik introduced FlashQueryFile: a flash-optimized layout and algorithm for interactive ad hoc SQL queries on big data. The idea is to optimize the data format in consideration of the underlying SSD characteristic for optimized big data analysis usage of flash.

Rini started the talk with the motivation that there are many use cases for ad hoc SQL queries, and storage plays an important role in ad hoc big data queries. Flash in a big data stack is faster and cheaper than DRAM, is non-volatile, and incurs lower energy and better total cost of ownership. Rini then discussed the challenges in flash adoption: Systems are currently HDD optimized; suboptimal performance/dollar on flash; flash sequential bandwidth is only 2–7x faster than HDD; flash is popular in OLTP, but not so much in OLAP or SQL data processing. Rini then took a look at the opportunity for data reduction in OLAP by looking at TPC-H Query 6. For selectivity, there are lots of irrelevant data reads. Rini then talked about flash optimized FlashQueryFile (FQF) challenges: Skipping data is intuitive in the projection phase as row IDs of interest are already known; simple random accession of data is not feasible; the same layout does not work across various column cardinalities.

Rini then introduced selection-optimized columnar data layout and projection-optimized columnar data layout in FQF. In the evaluation, Rini talked about the experimental setup. In terms of results, FQF achieved 11x–100x speedup and achieved a 38% to 99.08% reduction in data read compared with ORCFile on flash.

Margo Seltzer asked what happens when OLTP, instead of OLAP, is made flash aware. Rini replied that scalability is one issue and another issue is that the majority data of OLTP is read-only and no update is required in this case.

Hotpourri

Summarized by Cheng Li (chenglii@cs.rutgers.edu)

Assert(!Defined(Sequential I/O))

Cheng Li, Rutgers University; Philip Shilane, Fred Douglis, Darren Sawyer, and Hyong Shim, EMC Corporation

Cheng Li presented his research on revisiting the definition of sequential I/O. He first motivated the work by addressing the importance of the concept of sequential I/O, because many optimizations were made based on the concept of sequentiality to improve performance of disk or tapes. Many applications leverage sequential I/O such as caching and prefetching. In addition, many non-rotational devices such as SSDs favor sequential I/O because writes with large I/O size can reduce the number of SSD erasures, which improves SSD lifespan. Finally, a clear definition of sequential I/O helps classify workload characteristics, which will benefit the system researcher, trace analysis, and synthetic I/O generation.

Cheng showed a few definitions of sequential I/O and did a live survey with the audience, asking them which definition best matched their intuition. More people from the audience preferred the second definition of sequential I/O, but there was no consensus. Cheng suggested that sequentiality is heavily used in literature but rarely defined, and defined in an inconsistent way. Cheng used a big-data driven approach to compare different sequentiality metrics.

Cheng reviewed several definitions of sequentiality and properties of sequential I/O that might impact the sequentiality definition. First, he showed the canonical definition of sequentiality, the consecutive access ratio, defined as the fraction of consecutive accesses. Then he presented another way of measuring sequentiality, the consecutive bytes accessed. The consecutive bytes accessed incorporates the I/O size so it captures more properties compared to the simple consecutive access ratio. Then Cheng presented a strided range property that allows gaps, small backward seeks, and re-access continuations to be considered as sequential accesses. This property improves the strict definition of sequentiality. Cheng presented the multi-stream property that leverages application information to separate out mixed I/O streams and an inter-arrival property that defines consecutive I/O requests with long intervals as non-sequential.

Cheng presented the methodology of this study. He looked at the different combinations of the sequentiality properties in the definition. Then he tried to use different metrics to measure sequentiality of storage traces. He compared a ranked list produced by different metrics. If all metrics provide the same view towards sequentiality, then it doesn't matter which definition to use; otherwise, it's necessary to pick metrics that best align with the use case and study the correlation of different metrics.

Cheng presented several interesting results. The primary findings are: (1) Different sequentiality metrics provide different views towards sequentiality; (2) the metrics that incorporate I/O size show a more consistent view when quantifying access patterns; (3) many sequentiality metrics are negatively correlated, which means the results can change completely depending on which metrics to use; (4) although there might not be a global metric for sequentiality, system researchers should pick one that best aligns with the use case and state which definition to use.

Peter Desnoyers (Northeastern University) asked about what if the same application uses different metrics. Cheng answered that he looked at caching as an example, and different metrics indeed produce diverse different sequentiality values, which makes it hard to make a conclusion based on different metrics.

Towards Paravirtualized Network File Systems

Raja Appuswamy, Sergey Legtchenko, and Antony Rowstron, Microsoft Research, Cambridge

Sergey Legtchenko motivated the work by comparing VHD and NFS with emerging hardware. Then he asked, what are the tradeoffs in choosing one versus the other? Are current mechanisms sufficient with emerging hardware?

Sergey quantified the VHD overhead in the datacenter today and contrasted this with the VHD overhead in emerging datacenters. VHD causes high overhead but is fully compatible with other features. NFS incurs low overhead but is incompatible with other features. Clearly, there is a need for a new data access mechanism that can avoid nesting like NFS, and also enable hypervisor interception like VHD. So this work proposed a paravirtualization scheme.

The paravirtualized NFS client performs first-level DRAM caching and passes through cache misses to the hypervisor that acts as a proxy, while the hypervisor NFS client achieves second-level caching with flash or memory. Existing protocols, like SMB, can be used for forwarding requests to the NAS server. It is non-invasive, backward-compatible data access. There is no revisiting the guest-host division of labor.

There are several paravirtualization tradeoffs. The advantages of paravirtualized NFS client shows performance similar to NFS, feature compatibility similar to VHDs. In addition, it supports end-to-end semantic awareness. So clients can use NFS protocols for accessing and sharing data. And unlike VHD, files stored within an NFS server can be shared.

There are still challenges: e.g., implementing the low-overhead guest-host file I/O bypass, implementing file-level protocols. Still, there is a lack of full-system virtualization.

In conclusion, storage hardware is changing quickly; there is more low-latency RDMA-based access to storage class memory. There is a need for flexible, overhead-free data-access mechanisms. NFS is overhead free but incompatible with other features. VHD is compatible, but suffers from overhead due to translations. The proposed paravirtualizing NFS client is as fast as NFS, compatible as VHD. Paravirtualized NFS is non-invasive and builds on well-established protocols and interfaces.

The first question was for a clarification to avoid the confusion between the term NFS that the talk used referring to the general concept of network file systems and the NFS protocol. Sergey answered that they were using NFS to refer to a network file system, not the NFS protocol. The second question was on the experimental setup: Which version of the SMB protocol were they using and were all stacks Microsoft-based? Sergey answered that they are using SMB 3.0, which enables direct access over RDMA (SMBd), the host runs Hyper-V and the guest runs Windows Server 2012.

Evaluation of Codes with Inherent Double Replication for Hadoop

M. Nikhil Krishnan, N. Prakash, V. Lalitha, Birenjith Sasidharan, P. Vijay Kumar, Indian Institute of Science, Bangalore; Srinivasan Narayanamurthy, Ranjit Kumar, and Siddhartha Nandi, NetApp Inc.

Prakash first compared the triple replication of data in Hadoop with double replication. The Hadoop replication, while achieving high data protection, increases storage overhead significantly. Another useful scheme is the RAID6 + mirroring, which uses two parity blocks to ensure adequate resiliency.

The challenge to address was to apply inherent double replication coding schemes to improve locality for Hadoop. Prakash introduced the Heptagon-local code (HLC), which is an alternative code with inherent double replication. The HLC has reduced overhead for the desired resiliency but there is an issue relating to data locality. Clearly, it's important to leverage data locality to ensure computation is completed locally. The way they address the locality is to modify the HDFS to permit coding across files.

Prakash used several slides to explain how to build the Heptagon code, providing some insights on the Heptagon codes as a rearrangement of RAID+m. The resiliency of Heptagon code can tolerate two out of five node failures, recovered by parity. How to extend the code to a Heptagon code and how to recover from two/three node erasures and overhead/resiliency results were discussed next. Finally, Prakash discussed data locality for the Heptagon code and showed MR performance in Hadoop.

Someone asked about making a comparison with a class of error-correcting codes known as Fountain codes.

SSDelightful

Summarized by Prakash Narayanamoorthy (prakashnarayanamoorthy@gmail.com)

The Multi-streamed Solid-State Drive

Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho, Samsung Electronics Co.

Jeong-Uk Kang presented a multi-stream-based approach for improving the efficiency of garbage collection (GC) in solid state drives (SSDs). She started off by saying that SSDs share a common interface with HDDs, which facilitated faster adoption of SSDs. However, since rotating media and NAND-based SSDs are very different, such a common interface is enabled by the use of a Flash Translation Layer (FTL) in the SSDs. The FTL has two purposes; one is logical mapping of blocks and the other is to do bad-block management performed via GC, which serves to reclaim space and to erase blocks. However, in the current implementations, GC is an expensive operation and it highly affects the SSD life.

The authors presented a new approach for improving the efficiency of GC. Their idea is to create streams while writing data into SSD. The various streams are chosen based on the life expectancy of the data that is being written. Kang argued that the best performance is obtained when the lifetime of data being written is determined by the host system itself and passed on to the SSD, which then determines the stream ID. In this approach, GC can be done in a targeted manner, on the blocks corresponding to the individual streams. The idea was tested in Cassandra, using a new interface that implements up to four different streams and by using the YCSB benchmark. Performance with the "TRIM on" feature was also evaluated. The test-case with four streams showed the best performance.

Someone asked whether the approach was specific to flash, to which Kang remarked this to be general to all SSDs. As to whether modifications to Cassandra were necessary, Kang replied in the negative. Someone asked whether the SSD block layer had to be modified. Once again, Kang noted this as being not necessary.

Accelerating External Sorting via On-the-Fly Data Merge in Active SSDs

Young-Sik Lee, Korea Advanced Institute of Science and Technology (KAIST); Luis Cavazos Quero, Youngjae Lee, and Jin-Soo Kim, Sungkyunkwan University; Seungryoul Maeng, Korea Advanced Institute of Science and Technology (KAIST)

Young-Sik Lee presented a new architecture for improved in-storage processing in SSDs, which seeks to improve I/O performance and hence the life of the SSD. The idea was to use an active SSD architecture, which will perform external sorting algorithms more efficiently. Lee started off by stressing the importance of I/O in data-intensive computing and the need to migrate to SSDs to improve the I/O performance. Although in traditional SSDs, the storage and the host processor remain separate, in active SSDs, there is room for in-storage processing, which can further improve the I/O performance of the SSD.

Lee said that although there are existing architectures for active SSDs, they only perform aggregate functions (like min, max, count). However, given the increased processing power of active SSDs, more complex functions can be performed in-storage.

The new architecture considered by the authors would allow computation of non-aggregate functions. Specifically, the focus was on an operation referred to as the active-sort, which improves the efficiency of external sorting algorithms. Lee remarked that such algorithms played a major role in the Hadoop MapReduce framework. In the traditional way of sorting, the SSD stores partially sorted chunks, which are read out by the host to do the final merge. This merged output is then written back to the SSD to be used by the next stage of processing. In active sorting, the SSD simply keeps the partially sorted chunks, and when the next stage of processing demands the merged data, the merging happens inside the SSD itself; this result is fed to the next stage. Thus there are significant savings in write bandwidth, since the host need not write back the merged data to the SSD. There is, however, a small increase in read bandwidth to perform the in-storage merging. An implementation was carried out on an open SSD platform, consisting of four channels, each of 32 GB. The SORT benchmark was run on the new architecture, and measured quantities include read/write bandwidth and elapsed-time for the sort operation. A comparison was performed against the NSORT and QSORT algorithms, and gains were demonstrated, especially when the host memory was small compared to the size of the data being sorted. Lee concluded by saying that their next plan was to integrate this architecture with that of Hadoop MapReduce.

In the question and answer session, someone felt that even though there are savings in I/O for the SSDs, there might not be an improvement in its lifetime, since there are other factors affecting longevity. Another questioner wondered whether the proposed architecture could be applied in situations other than the MapReduce framework. Lee noted that this was also one of the points that they were actively thinking about.

Power, Energy, and Thermal Considerations in SSD-Based I/O Acceleration

Jie Zhang, Mustafa Shihab and Myoungsoo Jung, The University of Texas at Dallas

Jie Zhang generated a considerable amount of conversation among the audience around the topic of whether multi-resource SSDs that promise high I/O can deliver it at a reduced power and energy consumption, as is commonly believed, or whether they needed to consume energy to deliver the improved I/O performance. Zhang started off by noting that a single SSD chip has a very limited I/O rate, and it is common to use many chips to match the PCI-Express bandwidth. Modern SSDs also come with many channels and many controllers and cores to handle multiple tasks in parallel. The number of components integrated into these many-resource SSDs have increased by more than 62 times with respect to what was seen during the early 2000s.

While all these new components were added for improved I/O and latency performance, Zhang highlighted the lack of studies on the power, energy, and thermal considerations for these new many-resource SSDs.

Zhang presented many measurements to show that, contrary to popular beliefs, power, energy, and thermal properties of the new SSDs are much worse than traditional SSDs. Measurements revealed that while single-purpose SSDs measure around 95–120 degrees Fahrenheit (operating temperature), multi-purpose SSDs could go up to 180 degrees Fahrenheit during their operation. It was also demonstrated that the improved latency of the multi-purpose SSDs comes with an overhead of around seven times increased dynamic power consumption. Zhang also pointed out that due to such enormous power consumption, the internal mechanism of the SSD automatically reduces the performance in response to the heat generated. Zhang concluded that the overheating problem and power throttling issues are holding back state-of-the-art SSDs.

Someone asked whether making the SSDs byte-addressable and providing them with direct access to the memory bus would eliminate some of these power consumption issues. Zhang said that the current results may be affected by the suggested modifications. Another questioner wondered about profiling heat generation patterns of the various components and suggested studying which of the many components present in the multi-purpose SSDs contributed to the increased power consumption. Zhang noted that more measurements are needed in that direction and reserved that for future work.

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

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