usenix ;login:

AUGUST 2011 VOL. 36, NO. 4

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix UPCOMING EVENTS

20th USENIX Security Symposium (USENIX Security '11)

August 8–12, 2011, San Francisco, CA, USA
http://www.usenix.org/sec11

Workshops co-located with USENIX Security '11 include:

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

SPONSORED BY USENIX, ACCURATE, AND IAVOSS

August 8–9, 2011
http://www.usenix.org/evtwote11

4th Workshop on Cyber Security Experimentation and Test (CSET '11)
August 8, 2011
http://www.usenix.org/cset11

USENIX Workshop on Free and Open Communications on the Internet (FOCI '11)
August 8, 2011
http://www.usenix.org/foci11

5th USENIX Workshop on Offensive Technologies (WOOT '11)
August 8, 2011
http://www.usenix.org/woot11

2nd USENIX Workshop on Health Security and Privacy (HealthSec '11)
August 9, 2011
http://www.usenix.org/healthsec11

6th USENIX Workshop on Hot Topics in Security (HotSec '11)
August 9, 2011
http://www.usenix.org/hotsec11

Sixth Workshop on Security Metrics (MetriCon 6.0)
August 9, 2011
http://www.securitymetrics.org/content/Wiki.jsp?page=Metricon6.0

23rd ACM Symposium on Operating Systems Principles (SOSP 2011)

SPONSORED BY ACM SIGOPS IN COOPERATION WITH USENIX

October 23–26, 2011, Cascais, Portugal
http://sosp2011.gsd.inesc-id.pt

ACM Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT 2011)

SPONSORED BY ACM IN ASSOCIATION WITH USENIX

December 4–5, 2011, Boston, MA
http://chimit.acm.org/

25th Large Installation System Administration Conference (LISA '11)

SPONSORED BY USENIX IN COOPERATION WITH LOPSA AND SNIA

December 4–9, 2011, Boston, MA, USA
http://www.usenix.org/lisa11

ACM/IFIP/USENIX 12th International Middleware Conference (Middleware 2011)

SPONSORED BY ACM AND IFIP IN ASSOCIATION WITH USENIX

December 12–16, 2011, Lisbon, Portugal
http://2011.middleware-conference.org/

10th USENIX Conference on File and Storage Technologies (FAST '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGOPS

February 14–17, 2012, San Jose, CA
http://www.usenix.org/fast12
Paper titles and abstracts due: September 20, 2011

9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGCOMM AND ACM SIGOPS

April 25–27, 2012, San Jose, CA
http://www.usenix.org/nsdi12
Paper titles and abstracts due: September 27, 2011

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

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Jennifer Peterson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for nonmembers are $125 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

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

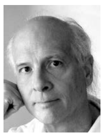
usenix ;login:

AUGUST 2011, VOL. 36, NO. 4

OPINION

Musings

RIK FARROW



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

It feels like the '90s all over again. No, I don't mean the IPO of LinkedIn for a ridiculous figure (16 times earnings). Rather, it's the rash of highly publicized security incidents, with Sony being the most recent—and the most frequent. LulzSec has bragged online that their "hack" was embarrassingly easy: a simple SQL injection [1]. The attacks on Sony's networks worldwide have led to a flood of released data, including info on registered users and source code.

The Sony attacks, and those on HBGary Federal and PBS.org, seem to have been done for political reasons. After a decade where computer attacks have been primarily focused on financial gain, this marks a rare turn toward vigilantism.

What these attacks also reveal is the very porousness of network security. When the only way we would learn of attacks was after the California law on disclosure of personally identifying information forced an announcement, it wasn't obvious just how often organizations were being broken into. Now it seems as if we are back in the era of unpatched Linux systems being taken over by automated attacks.

But this time around, things are different.

The Difference

Much has changed between 1999 and 2011. The publication of exploits has largely gone underground, as exploits are now sold on the black market both to governments and to criminals. Attackers are no longer motivated by becoming famous (or infamous), with the recent exceptions of Anonymous and LulzSec. Instead, criminal organizations use exploits to take over Windows PCs for use in botnets, or steal databases loaded with credit card information. The monetization of exploits has long been under way.

But some things have not changed at all.In 1999, only a fool would claim that their Internet-connected system was totally secure, and proof against all attacks. And that is just as true today. Only we behave as if it isn't so.

And instead of attacking Internet-facing *nix servers, today's attackers can rely on a different technique, one that totally trashes any concept of having a "network perimeter." They can gain a foothold inside any network through the use of email and Windows PCs. The Google attack, announced over a year ago (January 2010 [2]), has just been repeated. I believe that similar attacks are behind the exploitations of RockYou (32 million passwords stolen), Gawker (300,000), and certainly of HBGary Federal.

This type of targeted attack requires some research on the attacker's part, to determine the subject line and content for an email message likely to make it pass spam filters and be opened by the intended victim. You can read more about an example of this particular attack in Ned Moran's article in this issue about the Advanced Persistent Threat. But it is not just APT that can take advantage of "spear phishing," sending email containing exploits to targeted individuals. Anyone willing to spend enough time to understand the targeted organization, perhaps by using LinkedIn to uncover links between individuals that could be used to fashion an effective email message, could successfully breach a network today.

The only effective defense against such attacks is either using the mail command-line program on a *nix system without X Window to read email or keeping all critical assets on networks segregated from networks with GUI users. I don't believe for one minute that people are going to go command-line anytime soon, so it looks like we need to consider the alternative approach.

Assume the Worst

Dominique Brezinski explained, in a private email, that he has had a lot of success with the following alternative approach (quoted with permission):

> It is my opinion that in any computing environment with a non-trivial population you must assume some client devices and some user accounts are compromised. There are a couple advantages to making this assumption:
>
> Your defensive strategy no longer makes a delineation between external and internal adversaries. It is only a single, easy hop from outside in. Just assume it happens. Tailor your defensive strategy around the insider problem.
>
> Whether you own the client device or not makes little difference.

Dom's first point echoes what I have already written: assume the worst—your network has already been compromised. Even if it hasn't been (as unlikely as that is), assume it has. Then behave accordingly.

Much has been written about the insider threat (see, e.g., [3]), but Dom is thinking more tactically. Assume your attacker is an insider. How do you go about protecting your critical assets? Isolating them, using firewalls that separate out servers with critical information, is a good start. This is nothing new, as it was considered best practice in the '90s—even as it was usually ignored.

Just isolating servers with critical data is not enough. People still need access to those servers, as do applications. So you must limit that access carefully, both for applications and for users. Dom suggested that users who must have access use "authenticators that are unique per session" and so cannot be stolen or reused. And that users' access does not imply unlimited authorization: in other words, put a system in place that restricts access and logs all activities. It would have been nice if Sony had noticed that someone was downloading megabytes of source code. It would have been better if this had not even been possible. But at the very least, having a system in place that notices unusual activity and notifies someone is not just reasonable, it is a requirement for securing critical data.

Instead of just beating up on Sony, let's look at another example: WikiLeaks and the 251,287 diplomatic cables [4]. When I first learned of this immense treasure trove of secret or sensitive information, allegedly leaked by a young soldier, I couldn't believe it. Why on earth would an intelligence officer stationed in Iraq have access to diplomatic cables? This sharing of data that doesn't seem as if it belongs on a soldier's laptop came about as a reaction to the lack of sharing of intelligence that may have contributed to the success of the attacks of 9/11. But suppose that a system was in place that detected the collection of this vast amount of data, including cables going back more than 45 years? If even a reasonable amount of monitoring of access was being performed, would this soldier have been able to collect so much information without being detected? Keep in mind that Bradley Manning is now in prison not because of a system that managed and logged access, but because he chatted online with Adrian Lamo.

To ground these ideas, let's imagine you have a database that serves information to customers using a Web front-end. And to make things interesting, let's also imagine that this database contains a customer's name, address, and credit card information. You want your customers to have a pleasant experience while using your system. You don't want them to be able to download the entire database of sensitive information, just their own information. This is what is meant by limiting authorization: the customer has access to her own information only. Additional controls would include isolating the database on its own network, and using firewalls to limit access to that database to only the Web server's application and to the people who must manage the database. You must also include controls that prevent the dbadmins (or an attacker who has taken over their desktops) from abusing their privileges. And that is the really difficult part.

When you consider that the database of information used to restore the cryptographic info for lost SecureID tokens was stolen from RSA, a security company, you can see that the concept of isolating critical assets, even when their compromise will lead to terrible results [5], is often ignored.

On the other hand, I hope it is useful to view your network of email and Web connected desktops as already compromised. If it isn't already, it soon will be. Keep your valuables someplace else. Please.

Spam Kings

I don't spend all of my time lamenting the lack of real security or attending cool workshops like HotOS (see the reports in this issue). I sometimes get to read really interesting stories from researchers who have been doing tremendous work.

I interviewed Stefan Savage (UCSD) for this issue, and I can't say that it was hard work. Stefan is a great storyteller, and his story is a compelling one. Starting in 2006, Stefan, along with several other professors at UCSD, UCB, and ISI, began investigating spam. They started by looking at how spam is delivered, moved on to the botnets that deliver most spam, and, finally, studied the fulfillment side of spam. As we all know, if no one clicked on the links in spam and then actually bought something, spam would have vanished years ago. But spam is still a successful marketing tool.

Stefan tells us about how they got to the point where they were actually buying pharmaceuticals, fake Rolexes, and software by following spam links. This was the latest step in a long process, and the results were published in a paper [6] at the IEEE Symposium on Security and Privacy (Oakland) this summer. I enjoyed listening to Kirill Levchenko, in the crowded ballroom at the Claremont Resort, explain how hard it was to get permission to actually buy spam-advertised goods, as this involved both using research funds (you want to do WHAT!?!) and tricky negotiations that made it possible to track the transactions via credit card companies. And this is just one of the stories Stefan has to tell. I also liked the one about how the FBI was about to arrest them at a USENIX workshop, but you should read this for yourself.

Stefan's interview also provides a clear window into successful research. Stefan and his associates followed paths that were not always successful. Sometimes he worried that his students (and other advisors' students) were wasting their time, only to be surprised by the results. And the results so far have been over 14 papers published, including three at USENIX Security '11 and one at CSET '11.

Ned Moran has written about APT, using a recent example of a spear phishing attack against US government employees. APT differs from attacks by Anonymous in that it requires teams of people ready to react to a successful penetration. The actual technology does not appear that exciting, although I believe the details of the remote access tool that Ned dissects will prove interesting to *;login:* readers.

Raphael Mudge has been working with the Metasploit penetration testing software to create his own front-end, Armitage, that makes it easier for a team to work together. Raphael's tool is designed for helping red teams practice attacks, and if you are interested in the attacker's perspective, I suggest you read his article.

Peter Gutmann has provided us with a short article about the problem with SSH key fingerprints. It is not that the fingerprints are useless, it is that they are both not used properly and too easily abused.

Ben Hindman and a long list of co-authors explain Mesos, a system that works with Hadoop and other frameworks, such as MPI, used for performing work in parallel on many systems. Mesos is itself a framework that improves the performance of parallel tasks by using dynamic partitioning of systems, instead of the static partitioning supported by Hadoop and MPI.

Tom Limoncelli and Doug Hughes, co-chairs of LISA '11, explain why they believe that DevOps, the chosen theme for LISA, is important. DevOps implies close collaboration between sysadmins and developers, which is how systems are being developed today.

David Blank-Edelman continues on the theme of Web frameworks with Mojolicious. Mojolicious is similar to Dancer, but only when you first encounter it. Mojolicious provides a stand-alone (no other modules required) and complete Perl-based Web framework, designed to make difficult things, such as managing sessions, simple.

Peter Galvin has renamed his column "Galvin's All Things Enterprise." Peter's first installment covers that buzziest of buzzwords, the cloud, but without getting lost in the clouds. Peter defines cloud computing from an IT perspective and explains why it is important.

Dave Josephsen strays away from monitoring to tell us about a project he has been working on for months. Dave shares with us descriptions of the shell-based batch processing system that replaced the hundreds of scripts he inherited when he started working as a sysadmin.

Robert Ferrell takes us on a "circuitous ramble" through the different types of job interviews, before sliding into talking about entitlement and security. His own recent encounter with a surreal interview provides fodder for his column.

Elizabeth Zwicky reviews four books, including a couple she really likes, and one on interviewing for security positions. She did not communicate with Robert Ferrell, so this is a serendipitous occurrence. I contributed two book reviews myself this issue, including one on a novel written by a Google employee about the potential for abuse of data collected about users of a company that sounds vaguely like Google—not quite Google, as this fictional company has access to lots more data than Google does.

Finally, we have the HotOS summaries. HotOS is one of my favorite workshops, even if it only happens once every two years. This is the place for OS researchers to expose their sometimes very far-reaching ideas in front of an audience of critical thinkers.

Before leaving you, dear reader, I want to remind you that your network has been compromised. How do I know? I don't have to know, I can guess. Unless you are running a single stripped-down *BSD system on a firewalled network with no GUI, the odds that your network has working bots on it, along with remote access tools, is high. Even if your network hasn't been compromised, how would you know?

I like to put a sniffer outside my network and analyze the traffic I find there. This is possible for my network because there are only two users and a couple of lightly used Web servers. If you reconfigure your networks so that your critical assets live behind severely restricted firewalls, you could do this as well for the traffic going between the protected network and the rest of your networks. But in the "let's keep things as wide open as possible so we can make more money, uh, get work done" mode, real security is just not possible.

References

[1] "Hackers Claim to Have Hit Sony Again": http://www.reuters.com/article/2011/06/06/us-toni-cybersecurity-sony-idUSTRE75563L20110606.

[2] Rik Farrow, "Google Attacked via Email": http://blogs.usenix.org/2010/01/14/google-attacked-via-email/.

[3] Dawn Cappelli, Andrew Moore, and Timothy Shimeall, "Protecting against Insider Threat": http://www.sei.cmu.edu/library/abstracts/news-at-sei/security-matters
200702.cfm.

[4] United States diplomatic cables leak: https://secure.wikimedia.org/wikipedia/en/wiki/United_States_diplomatic_cables_leak.

[5] Dan Goodin, "Stolen RSA Data Used to Hack Defense Contractor": http://www.theregister.co.uk/2011/06/06/lockheed_martin_securid_hack/.

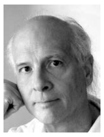
[6] Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Mark Felegyhazi, Chris Grier, Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage, "Click Trajectories: End-to-End Analysis of the Spam Value Chain," *Proceedings of the IEEE Symposium on Security and Privacy*, May 2011: http://www.imchris.org/research/levchenko_sp11.pdf.

Interview with Stefan Savage

On the Spam Payment Trail

RIK FARROW

Stefan Savage is a professor of computer science and engineering at the University of California, San Diego. He is also director of the Center for Network Systems (CNS) and co-directs the Cooperative Center for Internet Epidemiology and Defenses (CCIED), a joint effort between UCSD and the International Computer Science Institute.
savage@cs.ucsd.edu



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

I have been following, with great interest, the work of many researchers in following the spam trail. Over time, I noticed that a number of researchers were obviously working together a lot, combining their efforts into what appeared an immense task: understanding of an entire underground economy.

I was fortunate enough to find Stefan Savage, one of the primary investigators in this work, in a storytelling mood. What follows is his detailed account of successes and failures, approaches that appeared to be dead ends where students prevailed, and how we now have a large body of solid research in an area that has confounded many attempts to come to grips with its many interlocking pieces.

Stefan invited Vern Paxson and Geoff Voelker to participate in the email interview process. Both made suggestions and provided corrections to Stefan's tale, and were content not to have their contributions made explicit in this process.

If you are attending USENIX Security this year, you will find three papers related to this story, and one at CSET. And there are more than a dozen other papers that have come from this collaboration.

The Interview

[RIK] How did you get interested in assigning value to malware and spam?

[STEFAN] The truth is that Vern Paxson (UC Berkeley), Geoff Voelker (UC San Diego), and I started down this path back in 2006. We'd been working together for quite a few years on large-scale attacks (e.g., worms, viruses, DDoS, etc.), and while we'd had lots of technical successes looking at those problems head on, it was pretty clear that the world wasn't getting any more secure. Around that time we became exposed to the breadth of activity involved in underground trading of compromised accounts, credit cards, spam mailers, email lists, etc.—anything you could think of. This was really our inspiration, because we came to recognize the role that the profit motive was playing in all this (although spam was key to this evolution, we wouldn't make the link until later).

I think it helped that at the time I was reading a book on the history of the drug war and the failings of supply reduction as a strategy due to the poor understanding of drug distribution economics. We came to see that our community had a similarly poor understanding of the value chain for economically motivated attackers and thus didn't understand that our various technical interventions actually played minor roles, at best, in mitigating their actions.

During the summer of 2006, Vern had an intern up at ICSI, Jason Franklin from CMU, and we got him to focus on a big trace of underground IRC data we had gotten our hands on. The analysis was ultimately published in CCS [Computer and Communications Security conference, 2007], with Jason's advisor Adrian Perrig as a co-author [1], and while it was shallow and there was quite a bit we got wrong, I think it marked the turning point for us. From then on we started thinking much more holistically about our security work, trying in particular to understand what the underlying economic models were and how we might access those through measurement.

Around the same time (maybe just a bit earlier) we got into spam, due to a project that Geoff Voelker had started using a large spam sink. This was an old .com domain with no real users for which all the received mail was expected to be spam. David Anderson and Chris Fleizach (two master's students at UCSD) took this data and started crawling all the URLs embedded in the spam emails. They would then render any of the Web pages they found and cluster them together based on image similarity, using a technique called image shingling. The goal of that study [2] was to look at the servers being used to host the sites advertised in spam and look at the dynamics of their lifetime. Again, we didn't fully understand the subject matter, but we'd clearly found another piece of the puzzle—that one might want to consider the sites being advertised independently from the advertisements (i.e., the spam). It also helped us to build up some of the infrastructure experience that we'd need in the years to come. Finally, this infrastructure *inadvertently* got us into looking at Storm.

By the way, I should be up front that two people who definitely influenced our thinking early on were David Aucsmith (Microsoft) and Rob Thomas (Team Cymru). Dave I met through an NRC study I was on; he was the first person I'd run into who was talking about the *price* of sending spam and mounting DDoS attacks (as opposed to some technical quality like packets per second or spams per day). Rob was in this space very early, and I think several of us knew of him through different channels (via CAIDA, NANOG, etc.). We were heavily influenced by his world view, part of which got documented by *;login:* in his article [3], as well as his terminology (miscreants, underground economy, etc.).

[RIK] That article about the underground economy is still very popular today, judging by the number of downloads from the USENIX Web server. And the terminology presented was not just that invented by Rob Thomas. I had to ask him and the other authors to define many other terms, taken from the underground sites they had gained access to, like "bins," "rippers," "cashiers," and "wells," none of which matches its dictionary meaning.

Team Cymru did a lot of work by monitoring IRC and other servers. You mentioned collecting spam, and two papers where you analyzed links and Web pages that came from the collected spam. Did you do more work with your spam sources?

[STEFAN] We asked Chris Kanich, who was then a fairly new UCSD PhD student, to take over the spam feed, and we had some kind of idea about maybe trying to look at click-through rates by looking at spam-advertised URLs and then seeing if campus would let us monitor how many outbound visits went to those same domains, or something similar. In a bit of serendipity, our machine was suddenly hit by a large (at the time) DDoS attack, greater than 1 Gbps, which got campus network managers to notice. We weren't the only ones experiencing this, however.

What had happened is that the folks running the Storm botnet added a bit of logic that would profile visitors, and if a single visitor accessed too many of their sites within a particular time period they assumed that it was a security researcher and started DDoSing them—behavior activated by our spam crawler [4].

This attack caught our attention, as it did much of the research community. Indeed, I think it was this particular behavior of Storm that caused it to garner so much attention in the beginning. For a while we pursued a tangent, trying to use this behavior to measure a DDoS attack from the victim's vantage point. Consequently, with the permission of campus, we then restarted our crawler over a weekend and set up multiple packet monitors to get a full trace.

This project led to nothing, but had the serendipitous side effect of introducing us to Brandon Enright, then an undergrad working for the campus security group at UCSD. Brandon had become independently interested in Storm and had written code to crawl the Overnet Distributed Hash Table (DHT) that Storm used for coordinating its various bots. His goal was to enumerate all the IP addresses participating in the botnet at any particular time. A description of a later version of this work, and the challenges of such enumeration, later appeared in LEET [5]. Brandon was doing this to clean up Storm-infected bots at UCSD and sharing the data with others to do their own remediation, but he got our students interested in the details of how the botnet worked. Quickly a small group formed, with Brandon having the most hands-on malware experience, while Chris Kanich, Kirill Levchenko (a UCSD PhD student at the time), and Christian Kreibich (a researcher at ICSI) just started running instances of the Storm binary in a controlled environment and poking at it.

[RIK] I really liked that LEET paper. I asked Brandon and several other authors to write an article based on it for *;login:* [6].

[STEFAN] Let me give a tiny bit of background here to explain how this came to be. First, we'd had a close working relationship with Christian since 2006 when he collaborated with Kirill and Justin Ma (now a postdoc at Berkeley) on a system to automatically cluster packet traces by protocol (without a priori protocol knowledge). Christian was Vern's postdoc then, but Vern was completely open to him coming down for a couple of weeks to get this project done, and that pretty much set the stage for a silo-free group culture in which our various students and staff all feel free to work together (and tend to do so). Second, both the UCSD and the Berkeley groups had spent a bunch of time building malware containment systems—us with Potemkin [7], which was largely a research vehicle, while Weidong Cui (now at Microsoft Research) and Nicholas Weaver had built GQ [8]. Christian had then rewritten it, and that is what the group uses today. Moreover, GQ in turn benefited tremendously from Vern's investment in building Bro and related network analysis tools, so it was *reasonable* to automate the manipulation of network trace data (e.g., binpac [9], RolePlayer [10], etc.), which became important in the next year.

[RIK] I've often seen the names of a group of advisors and students from UCSD, UCB, University of Washington, and ICSI on related papers. I'm beginning to understand how this came about. It helps to see the bigger picture behind the names seen in papers, and how different researchers combine their strengths toward a common goal.

[STEFAN] Returning to the story, this little team got excited about understanding how Storm worked, but—aside from Brandon—they had basically zero skill doing reverse engineering. So not knowing that this was a crazy approach to pursue, they tried reverse engineering the command and control (C&C) protocol in a blackbox fashion—sending data at a captive bot, writing down what it did, theorizing about why it did those things, or letting it talk to its normal C&C and seeing what it tried to do in response to various commands it received. Brandon was busy, but provided key insights when they hit roadblocks (e.g., message encryption), but the rest was just raw guesswork over a period of several months. Vern and I had our doubts whether this was a good way for everyone to spend their time, since we weren't confident they could do it, or even what the research question would be if they succeeded. Geoff Voelker was on sabbatical in India for this period, so he was blissfully unaware of how much time was being wasted on this. However, we gave the students a long leash and somehow they pulled it off, documenting most of the C&C protocol and then building a set of parsers that could interpret it.

Once we realized how much information was contained—for example, how the spam messages were encoded within polymorphic templates, who the spam was being sent to, the delivery success rate, etc.—we realized we had a unique opportunity to look at how spam distribution worked from the standpoint of a botnet operator. We did a quick passive characterization of this data, which became the "On the Spam Campaign Trail" paper from LEET '08 [11]. As soon as we started writing it, however, Kirill pointed out that knowing how to parse Storm's C&C was also equivalent to being able to inject or change C&C commands. This would lead to our first real economics study. To give a bit more context, one needs to understand a bit about how Storm was structured circa late 2007–early 2008.

The basic Storm infrastructure was divided into three tiers: "worker bots," which were responsible for sending spam email or mounting DDoS attacks; "proxy bots," which provided public points of connection for worker bots; and master servers who provided commands to (and received feedback from) workers via the proxy tier. Workers and proxies were built out of compromised hosts and automatically differentiated based on whether they had external IP connectivity, allowing them to act as proxies versus workers. The master servers were dedicated machines in datacenters, such as Intercage, a California-based Web site hosting provider. Workers would select a quasi-random proxy using the Overnet DHT protocol and would then send, effectively, requests for work, which the proxy would then forward on to the master servers and similarly forward the responses of the master servers back to the workers. Proxies had some master server locations hardcoded and could received signed updates indicating the location of other such servers.

[RIK] I learned much of this by reading the LEET paper [5] and Brandon's *;login:* article [6].

[STEFAN] So, using a sample of the Storm malware, it was relatively easy to infect a machine and have it "become" a proxy and communicate with workers and master servers—just as a real infected host would (using our previous honeypot experience to carefully wall it off from accidentally sending email or DDoS attacks). Moreover, by building code to parse the messages as they went by, it was possible to actually change the responses being provided by the master servers in real time... in effect leaving the underlying process in place but manipulating one component. In particular, we could modify the URLs that the master servers provided to worker bots to be included in their outbound spam messages and have these point to sites under our control.

[RIK] I guess that this is the point where you needed to talk to lawyers?

[STEFAN] Yes, this is where we first started talking to our lawyer friends in depth. While the students were off making the capability a reality, we engaged with the people we knew who were best versed in Internet legal issues (e.g., the Computer Fraud and Abuse Act, the Electronic Communications Privacy Act, and the CAN-SPAM legislation) to help us figure out if we could actually do this. The first thing you find out when you start asking legal questions in this space is that no one can tell you "X is legal," nor is there any government agency who is authorized to certify such an effort. Even something as simple as sending ping packets to random hosts does not have "cut and dried" legality. Lawyers and legal scholars can frequently tell you whether something is clearly illegal, but if not, it's all about understanding the risk profile and working up the legal theory under which one operates. This took us quite a bit of time and we pursued multiple opinions to make sure there was agreement. In the end, while this area is rife with risk, the very specific circumstances around how Storm operated (e.g., being pull vs. push, using an existing DHT network, the kinds of information being sent, etc.) created a stage on which our advisors felt it was safe to proceed. Moreover, we developed a basic set of ethical principles to determine what could and couldn't be done in the study (based on consequentialism, the idea was that our intervention should defensibly cause no additional harm when compared to an alternate universe in which we had done nothing). This did indeed keep us from doing things that we had considered. For example, we had broken the private key for Storm's master server advertisements and we had the capability, in principle, to take over the entirety of the botnet.

Having addressed these issues, we dove into creating our experiment. We came to recognize that the most interesting questions revolved around the underlying economic model for spam: how many messages must a spammer send to get a sale; i.e., how often do people actually purchase? This determines the profitability of each spam message and implicitly drives the amount of spam being sent. Conversely, it also sets a lower bound that spam filters must reduce in order to make spam unprofitable.

It was Kirill Levchenko who first devised the pipeline metaphor that we would use in the paper [12], in which large numbers of messages are sent and then discarded at multiple filter tiers (e.g., rejected by mail servers, by spam filters, by mail readers, by site visitors who decide not to buy, etc.) until the final true purchases that monetize the entire activity get through. The basic experiment was simple: we'd change the URLs on the spam email templates that traversed our proxies and have them specify Web sites we controlled. We could then compare the number of spam messages each worker attempted to send with the number of visitors we received at the site. Further, if we duplicated the sites being advertised, we could further capture how often users tried to put particular items in their shopping carts and checked out. Since Storm was sending pharmaceutical spam (advertising for affiliates of the Glavmed "Canadian Pharmacy" program) we replicated their site in great detail. Then we started.

This is where we first started to get into trouble. First, we needed to acquire domain names to be used in this study. We simply bought a bunch from GoDaddy and started using them. This resulted in large numbers of complaints being directed to GoDaddy (since some subset of people receiving spam are technically sophisticated enough to identify the registrar of the site being advertised and motivated enough to send in their complaints), who in turn started suspending our domains and sending us various challenges/threats. We regrouped and found a different registrar whom we knew personally (really a reseller of Tucows), but this just added an additional layer in the chain of complainants. We briefly considered buying domains from ESTDomains (who at the time was a well known registrar used by criminal actors and who appeared to exert little oversight), but we decided this was a bridge too far for us. Instead, I had a surreal phone call with the fraud abuse group at Tucows to try to get their support. In trying to explain that domains we had registered were to appear in spam, but we were not sending the spam and that this was part of a research study, the first comment I received was, "You've got to be kidding me. This is the best story you could come up with?" However, after almost two hours of explanation, pointing them to past papers and mutual acquaintances to establish bona fides, the group over there realized that we weren't making it up. In the end, they thought it was pretty cool and agreed to allow us to proceed.

Our next problem was even more inadvertent. Storm also tried to infect hosts via social engineering ("Your friend sent you a card, click here to get it," sending you to a supposed eCard Web site that would provide an EXE containing the Storm binary). We also decided to replicate this using another replica site, but the binary we offered effectively did nothing (it simply reported that it had run, and even this behavior was automatically disabled if the date was later than our study period). Interestingly, AV signatures for our EXE soon appeared from most vendors (a clear indicator that the malware load had increased to a point that it was not possible to do any meaningful analysis on sample binaries). This was expected and, indeed, was ideal for our study since we wanted to—as much as possible—simulate the experience of Storm's operators (i.e., if our binary ran, it was in spite of AV and OS warnings not to do so, or indicated that users had no such security resources). However, this was the first time we had done something like this and we had not fully internalized that, in performing this infiltration, we were ourselves being monitored by others. And this is where things started to get squirrelly.

We had previously had contact with the FBI special agents in charge of investigating Storm and we had given them what insights we had. However, we had not thought to tell them that we were advancing our experiment to the next stage (i.e., changing links and setting up our replica sites). The consequence of this is that other investigators found our binary (originating out of UCSD) and concluded that we were potentially involved in working with the Storm operators. This in turn embarrassed our contact who had vouched for us, and now we looked like double agents. In the end, it was all resolved (indeed, at a meeting at the first LEET), and we learned an important lesson about communication, but we were told that, in the meantime, legal documents had been drawn up in anticipation of raiding the department's machine room and seizing our cluster.

There were other hiccups here and there, but by and large, the paper was a dream to write. In spite of its tremendous complexity, we made very few mistakes in the methodology. The only clear remaining issue was that we did not appreciate how quickly real spammers throw away spam-advertised domains (that then redirect to other sites) to mitigate the impact of blacklists. While we indeed used multiple different domains over time, ours were much longer lived, and thus blacklisting undoubtedly caused us to underestimate the response and conversion rate that the real spammers probably experienced. However, the broad results were quite clear: 75% of bot-originated spam was being immediately dropped on the floor, most of the remainder was filtered by spam filters, and only a very small fraction of users actually clicked on the links contained in such messages and an even smaller fraction ever decided to place an order. Yet in spite of this it was clear that the raw volume of this activity could produce significant revenue. This, in turn, would lead us to wonder about the composition of the spam value chain, who made the profit, which parts were weak, etc., but this was still some time away.

The next immediate concern was one of perception. Even though our paper was in submission (to CCS) and not public, many people seemed to have copies of it (people not on the PC) and more people still seemed to "know all about it." Indeed, one close colleague called me up from a conference and said, "I wanted to let you know that everyone is talking about your paper and a bunch of it isn't positive. Someone was talking to a group here and he says you guys are going to get the whole community in trouble."

Now, normally this isn't something we care much about. However, it was exacerbated by a contemporaneous factor. During this same period there was a big public to-do caused by Chris Soghoian's CNET blog entry [13] opining that the Colorado/Washington Tor exit node study in PETS constituted a breach of civil and criminal law and moreover represented a fundamental ethical violation because there had been no human subjects review. Now, while the Tor study issue was completely overblown (it was quickly resolved and no one was sued, arrested, or even censured), the underlying concern about oversight was real; it was clearly a wake up call to the security community about the human subjects issue. Indeed, little of the networking, systems, or security communities knew much about IRBs or even thought in those terms at the time. We were no exception. So Vern and I spent a bunch of time reading up, getting advice, and then writing a post-hoc human subjects proposal for our study with an explicit mea culpa to the IRB that we'd already done most of it and could we keep doing this study and keep the data. This took a very long time to get through the process (one of the challenges of a multi-university study), but ultimately all of our work and use of the data was approved without additional conditions. We also made a point to include an explicit section in our published paper on the underlying ethical issues and our justification for them—a practice that we continue to this day when the issues are non-obvious.

[RIK] Hmmm, this explains a lot about why I often hear you ask other researchers whether they bothered to get IRB approval. So what happened next?

[STEFAN] Ironically, in spite of our trepidation, we received little pushback from the community when the paper was published, and the work appears to have been widely appreciated. Indeed, part of what happened is that circumstances driven by other researchers eclipsed us, and while our work had once been "on the edge," it was now being highlighted by Marc Dacier in his CACM foreword for its "great care addressing the legal and ethical issues linked to the measurement."

For us, the immediate impact of the spamalytics study [12] is that it became *much* easier to get data from partners. In some sense, it was the reputation this work built with industry that planted the seeds that would support the next two years of activity.

[RIK] It seems like your Click Trajectories paper [14] at Security and Privacy in Oakland (2011) represents another chapter in this story.

[STEFAN] The "click trajectory" effort started a bit over two years ago (although the project name came much later). At that time we were starting to get quite a bit more spam data (10 distinct feeds at the peak from various anti-spam companies and honeypots), and Kirill Levchenko was tempted by the siren song of large-scale data mining. His view was that we should be able to cross-correlate all the data and create one of those TV movie FBI pictures with all the various participants linked by dependency arrows (in our case, botnets, spammers, fast flux clouds, registrars, affiliate programs, etc.)—the total picture of the spam ecosystem; who is responsible and where the weak points are in the business model. I think we were flush from the success on the spamalytics effort and really had no idea how much we were about to bite off.

The first big issue was how to collect additional data from our various spam feeds (sometimes millions of messages per day), including all the DNS data, registrar data, hosting data, Web page contents. Trying to bring back the old spamscatter infrastructure was a bust. It simply couldn't handle the load that we wanted to put on it and it was never designed for production use (nor did it record lots of the things we cared about). We also needed a place to put all these data that we could then make sense of. We decided to do everything from scratch.

At the core was the database. Kirill in particular had convinced us all that databases were good (all of the spamalytics work had been done using a database) which had a number of very cool side effects. First, it made certain questions very quick to answer (e.g., how many messages were sent to addresses of a particular form) and, as important, it made analyses easily repeatable. It has now become common for us to check in SQL query statements in our papers (as comments) along with the results. That way if we want to change something, we know exactly what the original query was and we can modify it without worrying if we're following the same methodology.

However, the data in spamalytics was modest by comparison. Moreover, for the click trajectory effort *everything* went through the database, because it was not only the store for final results, but also it was the trigger for additional measurements. We'd post-process raw spam emails and insert the links into "feed tables" which would be processed and then used to drive the various crawlers that would, in turn, put their results back into the database. We went through many versions of the database, killing mySQL and quickly going to Postgres, buying increasingly beefy hardware (the current core trajectory DB runs on 12 cores with 96 GB of memory, has multiple replicas, and manages a range of BLOBs in other servers, together comprising almost 100 TB of raw storage in total), and redesigning the database schema *many* times. Poor Kirill was constantly promising us that "things will be better in the next version of the schema." In the end, we needed to become very good at DB administration and optimization. UCSD PhD student Andreas Pitsillidis became that expert, through blood and sweat. In fact, about nine months ago, everyone else gave up trying to understand the full complexity of the DB system: only Andreas really gets it. While everyone did their part on this project (we had 15 authors on the final paper, all of whom made significant contributions), it was Andreas who ultimately made this all come together—I can't overstate the extent to which we could not have done this without him. Moreover, without the database (or equivalent technology) it would have been impossible to manage and process all the data we were collecting.

While the database was at the core, there were many moving pieces that fed it. First, the raw data feeds needed to be managed and normalized (and each of our data providers had their own favorite way of providing the data). Chris Kanich at UCSD became "feedmaster" (in addition to his other critical tasks) and dealt with the partners, created visualizations of the various feeds, and managed the ongoing relationships with feed providers.

The other source of feeds was from the GQ honeyfarm mentioned earlier. The honeyfarm ran network-neutered instances of major spam bots so that we could observe what spam they were being commanded to send. Keeping these bots going (and the honeyfarm itself) was a major endeavor. Christian Kreibich, with help from Chris Kanich, did most of this in the beginning, but eventually Chris Grier (then a new postdoc at Berkeley) took over the operational component (to everyone's relief), since it was also core to the next big project, his investigation of the pay-per-install market.

The other challenge here was to get the latest samples of new spam bots. Here we got help from lots of people, but in the end the go-to person was Brandon Enright (a long-time collaborator working for the campus networking and security organization at UCSD) who marshaled both his own private honeypot infrastructure and his considerable connections in the community to get whatever samples we needed. This gave us some "ground truth" about which botnets were advertising which URLs (allowing us to account for issues such as the Rustock botnet's spamming of random .com URLs to poison or overload blacklists).

After the raw feeds we had the crawlers. We had several implementations of a DNS crawler that would investigate each domain name we received and find its NS and A records. Over time, we learned that we needed to explore this space more completely to extract all the alternate answers being given due to fast-flux and CDNs (creating a name hosting "cloud" for each domain). Moreover, the load became large over time, and ultimately the crawler was rewritten from scratch by He "Lonnie" Liu (a first-year PhD student) to keep up. This particular artifact was remarkable because it is the only piece of infrastructure that we've built that "just worked." It never crashed, it never gave garbage data, it seemed to scale forever, and it was never the source of complaints from other members who depended on it. Lonnie never needed to say much at our weekly status meetings.

The Web crawler was a different story. I remember Geoff Voelker and I figured, "Hey, it's just crawling. How hard can it be?" We completely misunderstood the technical challenges in scaling up to large numbers of browsers and simulating associated humans. The poor recipient of our imperfect wisdom was Neha Chachra, also a first-year PhD student, who got handed the task of making a scalable Web crawler. She started by using an open source project called Selenium (designed to automate multiple Firefox instances) for the first version of the crawler, but we had no end of problems trying to get the features we wanted to work (grabbing raw page DOMs, screenshotting, inserting clicks, etc.) while synchronizing across large number of instances.

Ultimately, Neha wrote her own controller (with energetic help from Chris Grier for low-level Firefox-fu) that spawned and synchronized thousands of Firefox instances across a cluster of machines. Over time there were many changes to the crawler to handle various kinds of automated redirects, crash recovery, simulated user clicks, and so on (usually to deal with some crazy challenge that spammers had introduced). Even more significant, we discovered that many of the large hosting platforms used by spammers would blacklist our IP addresses if we visited too many times (ultimately blacklisting an entire /24). We acquired a broad range of diverse address space (Chris Grier put this together), and the Web crawler would schedule requests through proxies to these different blocks so we could see what a normal user would see. Neha went from implementing what we thought was a minor component of the system to becoming a central point of dependency for virtually everything (I'm sure she forgives us by now). Having so many parts, the crawler was constantly in revision; it was only recently that it became truly stable.

So, what to do with all these Web pages? Well, cluster them of course. The idea is to cluster all the URLs that lead to Web pages that are basically the same. However, the difference between "basically the same" and "exactly the same" hides quite a small nightmare. We tried quite a few different techniques. Early versions used a technique based on HTML structural features that Justin Ma came up with, and we experimented with SIFT and GIST-based visual features, but in the end we used a simple q-gram metric (how many sequences of length q are identical over some window) that worked incredibly well except for pages that were entirely based on images.

Clusters were useful for visualizing the data, and Andreas Pitsillidis created a great reporting interface that let us look at the relationship between particular groups of similar Web pages, their name server hosting, Web server hosting, the feeds we received them from, and so on. However, the real reason for clustering is that we operated under the assumption that if two pages look the same then they are probably part of the same "affiliate program," and this was key to our subsequent analysis.

Here it's worth taking a small digression to explain that modern spam is basically outsourced advertising. The spammers do not themselves sell any products but work on a commission basis for an affiliate program that handles payment processing, fulfillment, and customer service. Hosting of content and name services can be handled by the spammer or by the affiliate program, depending on circumstances (e.g., advertising based on search engine optimization, or SEO, is typically hosted by the program). Moreover, the actual spam delivery may itself be subcontracted from the spammer to a botnet operator, depending on the situation. However, these facts were not just assumptions. We spent quite a bit of time trolling around on underground forums trying to understand what we were dealing with. I did much of this work in the wee hours of the morning (as the group will attest from my random 2 a.m. ramblings about each new "discovery"), and Kirill would help when Google Translate barfed too badly on the Russian translation (many of the big programs are run by Russian speakers). Along the way we managed to acquire the "source code" for the e-shops from two of the largest pharma programs, Glavmed and RX-Promotion, which gave us ground truth about how different "storefronts" might all map to the same affiliate program. Moreover, via the broad underground marketplace, we were able to identify most of the other major programs. When we ran into a wall, Damon McCoy, a CIFellow postdoc, was the go-to person to hunt down a program.

This led to the development of another major element of the project: the tagger. The tagger is basically an oracle that looks at the HTML for a Web page and determines (1) what it is selling and (2) for which affiliate program. The first problem is easy, particularly because we don't care about false positives. We had decided to focus on pharma/herbal, luxury replicas (e.g., Rolex) and software—as these were the most spammed product categories (actually gambling and porn probably beat out software, but we had decided not to do either of those for institutional reasons), and we just checked to see if the Web page included any associated brand names (e.g., Viagra and Cialis for pharma, Rolex and Movado for replicas, and so on). This worked quite well; for example, the number of pharmacy pages we didn't classify as being in the pharma class was vanishingly small (typically these would be "image-only" redirect pages). However, classifying which program was advertising the page was quite a bit harder.

Tristan Halvorson, yet another first-year PhD student, got pressed into service generating regular expressions based on example pages I would find for each program. I'd gotten to the point where I could recognize most programs on sight, but Tristan had to somehow render this into code. So he'd try to capture what I was recognizing, then tag the whole corpus with affiliate names. I'd go look through it and find errors, and then we'd repeat. It's really hard to describe how much work this was. I looked at easily several tens of thousands of pages over the course of the project. I still remember Vern asking me late one night, "So how did we validate the tagger?" to which I replied, "Manually." He said, "Yeah, but really, that wouldn't scale." He was right in principle, but in practice Tristan and I (with Geoff lending a hand) just spent days at it—scaling be damned. This is not an approach we'll repeat again, however. We've had another student build a supervised machine learning tool to do this that seems to do almost as well with much less effort, so hopefully that's the future.

The last big component was purchasing. We really wanted to do the end-to-end analysis—where the spam came from to where it was fully monetized, and this meant purchasing goods and receiving them. This created a whole host of problems. First, we needed the university to permit it. You can imagine the conversations: "We need to make credit card purchases from criminals for goods that we may not get. Oh, and it's entirely possible that there will be fraud directed against these cards." I still remember questions like, "Why can't you just use a purchase order?" This took at least a year of education, negotiation, explanation, documentation, pleading, and much passing of the buck before we worked it all out. The purchase phase involved huge amounts of oversight, including by our own lawyers, university general counsel, and the systemwide office for research compliance. Finally, however, a few key people at UCSD (and, perhaps more importantly, at the UC Office of the President) came through for us and gave us the approval we needed.

The next problem was where to get these credit cards. Prepaid gift cards seemed like the ideal instrument. They get processed exactly like Visa and Master Card, and you can purchase them on demand, in bulk. Plus, you can set the name and home address as you like. It was too good to be true, unfortunately. First, most of these cards had no way to get a statement: Was a charge placed on the card, for how much, and who did they claim to be? Instead, they had phone support, where you could call in to get information. We did find a small number of such cards that had an online Web statement interface and so we placed an order for a few thousand dollars' worth of these. However, we discovered that the statement didn't include the Acquirers Reference Number (this is the 23-digit number you may find on your personal credit card statement) which identifies the Bank Identification Number (BIN) for the bank acting on behalf of the merchant in the credit card transaction. Without this we wouldn't know what bank was being used and we'd need to trust the information in the merchant identification string (which is routinely false, in our experience). We tried calling in to get this information, but it was very slow going, in part because the call center was staffed by only a few people and they grew suspicious at the large numbers of calls they kept getting from us.

Using our personal networks in the security community, we did manage to find investigators we knew who had done some similar work, and they identified for us the one card that had all the properties we desired: a Web interface and online access to the ARN number for each transaction. Ironically, it was the store brand at the Ralph's supermarket near us. It was perfect.

Then the Credit Card Reform Act passed. As part of this, the Department of the Treasury instituted a rule requiring suspicious transaction reporting on foreign transactions for prepaid cards (precisely because such cards are perfect for money laundering). The added reporting overhead made most providers just stop offering international transactions (go read the fine print on the pre-paid gift cards at your local supermarket), which included the bank sponsoring Ralph's cards. Sigh. At this point I gave up and decided we'd simply have to do without.

Thankfully, Chris Kanich hadn't given up hope. On his own initiative, he started cold-calling credit card issuers explaining the service we needed. Amazingly, he found a company who was game to help us and then negotiated a contract. One day Chris came in and said, "I think I got the credit cards." It turned out to be a spectacular resource: for a modest fee, new credit cards were created on demand including detailed information (the BIN, the card acceptor ID, the country code, and so on, far more than we ever hoped to get) on each authorization or settlement transaction of interest. Over the course of our studies, Chris and Damon ran our purchasing operation, using hundreds of different cards, email accounts, and a bevy of Google voice phone numbers that redirected to a few "burner" phones they each carried.

Surprisingly, getting these orders to properly clear was non-trivial and we had to reverse engineer components of their fraud detection system (e.g., using co-located IP addresses to source purchases, non-free emails, etc.), plus Chris and Damon needed to handle a constant stream of follow-up confirmation calls from the affiliate program's customer service arms. On top of that, managing all the raw credit card transaction data and keeping it in sync with the associated Web site data was a major time sink. Here we made the mistake of trying to make due with a large Google Docs spreadsheet, a decision we're still paying for.

These were the major pieces, but there were countless details I skipped in this description: for example, Mark Felegyhazi's whois crawler and the cross-DNS matching work that Nick Weaver did in the 11th hour.

I also skipped an adequate description of all of our failures. First, we failed repeatedly to wrap our minds around this paper. We had at least two aborted attempts to submit a paper only to discover that we still didn't really understand what we were doing. I know that Vern, Geoff, and I all had doubts if this thing would ever come together (18 months of work without anything to show can shake even the most confident person). We tried, but ended up failing, to incorporate a strong analysis of the spam delivery component (which programs were advertised by which botnets, which used Webmail, etc.), and we spent months building complex models for inferring the different individual affiliates of different program,s ultimately to discard them for the final paper. There is at least another paper's worth of work in all the stuff that we left on the "cutting room floor," but we chose to focus on the parts we were the most confident about.

For the paper submission there were a few major turning points. One was a meeting where we came up with the conceptual model of the spam value chain as comprising advertising (spam delivery), click support (translating a recipient's click into a Web site), and realization (payment processing and fulfillment). This model, beautifully illustrated by Christian in the paper [14], gave us a way to focus on the problem. It also led to us choosing to focus our analysis on the challenges of intervening at any given place in the value chain (this had always been a goal, but originally just one among many). The other major event is when the credit card data first started coming in and we realized that there were really only a handful of banks involved in processing money for spam-advertised programs. We'd hypothesized that this might be true, but with the data in hand we knew we had a great story. Finally, in the week before submitting the paper, most of the ICSI folks came down to UCSD and everyone pushed hard to get everything done. That was an amazing time and huge amounts of work got done with everyone pitching in. This is also one of those papers where the final paper actually differs in non-trivial ways from the submission. We used the time we had to really tie up loose ends and polish the analysis. I think we all knew that this was going to be one of our important papers and everyone put in the time to make it crisp.

It also kicked up a half-dozen other projects that we're working on as we speak, including several papers to appear at CSET '11 [15] and USENIX Security '11 [16].

The one 10,000-foot thing that I really hope comes out is that our core approach is to try to understand these issues from the standpoint of the attacker rather than simply from the standpoint of the victim. I think we frequently hamstring ourselves in the security community with the notion that the adversary is some abstract and arbitrary entity, whereas frequently the adversary is concrete and has very specific goals. Understanding these goals (particularly those focused on profit-making) then lets us consider defense as a form of offense: What security investments can I make that will maximally undermine the adversary's goals? Absent this kind of analysis we end up just blindly treating random symptoms of the problem, rather than focusing on the core drivers.

References

[1] Jason Franklin, Adrian Perrig, Vern Paxson, and Stefan Savage, "An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants," *Proceedings of the ACM Conference on Computer and Communications Security*, Alexandria, VA, October 2007: http://www.icsi.berkeley.edu/pubs/networking/miscreant-wealth.ccs07.pdf.

[2] David S. Anderson, Chris Fleizach, Stefan Savage, and Geoffrey M. Voelker, "Spamscatter: Characterizing Internet Scam Hosting Infrastructure," *Proceedings of the 16th USENIX Security Symposium*, August 2007: http://www.usenix.org/events/sec07/tech/full_papers/anderson/anderson.pdf.

[3] Rob Thomas and Jerry Martin, "The Underground Economy: Priceless," *;login:*, vol. 31, no. 6, December 2006: http://www.usenix.org/publications/login/2006-12/openpdfs/cymru.pdf.

[4] Don Jackson, "Analysis of Storm Worm DDoS Traffic," Sept. 11, 2007: http://www.secureworks.com/research/blog/index.php/2007/09/12/analysis-of-storm-worm-ddos-traffic/.

[5] Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage, "The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff," First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08), April 2008: http://www.usenix.org/event/leet08/tech/full_papers/kanich/kanich.pdf.

[6] Brandon Enright, Geoff Voelker, Stefan Savage, Chris Kanich, and Kirill Levchenko, “Storm: When Researchers Collide,” *;login:*, vol. 33, no. 4, August 2008: http://www.usenix.org/publications/login/2008-08/openpdfs/enright.pdf.

[7] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage, “Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm,” *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October 2005: http://cseweb.ucsd.edu/~savage/papers/Sosp05.pdf.

[8] Weidong Cui, Vern Paxson, and Nicholas Weaver, “GQ: Realizing a System to Catch Worms in a Quarter Million Places,” ICSI technical report TR-06-004, September 2006: http://www.icir.org/vern/papers/gq-techreport.pdf.

[9] R. Pang, V. Paxson, R. Somer, and L. Peterson, “binpac: A yacc for Writing Application Protocol Parsers,” *Proceedings of the 2006 Internet Measurement Conference*, October 2006: http://conferences.sigcomm.org/imc/2006/papers/p29-pang.pdf.

[10] W. Cui, V. Paxson, N.C. Weaver, and R.H. Katz, “Protocol-Independent Adaptive Replay of Application Dialog,” *Proceedings of the 13th Symposium on Network and Distributed System Security (NDSS 2006)*, February 2006: http://research.microsoft.com/en-us/um/people/wdcui/papers/roleplayer-ndss06.pdf.

[11] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage, “On the Spam Campaign Trail,” First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET ’08): http://www.usenix.org/events/leet08/tech/full_papers/kreibich/kreibich_html/.

[12] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage, “Spamalytics: An Empirical Analysis of Spam Marketing Conversion,” *Proceedings of the ACM Conference on Computer and Communications Security,* Alexandria, VA, October 2008: http://www.cs.ucsd.edu/~savage/papers/CCS08Conversion.pdf.

[13] Chris Soghoian, CNET, “Researchers Could Face Legal Action for Network Sniffing,” July 24, 2008: http://news.cnet.com/8301-13739_3-9997273-46.html.

[14] Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Mark Felegyhazi, Chris Grier, Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage, “Click Trajectories: End-to-End Analysis of the Spam Value Chain,” *Proceedings of the IEEE Symposium on Security and Privacy*, May 2011: http://cseweb.ucsd.edu/~savage/papers/Oakland11.pdf.

[15] Chris Kanich, Neha Chachra, Damon McCoy, Chris Grier, David Wang, Marti Motoyama, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker, “No Plan Survives Contact: Experience with Cybercrime Measurement,” Fourth Workshop on Cyber Security Experimentation and Test (CSET ’11), USENIX, August 8, 2011.

[16] Chris Kanich, Nicholas Weaver, Damon McCoy, Tristan Halvorson, Christian Kreibich, Kirill Levchenko, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage, “Show Me the Money: Characterizing Spam-Advertised Revenue,” *Proceedings of the 20th USENIX Security Symposium,* August 8–12, 2011: http://www.usenix.org/events/sec11/tech/full_papers/security11_proceedings.pdf.

Understanding Advanced Persistent Threats

A Case Study

NED MORAN

Ned Moran has worked in cyber threat intelligence analysis for nearly a decade and a half. He currently serves as the Director of Technical Research at a cyber risk management company. He has also served as an Adjunct Professor of Information Privacy and Security at Georgetown University since 2008. He has been an invited speaker at NATO's Cooperative Cyber Defense Center of Excellence's Conference on Cyber Warfare in Tallinn, Estonia, and at the RSA Conference in San Francisco. He is frequently interviewed by the media concerning the intersection of terrorism and technology.
ned.moran@gmail.com

APT, short for Advanced Persistent Threat, is a commonly used and controversial term bandied about the IT security sector. Many feel that this term is abused and simply used to describe attacks that network defenders failed to prevent—no matter the sophistication of the attack. This article seeks to establish a working definition for APT and to highlight that the sophisticated nature of these attacks lies not within the technology used but, rather, the logistical organization of the adversary. This article will offer an in-depth examination of an APT-style attack as a means of highlighting the operational efficiency of the adversary.

What Is APT?

It is first necessary to establish an accepted definition of APT. Richard Bejtlich, the Chief Security Officer at Mandiant and long-term observer of APT-style intrusions, defines APT as follows [1]:

> **Advanced** means the adversary can operate in the full spectrum of computer intrusion. They can use the most pedestrian publicly available exploit against a well-known vulnerability, or they can elevate their game to research new vulnerabilities and develop custom exploits, depending on the target's posture.
>
> **Persistent** means the adversary is formally tasked to accomplish a mission. They are not opportunistic intruders. Like an intelligence unit, they receive directives and work to satisfy their masters. Persistent does not necessarily mean they need to constantly execute malicious code on victim computers. Rather, they maintain the level of interaction needed to execute their objectives.
>
> **Threat** means the adversary is not a piece of mindless code. This point is crucial. Some people throw around the term "threat" with reference to malware. If malware had no human attached to it (someone to control the victim, read the stolen data, etc.), then most malware would be of little concern (as long as it didn't degrade or deny data). Rather, the adversary here is a threat because it is organized and funded and motivated. Some people speak of multiple "groups" consisting of dedicated "crews" with various missions.

This definition of APT is extremely useful, because it does not focus on the technical sophistication of the adversary or the elegance of the attack code used. Rather, it focuses on the organizational capabilities and intentions of the adversary.

Misunderstanding the A in APT

Thinking about an APT in light of the adversary organization's capabilities and intentions highlights that the "advanced" in APT is less about code and more about techniques, tactics, and procedures. Just as the power of the US military lies largely in its ability to organize and secure a logistics supply chain to its frontline troops, APT actors are able to provide robust support to the frontline intrusion operators—the guys at the keyboard.

This type of organizational structure and efficiency is what truly defines APT. True APT actors are part of a robust organizational infrastructure driven by specific collection requirements that focus these actors onto a set of targets. Further, the ongoing and strategic nature of these collection requirements forces APT actors to develop tools and tradecraft that enable them to fully exploit new collection opportunities—e.g., an infected drone in a targeted organization.

APT actors support their front-line intrusion operators with tools that notify the operators of new infections as well as tools that enable the operators to quickly leverage these initial footholds into a deeper and more resilient presence inside the targeted organization.

What Motivates APT Actors?

APT actors are primarily interested in maintaining reliable access to their targets. The desire for access to sensitive intellectual property drives this need for persistent and reliable access. APT actors are not interested in quick smash and grab attacks. Instead, they want to quietly get inside a target environment and set up a number of redundant listening posts, so that if any one infection is detected the other infections will still provide the adversary with the required access into the targeted organization.

Persistence, the P in APT, also means that the adversary keeps coming back. They will consistently attack the same target over and over again in an effort to maintain a secure foothold within a targeted organization. As infections are discovered and remediated by the victim organization, the adversary will launch a new salvo designed to regain a foothold. The purpose of this foothold is to enable the type of ongoing monitoring required to deliver a complete picture of an organization's internal communications and access to intellectual property.

A Closer Look at an APT-Style Attack

On April 12, 2011, a spear phishing email sent to specific targets was observed in the wild. This observed spear phishing attack provides a good example of an APT-style attack. The targets of this spear phish were specifically chosen by the attackers because the attackers perceived that these targets held information of value.

The spear phishing email contained a Microsoft Word document attachment. This Word document was crafted to appear as a legitimate document and contain specific subject matter of interest to the targeted victim. This type of social engineering is a common tactic in an APT style attack. The adversary conducts detailed pre-attack reconnaissance in an effort to better understand the victims. This reconnaissance enables the adversary to design spear phishing lures that entice the victim into opening the malicious attachment.

Embedded within this document was a malicious Flash file designed to exploit the recently announced Adobe zero-day CVE-2011-0611.

The exploit first makes use of a heap spray to fill memory with 0x11111111 and then loads a second SWF file. It is this secondary SWF that actually triggers the vulnerability. The SWF file makes use of several common obfuscation techniques. The code attempts to confuse disassemblers by setting the size of a group of constants to 0x15 when there really are 0x14 present, causing disassembly to be misaligned with the actual code.

In addition, it does several things which are also fairly usual. For example, streams of instructions which are effectively dead code, conditional branches which can never be taken, jumping around unnecessarily, and blocks of instructions which have no effect on the program itself. All of this isn't really a factor in the exploit itself but is simply obfuscation.

The shellcode executed by this exploit drops a malicious payload with the following properties:

```
File: scvhost.exe
Size: 22016
MD5: 4EC6D3A6B5A5B67D4AB5F04C41BFB752
```

Scvhost.exe is installed and launched with the filename msdtc.exe.

The msdtc.exe payload initiated traffic over port 80 with a command and control server at msejake.7766.org. 7766.org is a dynamic DNS provider that allows domain administrators to quickly and easily point their domain to any IP under their control. During the observed attack the domain resolved to 125.46.42.221. Infected victims were observed sending base64 encoded messages to the control server at msejake.7766.org. Observed traffic was as follows:

```
bG9nb258U1lTVEVNLTEyMzQ1Njc4OXxXaW5kb3dzIFhQfDEwMDcwN3
wzY2I4ZGM5MjI4N2UzZmJmMTA0MmQ2NTRlYzRkY2RhMnw=
YWN0aXZlfA==
```

This traffic decodes to:

```
logon|SYSTEM-DDBLV6BQXN|Windows XP|100707|3cb8dc92287e3fbf1042
d654ec4dcda2|
active|
```

This traffic appears to serve an initial reconnaissance function whereby the infected machine reports back to the control server basic system information including machine name, operating system, and state.

Static analysis of the msdtc.exe reveals a number of strings of interest. These strings include but are not limited to:

```
shell|
filelist|
upload|
```

These are likely commands the trojan is designed to recognize and act upon. "Shell" is likely a command that can be issued by a remote hostile actor to open a shell on the compromised machine. "Filelist" is likely a command that can be issued by a remote hostile actor to enumerate a list of files within a given directory. Finally, "upload" is likely a command that can be issued by a remote hostile actor to exfiltrate information from the victimized machine. For a full list of commands identified via static analysis of the msdtc.exe payload, please see the Appendix.

It is likely that this trojan was designed to enable a remote operator to fully reconnoiter a victim's machine, search for and acquire deeper access within the targeted network, and exfiltrate sensitive information.

In laboratory testing, the intrusion operator was observed establishing active sessions on an infected machine. As noted above, the infected machine beaconed an "active" message to the command and control server—likely informing the controller that the infected machine was available for exploitation. Within 49 minutes of initial exploitation, an intrusion operator initiated multiple sessions on the infected machine. An analysis of the sessions indicates that a live person at a keyboard (as opposed to a scripted service or program) initiated the connection to the back-doored computer.

The malicious operator maintained three different sessions on the infected machine. One served as a command and control session. A second session was used to gather intelligence about the networking infrastructure surrounding the victim. A third session was used to enumerate the hard disk likely in search of sensitive information.

The following table illustrates the commands passed by the operator during these sessions. The first command of "SHELL |" was likely issued to open the back door on the infected computer. The table below shows the number of seconds elapsed between the issued commands and the first "SHELL |" command.

Commands in **bold** were part of a command and control session, and those in *italics* were from a session established to reconnoiter the network of the infected victim. APT actors typically establish a beachhead on the infected machine and then immediately look for additional opportunities to establish additional footholds deep within the penetrated network.

Commands in regular type were part of a session established to scan the hard drive of the infected victim. APT actors also typically scan compromised hosts for sensitive intellectual property.

TIME (in seconds)	**COMMAND**
0	**SHELL |**
2	**SHELL START|**
7	*COMMAND|NET USER*
13	*COMMAND|IP CONFIG /ALL*
62	*COMMAND|NET VIEW/DOMAIN*
70	FILES|
72	FILELIST|C:\*.*
74	FILELIST|C:\DOCUMENTS AND SETTINGS*.*

78	FILELIST|C:\DOCUMENTS AND SETTINGS\ADMINISTRATOR\*.*
70	FILES|
72	FILELIST|C:\*.*
91	*STOP|*
97	FILELIST|C:\DOCUMENTS AND SETTINGS\ADMINISTRATOR\ MY DOCUMENTS*.*
101	FILELIST|C:\DOCUMENTS AND SETTINGS\ADMINISTRATOR\DESKTOP\*.*
125	FILELIST|C:\DOCUMENTS AND SETTINGS\ADMINISTRATOR\DESKTOP\ SERVICE PACKS*.*
140	STOP|
144	**UNINSTALL**

Figure 1: Decoded commands issued by intrusion operator

These commands were likely issued by the remote attacker via a management dashboard that enables the attacker to select a number of pre-configured scripted actions. Every command issued by the attacker was base64 encoded. The time delays between these commands can be explained by a human operator evaluating the responses from the infected victim prior to deciding which commands to issue next.

This is the type of organizational sophistication that defines the A in APT. The particular trojan used in this attack was not in and of itself sophisticated. No rootkits were used, and a knowledgeable sysadmin likely would have detected the infection via host or network-based analysis.

However, the efficiency of the adversary, as demonstrated by their actions during this observed attack, highlights what the "advanced" in APT is all about. First, the attack was targeted, indicating that the adversary was only interested in penetrating a set of victims they were interested in monitoring.

Second, the speed with which a human operator established a session on the infected host demonstrates the advanced logistics capabilities of the adversary. The adversary established a session within 49 minutes of the initial infection and then proceeded to iterate the infected machine, presumably in search of sensitive information or for connections deeper within the targeted network. This demonstrates that the adversary had "trained" operators on standby ready to exploit newly compromised computers. This type of operation requires resources that only a determined adversary would support.

Third, the operator's decision to quietly uninstall the trojan from the infected machine after two minutes and 22 seconds of reconnaissance indicates that the operator knew exactly what he or she was looking for. This demonstrates that this type of APT attack was driven by a set of requirements, and when the operator of this intrusion was unable to locate data meeting those requirements, he or she decided to quickly retreat and exploit another newly infected computer.

Conclusion

The above example demonstrates that the threat from APT actors is not their embrace of novel exploit code or sophisticated attack tools, but, rather, their organizational efficiency. These actors patiently study their targets, craft enticing spear phishing attacks, and quickly exploit new victims. They will continually seek to maintain secure footholds within their targets, as this is the most efficient way for them to routinely exfiltrate the sensitive data they are tasked with acquiring.

Reference

[1] http://taosecurity.blogspot.com/2010/01/what-is-apt-and-what-does-it-want .html.

Appendix: Strings Pulled from msdtc.exe

```
CmD
shelldata|
shell|
shellstart
command
stop
upload|
upload
files|
driver
driver|
filelist
listerror|
filelist|
delete
run
renamefile
stop
down|
dirlist
direrror|
dirlist|
logon|
shell
active
files
upload
closeos
restart
down
dclose
uclose
uninstall
active|
```

Network Attack Collaboration

Sharing the Shell

RAPHAEL MUDGE

Raphael Mudge is a Washington, DC, based code hacker. His current work is the Armitage GUI for Metasploit. His past projects include the After the Deadline proofreading software service and the Sleep scripting language. Raphael has worked as a security researcher, software engineer, penetration tester, and system administrator. Raphael holds a commission in the Air National Guard.
rsmudge@gmail.com

Working in a network attack team today is cumbersome. Penetration-testing tools such as Core Impact, Immunity Canvas, and Metasploit assume a single user. Team members have limited means to share access to compromised hosts, and good intentions are quickly mired in a disorganized free-for-all.

To address this problem I developed Armitage, a technology that allows a network attack team to communicate in real time, share data, and seamlessly share access to hosts compromised by the Metasploit exploitation framework. This article discusses the needs for network attack collaboration, the inner workings of the solutions in Armitage, and the lessons learned using this technology with the 2011 Northeast and Mid-Atlantic Collegiate Cyber Defense Competition red teams.

Metasploit

Metasploit [1] is a popular exploit development framework. H.D. Moore started the project in 2003. Metasploit makes it easy for security researchers to develop exploits for software flaws and use them in the context of a very feature-rich tool.

Metasploit features include multiple user interfaces, support for multiple platforms, and powerful post-exploitation tools. Metasploit users may choose which payload to execute when an exploit is successful. Metasploit payloads range from simple command shells to powerful post-exploitation tools like Meterpreter.

Through Meterpreter, users may transfer files, execute and interact with processes, dump the Windows SAM database, navigate the file system, and manage processes on the compromised host. When delivered with an exploit, Meterpreter is capable of running completely from RAM without ever touching disk. Metasploit provides a command-line interface for interacting with Meterpreter.

Meterpreter is not a Metasploit-only concept. Other exploitation tools, such as Immunity Canvas and Core Impact, have built-in post-exploitation agents too. The *Shellcoder's Handbook* [2] explains this practice. Exploitation tools that simply provide a shell lose the ability to transfer files, give up access to the Win32 API, and in some cases lose access to any privileged tokens the current thread might hold. Post-exploitation agents, such as Meterpreter, implement a protocol that allows users to carry out these and other actions. The session sharing ideas presented in this article should apply to these other post-exploitation agents.

Armitage

Armitage [3] is the graphical user interface I wrote to support teams using Metasploit. Armitage organizes Metasploit's features around the network attack process. There are features for host discovery, exploitation, post-exploitation, and maneuver.

Armitage exposes Metasploit's host management features. It's possible to import hosts and launch scans to populate Metasploit's database with target and service information. Armitage's user interface also displays the target database in a table or graph format.

Armitage's *find attacks* feature recommends remote exploits using known host and service information. Users may also launch browser exploits, generate malicious files, and create executable files to call back to Metasploit from Armitage.

Armitage provides several post-exploitation tools for Windows targets built on the capabilities of Metasploit's Meterpreter agent. Menus are available to escalate privileges, dump password hashes to a local credentials database, browse the file system, and open command shells. For Linux and Mac OS X targets, Armitage lets users interact with a command shell.

Finally, Armitage aids the process of setting up pivots, a Meterpreter capability that lets users exploit a compromised host to attack and scan other hosts. Armitage also exposes Metasploit's SOCKS proxy module, which turns Metasploit into a proxy server that routes outgoing connections through existing pivots. With these tools, users may further explore and move through a target network.

Figure 1 shows the Armitage user interface. Armitage's targets panel visualizes known hosts, active sessions, and existing pivots. A session is an active Meterpreter agent or a shell on a compromised host. The module browser in the top left lets users search for and launch Metasploit modules. These GUI components are always visible. Armitage uses tabs to organize open consoles, command shells, and browsers.

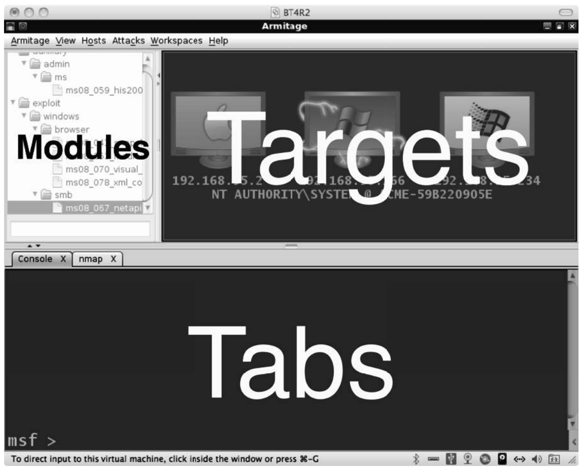

BT4R2
Armitage
Armitage View Hosts Attacks Workspaces Help
admin
ms
ms08_059_his200
exploit
windows
browser
Modules
ms08_070_visual_
ms08_078_xml_co
smb
ms08_067_netapi
Targets
NT AUTHORITY\SYSTEM @ ...CME-59B220905E
Console X
nmap X
Tabs
msf >
To direct input to this virtual machine, click inside the window or press ⌘-G

Figure 1: Armitage user interface

Teaming Architecture

Through Armitage, it's possible to manage and share a remote Metasploit instance via its RPC server. Metasploit's RPC server allows clients to send commands to Metasploit using an XML-based protocol.

Armitage extends Metasploit's RPC interface to provide real-time communication, data sharing, and session sharing through a deconfliction server. The deconfliction server offers Armitage clients additional functionality, helps with scalability, and manages multiple clients accessing Meterpreter and shell sessions.

The deconfliction server is part of Armitage. It's started using the `–server` command line option. The deconfliction server connects to Metasploit like any other client. When it connects, it sets a global variable in Metasploit to instruct Armitage clients to connect to it. Adding features through a separate server protects the teaming features from internal changes to the Metasploit framework.

Some Metasploit features require the client to modify or read a local file. Metasploit's RPC server does not offer an API for reading and writing local files. For these cases, the deconfliction server offers the missing functionality. The extra functions in the deconfliction server allow Armitage to offer real-time communication to team members, lock and unlock shell sessions, and transparently download screenshots taken through Meterpreter.

The deconfliction server also helps Armitage with team scalability. Armitage clients used to poll Metasploit to get the list of current hosts, sessions, and known services. Constant polling from multiple clients caused Metasploit to stop responding with more than five active clients. The deconfliction server temporarily caches the output of some commands to reduce load on the Metasploit RPC server. Armitage is now able to support a team of ten or more clients.

The deconfliction server's primary purpose is to act as a proxy between Armitage clients and Metasploit for session interaction. All session read and write commands go through the deconfliction server. The deconfliction server manages these operations using Meterpreter multiplexing to provide transparent session sharing for the user.

Figure 2 shows the relationship between the Armitage clients, the Metasploit RPC server, and the deconfliction server. The dashed lines show the communication path for Meterpreter commands. The solid lines show the path for most Metasploit commands.

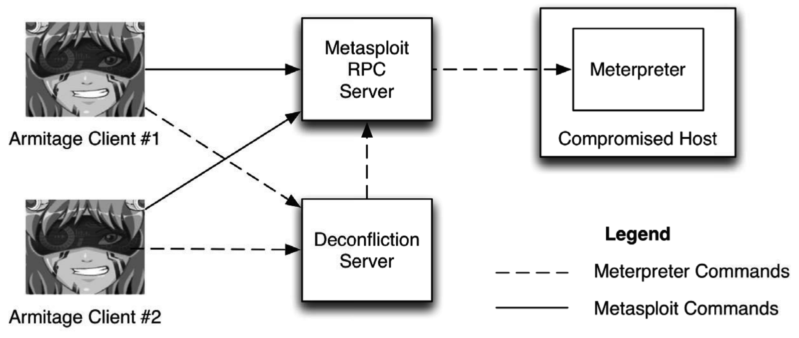


Figure 2: Teaming architecture

Real-Time Communication

In a collaboration situation, it helps to have real-time communication. Red teams often rely on Internet relay chat, instant messaging, a shared wiki, or even yelling across a room. Armitage's deconfliction server offers Armitage clients read and write access to a shared event log.

Armitage presents this shared event log as a new tab. Users may search it and type messages into it as if they're using a chat room. Armitage prefixes a user-provided nickname to each message. The Armitage client also reports events to this shared log. These events include scans, exploits, login attempts, changes to the pivot configuration, and clearing the database.

In practice, I haven't seen the Armitage event log overtake other real-time communication methods. However, the event log is useful for attributing damaging actions to team members. At one event, a team member launched a mass automated exploitation attack from a shared server meant for post-exploitation only. Another team member accidentally cleared all of the hosts in the Metasploit database. The event log helped us identify which team members to counsel.

Data Sharing

Network attack teams generate and capture a lot of data during an engagement. This data includes port scans, vulnerability scans, encrypted passwords, working credentials, and other captured artifacts. Making this data available so that the whole team can locate it and work with it is difficult. Teams often rely on a version control system, an ftp server, or a wiki to store this information. The Dradis Framework [4] is an example of a specialized project to help attack teams organize their data and make it available to the whole team.

Ryan Linn examined these sharing options and presented another alternative in his Multiplayer Metasploit [5] work. Mr. Linn observed that wikis and other non-attack-specific storage mediums suffer from arbitrary organization. Dradis is a good alternative but it's hard to take action on the data from Dradis. His alternative idea is to use Metasploit as a data repository. Metasploit has several database tables to store credentials, encrypted passwords, known services, and data taken from hosts by automated post-exploitation scripts. Mr. Linn modified the Metasploit Framework to expose this data through Metasploit's RPC interface.

Armitage builds on Ryan Linn's work by using Metasploit for data sharing. Armitage's targets view shows the current hosts and active sessions. Any team member may right-click a host and select Services to see the known services and any banner information associated with the service.

In practice, Armitage's data-sharing features provided shared situational awareness and easy access to data automatically stored by Metasploit. Shared access to the Metasploit credentials table proved valuable in many situations. Each team member had access to all successful credentials when attempting to log in to a service. This data sharing also allowed any team member to export stored password hashes and attempt to crack them.

The Access Sharing Problem

In a team situation, the person who gets access to a host is usually the one who handles post-exploitation. This happens because it is difficult to share access with other team members. This limitation forces teams to organize themselves by target types. For example, team members who are Windows experts attack Windows systems. This is a limiting tactic, as it is still hard for task specialization to occur. The Windows expert can't delegate post-exploitation tasks to one or more team members.

One possible solution to the access sharing problem is session passing. The multi_meter_inject script in Metasploit generates a Meterpreter executable bound to a callback host and port, uploads it to a host, and executes it on the compromised host.

Session passing is a threat to the network attack team's stealth. The uploaded Meterpreter executable may trigger the local antivirus or other personal protection product. More connections may help the system administrator determine that the host is compromised. In some situations, session passing is not practical. Sometimes it is difficult to pass a session for an available host, because it would require pivoting connections through another compromised host.

Session Sharing

An ideal solution to the access sharing problem is session sharing. Network attack teams would benefit from putting all successful compromises into a shared pool for any other member to use. Session sharing has a stealth advantage. It does not require creating a new access by uploading and executing a new program. Session sharing also allows all actions to occur through one communication channel. A system administrator cannot know if one person or five people are working on their host. Session sharing allows team members to benefit from each other's work. Session sharing also allows specialization of tasks. Some team members may focus on getting access to hosts, others may focus on persistence, and the rest may focus on post-exploitation.

For Meterpreter sessions, Armitage implements session sharing using "meterpreter multiplexing" described in the next section. With meterpreter multiplexing, multiple team members are able to simultaneously use a session. This solution creates an illusion that the access is not shared.

For shell sessions, Armitage limits access to the session to one team member at a time. Opening the session sends a request to the deconfliction server to see who owns it. If the session is in use, Armitage notifies the team member that the session is in use. If the session is not in use, Armitage locks it and lets the user interact with it. When the user closes the tab, Armitage notifies the deconfliction server that the session is available again.

Session sharing is the most useful teaming capability in Armitage. At all events I participated in, session sharing allowed team roles to emerge organically. Team members gravitated toward tasks they were most comfortable with. This is different from my previous exercise experiences, where team members who couldn't compromise hosts had limited participation opportunities.

Meterpreter Multiplexing

Meterpreter multiplexing is the Armitage feature that allows multiple clients to share one session. Armitage adds every meterpreter command to a queue specific to that session. A separate thread executes these commands in a first-in first-out way. When Armitage executes a Meterpreter command, it reads output until the command is complete. This output is then sent to the command requestor using the identifier stored with the command.

Armitage uses a heuristic to decide when a command is complete. The simplest heuristic is to read from a session until a read returns an empty string. Some commands return an empty string before they're finished. For these commands, Armitage expects a set number of empty reads to consider the command completed. For all commands, Armitage has a 12-second timeout. This timeout prevents a failed command from making the session non-responsive.

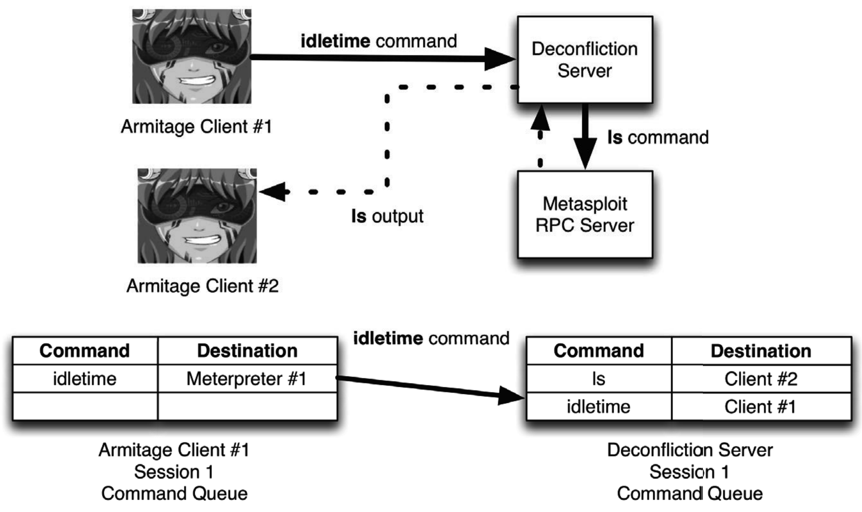
Some Meterpreter scripts execute in the background and report their output later. These scripts create a problem for the command-multiplexing scheme. It's possible for the output of a script to mix in with the output of another command. Armitage mitigates this by reading from Meterpreter before executing a command. When used with a local Metasploit instance, Armitage displays stray output in any Meterpreter tab.

The deconfliction server drops stray output because it does not know which client to route the information to. This is a drawback, but in practice it's limited to a few post-exploitation scripts. Metasploit is moving away from post-exploitation scripts in favor of post-exploitation modules. These modules are configured and executed just like exploits. Eventually, this problem with Meterpreter scripts will not exist.

Armitage queues commands in the Armitage client and deconfliction server. In the local client, the command queue delivers Meterpreter output to the GUI component that requested it. A user may execute multiple actions and Armitage will not become confused.

The deconfliction server uses the stored command identifier to identify the client that requested the command. When the deconfliction server finishes executing a command, it routes the output to the right client.

This multiplexing scheme creates the illusion that a shared access is not shared. When a team member executes a command, the command is added to the local command queue. When the command is executed locally, it is added to the deconfliction server command queue. When the command completes, the deconfliction server sends the command to the right client. The local client receives this output and routes it to the local GUI component. Figure 3 illustrates this process.



Command	Destination
idletime	Meterpreter #1

Command	Destination
ls	Client #2
idletime	Client #1

Figure 3: Meterpreter multiplexing in action

Windows command shells are a special case. Armitage interacts with the Windows command shell using Meterpreter channels. When Meterpreter executes a process, it creates a channel. Meterpreter provides commands to read from and write to these channels. When a client wants a command shell, it creates a new process through Meterpreter and it notes the channel associated with this process. Clients interact only with their own channels. Armitage uses the command queue to execute read and write commands to these channels. With this mechanism, multiple clients may interact with multiple command shells through one Meterpreter session.

Red Team Formations

This article has shown you how Armitage gives a network attack team real-time communication, data sharing, and session sharing built on the Metasploit framework. These features make it possible to experiment with different team organizations.

Excited about this shiny new technology, I used it to centralize the reconnaissance, exploitation, and post-exploitation activities in a collaborative capture-the-flag experiment. Under this organization, each team member used the shared Metasploit server to scan, attack, and carry out post-exploitation activities. I reckoned that this scheme would allow the team to move deeper into a target network, like an army marching deep into enemy territory. But I do not recommend this approach for attacking all hosts. A detected attack risks all sessions associated with the attack host. Detection is more likely when uncoordinated team members launch scans or attacks against the same host.

The most successful teaming option I've seen is to allow everyone to attack locally and handle post-exploitation through a shared Metasploit instance. Here, team members use their own tools to get access to a host. Once they're successful, they pass a session to the shared Metasploit server and kill their session. This decouples the attack host from the post-exploitation server. This worked well in practice, as everyone had access to existing sessions. This also reduced the normal red team chaos, as team members had no need to exploit already compromised hosts to get access. Team members with noisy attacks and scans risk detection of their local host only. Network defenders must find the attack source and the shared Metasploit server to keep the red team out.

Once a network foothold is available, it's safe to use a pivot set up on the shared Metasploit server for attack and reconnaissance of internal hosts. System administrators often focus on traffic entering and leaving their network, with little regard for what happens inside it. The risk of detection is low. Using a shared post-exploitation server also ensures that internal hosts get attention from the red team. Without session sharing, each team member needs a session on a host capable of reaching the desired internal targets.

A shared Armitage server also gives red teams the option to use Armitage as a dashboard for displaying the tactical situation. At the Northeast Collegiate Cyber Defense Competition we displayed Armitage using a projector in the red team room. The target area gave us situational awareness of what sessions we had at the time. The shared event log on the projector provided a timeline of recent sessions opening and closing.

Final Thoughts

Armitage helps network attack teams break away from the single-user assumption of Metasploit. In this article I described the communication, data sharing, and session sharing needs for network attack. I also described Armitage's features to meet these needs.

In practice, no feature completely replaced the old ways of collaboration. However, these features successfully augmented existing approaches. More importantly, session sharing allowed experimentation with different attack team organization and task delegation. In two cyber defense competitions, these features enabled collaboration on post-exploitation and shared situational awareness.

This article is the beginning of what's possible. I look forward to seeing what a mature red team does with this technology. It's now possible to experiment with and develop squad-level tactics for network attack.

References

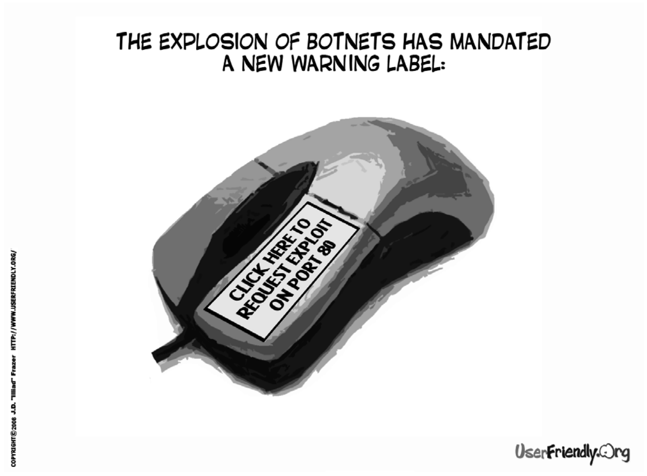
[1] The Metasploit Project: http://www.metasploit.com.

[2] J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, and R. Hassell, *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* (Wiley, 2004), pp. 147-148.

[3] Armitage homepage: http://www.fastandeasyhacking.com.

[4] Dradis Framework: http://dradisframework.org/.

[5] R. Linn, "Multiplayer Metasploit," DefCon 18 (2010): https://www.defcon.org/images/defcon-18/dc-18-presentations/Linn/DEFCON-18-Linn-Multiplayer-Metasploit.pdf.


THE EXPLOSION OF BOTNETS HAS MANDATED A NEW WARNING LABEL:
CLICK HERE TO REQUEST EXPLOIT ON PORT 80
COPYRIGHT © 2008 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
UserFriendly.Org

Do Users Verify SSH Keys?

PETER GUTMANN

Peter Gutmann is a researcher in the Department of Computer Science at the University of Auckland. He works on design and analysis of cryptographic security architectures and security usability, which includes lots of grumbling about the lack of consideration of human factors in designing security systems.
pgut001@cs.auckland.ac.nz

Abstract

No.

Discussion

Anecdotal evidence suggests that the majority of users will accept SSH server keys without checking them. Although SSH users are in general more security-aware than the typical Web user, the SSH key verification mechanism requires that users stop whatever they're trying to do when connecting and verify from memory a long string of hex digits (the key fingerprint) displayed by the client software. A relatively straightforward attack, for the exceptional occasion where the user is actually verifying the fingerprint, is to generate random keys until one of them has a fingerprint whose first few hex digits are close enough to the real thing to pass muster [1].

There are even automated attack tools around that enable this subversion of the fingerprint mechanism. The simplest attack, provided by a man-in-the-middle (MITM) tool called ssharpd [2], uses ARP redirection to grab an SSH connect attempt and then reports a different protocol version to the one that's actually in use (it can get the protocol version from the information passed in the SSH handshake). Since SSHv1 and SSHv2 keys have different fingerprints, the victim doesn't get the more serious key-changed warning but merely the relatively benign new-key warning. Since many users never check key fingerprints but simply assume that everything should be OK on the first connect, the attack succeeds and the ssharp MITM has access to the session contents [3]. (Since ssharp is based on a modified, rather old, version of OpenSSH, it'd be amusing to use one of the assorted OpenSSH security holes to attack the MITM while the MITM is attacking you.)

```
> ssh test@testbox
The authenticity of host 'testbox (192.168.1.38)' can't be established.
RSA key fingerprint is 86:9c:cc:c7:59:e3:4d:0d:6f:58:3e:af:f6:fa:db:d7.
Are you sure you want to continue connecting (yes/no)?

> ssh test@testbox
The authenticity of host 'testbox (192.168.1.38)' can't be established.
RSA key fingerprint is 86:9c:cc:d7:39:53:e2:07:df:3a:c6:2f:fa:ba:dd:d7.
Are you sure you want to continue connecting (yes/no)?
```

Figure 1: Real (top) and spoofed (bottom) SSH servers

A much more interesting attack can be performed using Konrad Rieck's concept of fuzzy fingerprints. Fuzzy fingerprints are SSH key fingerprints that are close enough to the real thing to pass muster, and as with the standard SSH MITM attack there's a tool available to automate the process for you [4]. This attack, illustrated in Figure 1, takes a target SSH server key and generates a new key for which the fingerprint is close enough to fool all but a detailed, byte-for-byte comparison. (Because the key doesn't have to be secure, merely to work for the RSA computation, you can simplify the key generation to require little more than an addition operation. The rate-limiting step then becomes the speed at which you can perform the hashing operation, and even there you can pre-compute almost everything but the last hash block before you start, making the key-search process extremely quick.) Since few users are likely to remember and check the full 40-hex-digit fingerprint for each server they connect to, this, combined with ssharpd, is capable of defeating virtually any SSH setup [5].

When the SSH fuzzy fingerprint work was first published, I wanted to run a real-world evaluation of its effectiveness, so I approached two large organizations with several thousand computer-literate (in some cases highly so) users to see if I could use their servers for the test. In order to determine the base rate for the experiment, I asked the IT support staff how many users had called or emailed to verify the SSH server key whenever it had changed in the past several years. They were unable to recall a single case, or locate any records, of any user ever verifying any SSH server key out-of-band. As the base rate for verification of completely different key fingerprints was already zero there didn't seem to be much to be gained by running an experiment with approximately matching fingerprints, since the result couldn't be worse than zero.

Conclusion

This result represents good news for both the SSL/TLS PKI camps and the SSH non-PKI camps, since SSH advocates can rejoice over the fact that the expensive PKI-based approach is no better than the SSH one, while PKI advocates can rest assured that their solution is no less secure than the SSH one.

References

[1] Dan Kaminsky, "Black Ops 2006: Pattern Recognition," 20th Large Installation System Administration Conference (LISA '06), December 2006: http://usenix.org/events/lisa06/tech/#friday.

[2] Sebastian Krahmer, "SSH for Fun and Profit," July 1, 2002: ftp://ftp.pastoutafait.org/pdf/ssharp.pdf.

[3] Jon Erickson, *Hacking: The Art of Exploitation*, No Starch Press, 2003.

[4] "THC Fuzzy Fingerprint," October 25, 2003: http://www.thc.org/thc-ffp/.

[5] Plasmoid, "Fuzzy Fingerprints: Attacking Vulnerabilities in the Human Brain," October 25, 2003: http://www.thc.org/papers/ffp.html.

Mesos

Flexible Resource Sharing for the Cloud

BENJAMIN HINDMAN, ANDY KONWINSKI, MATEI ZAHARIA, ALI GHODSI, ANTHONY D. JOSEPH, RANDY H. KATZ, SCOTT SHENKER, AND ION STOICA

Benjamin Hindman is a fourth-year PhD student at the University of California, Berkeley. Before working on resource management for cluster computing, he worked on resource management for single-node parallel computing. His interests include operating systems, distributed systems, programming languages, and all the ways they intersect.
benh@eecs.berkeley.edu

Andy Konwinski is a PhD student in computer science at the University of California, Berkeley, who has worked on tracing and scheduling in distributed systems such as Hadoop and Mesos.
andyk@berkeley.edu

Matei Zaharia is a fourth-year graduate student at the University of California, Berkeley, working with Scott Shenker and Ion Stoica on topics in cloud computing, operating systems, and networking. He is also a committer on Apache Hadoop. He got his undergraduate degree at the University of Waterloo in Canada.
matei@eecs.berkeley.edu

Ali Ghodsi got his PhD from KTH Royal Institute of Technology in 2007. He is on leave from his position as an assistant professor at KTH and has been visiting the University of California, Berkeley, since 2009. His interests include cloud computing, distributed computing, and micro-economic applications in computer science.
alig@cs.berkeley.edu

Anthony D. Joseph is a Chancellor's Associate Professor in Electrical Engineering and Computer Science at the University of California, Berkeley. He is developing adaptive techniques for cloud computing, network and computer security, and security defenses for machine-learning–based decision systems. He also co-leads the DETERlab testbed, a secure scalable testbed for conducting cybersecurity research.
adj@eecs.berkeley.edu

Randy H. Katz is the United Microelectronics Corporation Distinguished Professor in Electrical Engineering and Computer Science at the University of California, Berkeley, where he has been on the faculty since 1983. His current interests are the architecture and design of modern Internet Datacenters and related large-scale services.
randy@cs.Berkeley.edu

Scott Shenker spent his academic youth studying theoretical physics but soon gave up chaos theory for computer science. Continuing to display a remarkably short attention span, over the years he has wandered from computer performance modeling and computer networks research to game theory and economics. Unable to hold a steady job, he currently splits his time between the University of California, Berkeley, Computer Science Department and the International Computer Science Institute.
shenker@icsi.berkeley.edu

Ion Stoica is an Associate Professor in the EECS Department at the University of California, Berkeley, where he does research on cloud computing and networked computer systems. Past work includes the Chord DHT, Dynamic Packet State (DPS), Internet Indirection Infrastructure (i3), declarative networks, replay-debugging, and multi-layer tracing in distributed systems. His current research includes resource management and scheduling for data centers, cluster computing frameworks for iterative and interactive applications, and network architectures.
istoica@eecs.berkeley.edu

Clusters of commodity servers have become a major computing platform, powering both large Internet services and a growing number of data-intensive enterprise and scientific applications. To reduce the challenges of building distributed applications, researchers and practitioners have developed a diverse array of new software frameworks for clusters. For example, frameworks such as memcached

[4] make accessing large datasets more efficient, while frameworks such as Hadoop [1] and MPI [6] simplify distributed computation.

Unfortunately, sharing a cluster efficiently between two or more of these frameworks is difficult. Many operators statically partition their clusters at physical machine granularities, yielding poor overall resource utilization. Furthermore, static partitioning makes it expensive to share big datasets between two computing frameworks (e.g., Hadoop and MPI): one must either copy the data into a separate cluster for each framework, consuming extra storage, or have the frameworks read it across the network, reducing performance.

This article introduces Mesos, a platform that enables fine-grained, dynamic resource sharing across multiple frameworks in the same cluster. For example, using Mesos, an organization can simultaneously run Hadoop and MPI jobs on the same datasets, and have Hadoop use more resources when MPI is not using them and vice versa. Mesos gives these and other frameworks a common interface for accessing cluster resources to efficiently share both resources and data.

In designing Mesos, we sought to make the system both flexible enough to support a wide range of frameworks (and maximize utilization by pooling resources across all these frameworks), and highly scalable and reliable (to be able to manage large production clusters). Specifically, we had four goals:

High utilization: share resources dynamically as the demand of each application changes
Scalability: support tens of thousands of machines and hundreds of concurrent jobs
Reliability: recover from machine failures within seconds
Flexibility: support a wide array of frameworks with diverse scheduling needs

Mesos achieves these goals by adopting an *application-controlled* scheduling model. The Mesos core is only responsible for deciding how many resources each framework should receive (based on an operator-selected policy such as priority or fair sharing), while frameworks decide which resources to use and which computations to run on them, using a mechanism called *resource offers*. This design has the dual benefit of giving frameworks the flexibility to schedule work based on their needs and letting the Mesos core be simple, scalable, and robust. Indeed, we show that Mesos scales to 50,000 nodes, recovers from master failures in less than 10 seconds, and lets applications achieve nearly perfect data locality in scheduling their computations.

Finally, Mesos provides important benefits even to organizations that only use one cluster computing framework. First, an organization can use Mesos to run multiple, isolated instances of the framework on the same cluster (e.g., to isolate production and experimental Hadoop workloads), as well as multiple versions of the framework (e.g., to test a new version). Second, Mesos allows developers to build *specialized* frameworks for applications where general abstractions like MapReduce are inefficient, and have them coexist with current systems. Later in this article we describe a specialized framework we developed for iterative applications and interactive data mining called Spark, which can outperform Hadoop by a factor of 30 for these workloads. We hope that other organizations also leverage Mesos to experiment with new cluster programming models.

Mesos began as a research project at UC Berkeley and is now open source under the Apache Incubator. It is actively being used at Twitter, Conviva, UC Berkeley, and UC San Francisco.

Mesos Architecture

Mesos enables efficient resource sharing across frameworks by giving them a common API to launch units of work, called *tasks*, on the cluster. A task typically runs on a slice of a machine, within a resource allocation chosen by the framework (e.g., 1 CPU core and 2 GB RAM). Mesos isolates tasks from each other using OS facilities like Linux Containers [2] to ensure that a runaway task will not affect other applications.

To support a wide range of frameworks while remaining scalable and robust, Mesos employs an application-controlled scheduling model. Mesos decides how many resources each framework should receive according to an organization-defined policy such as fair sharing. However, each framework is responsible for dividing its work into tasks, deciding which tasks to run on each machine, and, as we shall explain, selecting which machines to use. This lets the frameworks perform application-specific placement optimizations: for example, a MapReduce framework can place its map tasks on nodes that contain their input data.

Figure 1 shows the architecture of Mesos. The system has a fault-tolerant *master* process that controls *slave* daemons on each node. Each framework that uses Mesos has a *scheduler* process that registers with the master. Schedulers launch tasks on their allocated resources by providing *task descriptions*. Mesos passes these descriptions to a framework-specific *executor* process that it launches on slave nodes. Executors are also reused for subsequent tasks that run on the same node, to amortize initialization costs. Finally, Mesos passes *status updates* about tasks to schedulers, including notification if a task fails or a node is lost.

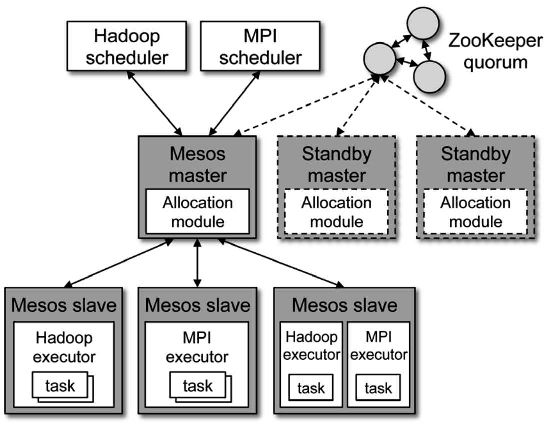


Figure 1: Mesos architecture, showing two running frameworks (Hadoop and MPI)

Mesos uses a mechanism called *resource offers* to let frameworks choose which resources to use. When resources on a machine become free, Mesos offers them to each framework scheduler in turn, in an order defined by the cluster's allocation policy (e.g., starting with the framework furthest below its fair share). Each framework may accept the resources and launch a task using some of them, or reject the resources if, for example, it has no data on that machine. Refusing resources keeps the framework at the front of the allocation queue, ensuring that it is offered future resources before other frameworks. While it may seem counterintuitive that refusing resources can help frameworks, we found that a simple policy where frameworks wait a short time for local resources achieves near-perfect data locality in typical cluster workloads.

One natural concern with resource offers is whether a framework will need to wait for a large number of offers to find a resource that it wants. To prevent this scenario, Mesos also provides an API for *requests* that lets frameworks specify which resources they wish to be offered. For example, a framework might provide a minimum amount of memory it needs, or a whitelist of nodes to run on. One important benefit of the resource offer model, however, is that frameworks whose needs cannot be expressed using requests can still achieve good task placement. That is, requests are an optimization, while resource offers guarantee correctness and allow the system to support arbitrary framework placement preferences.

More importantly, Mesos's application-controlled scheduling model also helps make the system extremely simple, scalable, and robust. Here is how Mesos achieves each of the four goals outlined in the introduction:

> **High utilization:** Each framework is only allocated the resources to run its current tasks, as opposed to a static partition of the cluster.
> **Scalability:** The Mesos master only makes inter-framework scheduling decisions (to pick which framework has priority for new offers), which are much simpler than the intra-framework decisions required for many applications (e.g., to achieve data locality). Our optimized C++ implementation can make thousands of decisions per second with sub-second latency and manage tens of thousands of nodes.
> **Reliability:** The Mesos master only needs to store *soft state*: the list of currently active frameworks and tasks. Therefore, if the master crashes, a standby master can take over and repopulate its state within seconds when the frameworks and slaves connect to it.
> **Flexibility:** Resource offers allow each framework to control its scheduling, while requests represent an extensible and efficient mechanism for frameworks to indicate their placement needs to the master.

Example Framework: Computing Pi

The Mesos team has already ported several popular frameworks, like Hadoop and MPI, to run on Mesos, but one of our main goals with Mesos was to let users easily develop other cluster applications that can run alongside existing frameworks. To show you how a Mesos framework looks from a programmer's perspective, Figure 2 illustrates a simple Python framework that computes π. Mesos also has APIs in C++ and Java.

The framework is composed of a scheduler, which launches tasks, and an executor, which runs them. The scheduler launches NUM_TASKS independent tasks, each of which computes an estimate of π and then averages the results. Each task uses an inefficient, but easy to explain method to estimate π: it picks random points in the unit square (from (0,0) to (1,1)) and counts what fraction of them fall in the unit circle. This fraction should be π/4, because one quarter of the unit circle is inside this square, so we multiply the result by 4. The tasks return their results in the data field of a Mesos status update. Note that the executor runs each task in a separate thread, in case a single machine is given multiple tasks.

Thanks to building on top of Mesos, this application does not need to implement infrastructure for launching work on the cluster or for communicating between tasks and the main program. It can just implement a few callbacks, such as `resourceOffer` and `statusUpdate`, to run on the Mesos-managed cluster.

```
class MyScheduler(mesos.Scheduler):
  def resourceOffer(self, driver, id, offers):
    tasks = []
    for offer in offers:
      if self.tasksStarted < NUM_TASKS:
        self.tasksStarted += 1
        task = createTask(offer.slave_id,
                          {"cpus": 1, "mem": 32})
        tasks.append(task)
    driver.replyToOffer(id, tasks, {})

  def statusUpdate(self, driver, update):
    if update.state == TASK_FINISHED:
      self.resultSum += float(update.data)
      self.tasksDone += 1
      if self.tasksDone == NUM_TASKS:
        driver.stop()
        result = self.resultSum / NUM_TASKS
        print "Pi is roughly %f" % result
```

```
class MyExecutor(mesos.Executor):
  def launchTask(self, driver, task):
    # Create a thread to run the task
    thread = Thread(target = self.runTask,
                    args = (driver, task))
    thread.start()

  def runTask(self, driver, task):
    NUM_SAMPLES = 1000000
    count = 0.0
    for i in range(1, NUM_SAMPLES):
      x = random()
      y = random()
      if x*x + y*y < 1:
        count += 1
    result = 4 * count / NUM_SAMPLES
    driver.sendStatusUpdate(
      task.task_id, TASK_FINISHED, str(result))
```

Figure 2: A sample Mesos framework, in Python, for computing π. The scheduler (left) launches NUM TASKS tasks and averages their results, while the executor (right) runs a separate estimation of π in a thread for each task. We omit some boilerplate initialization code.

Use Cases

Mesos Usage at Twitter

Twitter has been using Mesos internally as an end-to-end framework for deploying some of their application services. Using Mesos for some of their services appealed to Twitter for many reasons, including:

Flexible deployment: Statically configuring where services should run makes it difficult for different teams within Twitter to operate autonomously. By leveraging Mesos, engineering teams can focus on doing code deploys against a generic pool of resources, while the operations team can focus on the operating system and hardware (e.g., rebooting machines with new kernels, replacing disks, etc).
Increased utilization: Many services within the cluster are sharded for better fault-tolerance and do not (or cannot) fully utilize a modern server with up to 16 CPU cores and 64+ GB of memory. Mesos enables Twitter to treat machines as a pool of resources and run multiple services on the same machine, yielding better overall cluster utilization.
Elasticity: Certain services might want to "scale up" during peak or unexpected events when traffic and load has increased. Using Mesos, it's easy for different services to consume more or less resources as they are needed.

Using Mesos to facilitate normal datacenter maintenance and upgrades has been especially compelling at Twitter. Because Mesos notifies frameworks when machines fail, operators can easily remove machines from the cluster (provided there is enough general capacity). Frameworks simply react to these "failures" and reschedule their computations as needed.

Because of Mesos's two-level scheduling design, Twitter can provide its own organizational policies for how resources should be allocated to frameworks. For example, some machines can have most of their resources dedicated to applications serving user requests (e.g., Web servers and databases), allowing unused "slack" resources to be used for lower-priority applications. Twitter uses Linux Containers [2] to isolate services running on the same machine from one another.

Using Mesos, engineers at Twitter have been able to easily experiment with building new services, including spam detectors, load testers, distributed tracing frameworks, and service quality monitors, among others. Twitter continues to experiment with using Mesos for deploying more services in their clusters.

Managing Hadoop Clusters

Running the popular Hadoop framework on Mesos has many advantages. In current versions of Hadoop, a single master process (the job tracker) manages an entire cluster, which creates a single point of failure and leads to poor isolation between workloads (for example, a single user submitting too large a job may crash the job tracker). Mesos has been designed to support many concurrent frameworks, so it can run each Hadoop job separately, with its own job tracker, isolating MapReduce applications from each other. Mesos also provides stronger isolation of the resources on each machine through Linux Containers. Finally, from an operations viewpoint, an important advantage of running Hadoop on Mesos is that it enables organizations to experiment with different versions of Hadoop in one cluster, or to gradually upgrade from an older version to a newer one.

More recently, the next-generation Hadoop design was announced, which refactors the current Hadoop job tracker into a simpler resource manager and a separate application master for each job to achieve similar isolation benefits [7]. These new, lightweight application masters fit cleanly as framework schedulers in the Mesos model, and we are working to port them to run on top of Mesos to let Hadoop share resources with the other frameworks supported by Mesos.

Spark: A Framework for Low-Latency In-Memory Cluster Computing

One of our main goals with Mesos was to enable the development of new analytics frameworks that complement the popular MapReduce programming model. As an example, we developed Spark, a framework for iterative applications and interactive data mining that provides primitives for in-memory cluster computing. Unlike frameworks based on acyclic data flow, such as MapReduce and Dryad, Spark allows programmers to create in-memory *distributed datasets* and reuse them efficiently in multiple parallel operations. This makes Spark especially suitable for iterative algorithms that reuse the same data repeatedly, such as machine learning and graph applications, and for interactive data mining, where a user can load a dataset into memory and query it repeatedly. As previously mentioned, Spark can outperform Hadoop by a factor of 30 in these tasks.

Spark provides a language-integrated programming interface, similar to Microsoft's DryadLINQ [9], in Scala [5], a high-level language for the Java VM. This means that users can write functions in a single program that automatically get sent to a cluster for execution. For example, the following code snippet implements the π estimation algorithm from earlier in this article:

```
val count = spark.parallelize(1 to NUM_SAMPLES).map(i =>
  val x = Math.random
  val y = Math.random
  if (x*x + y*y < 1) 1.0 else 0.0
).reduce(_ + _)
println("Pi is roughly " + 4 * count / NUM_SAMPLES)
```

Here, the arguments to `map` and `reduce` are Scala function literals (closures) that are automatically shipped to the Mesos cluster for parallel execution. The `_ + _` syntax means a function to add two numbers.

As a more interesting example, the code below implements logistic regression [3], an iterative machine learning algorithm for classification (e.g., identifying spam). We build an in-memory distributed dataset called points by loading the data in a text file, then run map and reduce operations on it repeatedly to perform a gradient descent. Loading points into memory allows subsequent iterations to be much faster than the first and lets Spark outperform Hadoop for this application.

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D)
for (i <- 1 to ITERATIONS) {
  val gradient = points.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}
println("Final separating parameter: " + w)
```

Spark can also be used interactively from a modified Scala interpreter to build and query distributed datasets. We have used Spark to analyze several large traces in the course of our research.

Spark is being used by several groups of machine learning researchers at Berkeley, for projects including traffic estimation and spam detection on social networks. It is also being used at Conviva, an online video distribution company, to run analytics on large Hadoop and Hive datasets. The system has grown into a research project of its own, and is open source at http://www.spark-project.org.

Experimental Results

We evaluated Mesos through a series of experiments included in our NSDI '11 paper [8]. We sketch three of them here.

Job performance in a shared cluster: In the first experiment, we wanted to compare Mesos's performance with a static partitioning of a cluster, where each partition ran a separate framework. For this, we ran a 100-node cluster on Amazon EC2 and concurrently ran four frameworks: (1) a mixed Hadoop workload based on the workload at Facebook, (2) a Hadoop batch workload, (3) a Spark instance running machine learning jobs, and (4) the popular Torque scheduler running MPI jobs. Table 1 compares job completion times for Mesos and static partitioning. As seen, most jobs speed up when using Mesos. Note that the Torque framework was configured to never use more than a fourth of the cluster. It is therefore expected not to see any speedup. The slight slowdown for Torque was due to a slow machine on EC2. The speedups are due to frameworks scaling up and down dynamically to use other resources when another framework's demand is low. In contrast, with static partitioning, frameworks are confined to a fixed fraction of the cluster machines.

Scalability: The second experiment investigated how the Mesos master scales with the cluster size. We ran 200 frameworks filling the whole cluster with tasks that on average took 30 seconds to finish. Thus, the Mesos master was busy making scheduling decisions as the tasks were continuously finishing and being launched by the frameworks. We then launched one additional framework that ran one task and measured the overhead of scheduling this task. The result was that the scheduling overhead remained on average under one second for up to 50,000 slave daemons (which we ran as separate processes on up to 200 physical machines), showing that the master can manage large clusters with heavy workloads. Much of the system's scalability stems from our use of C++ and efficient I/O mechanisms in the master.

Reliability: In the final experiment, we wanted to measure how fast Mesos recovered from master failures. As in the scalability experiment, we filled the cluster with tasks. We then killed the master node and measured how long it took for the system to elect a new master node and repopulate its state. For a 4000-node cluster, the whole system recovered within 10 seconds.

Framework	Sum of Exec Times w/ Static Partitioning (s)	Sum of Exec Times with Mesos (s)	Speedup
Facebook Hadoop Mix	7235	6319	**1.14**
Large Hadoop Mix	3143	1494	**2.10**
Spark	1684	1338	**1.26**
Torque / MPI	3210	3352	**0.96**

Table 1: Aggregate performance of each framework in the macro-benchmark (sum of running times of all the jobs in the framework). The speedup column shows the relative gain on Mesos.

Conclusion

As the number of software frameworks for clusters grows, it is becoming increasingly important to dynamically share resources between these frameworks. We have presented Mesos, a scalable and reliable platform that enables efficient, fine-grained sharing of clusters among diverse frameworks by giving frameworks control over their scheduling. Mesos can currently run Hadoop, MPI, the Torque resource manager, and a new framework, called Spark, for fast in-memory parallel computing. We hope that Mesos also encourages the development of other frameworks that can coexist with these. Mesos is open source at http://www.mesosproject.org.

References

[1] Apache Hadoop. : http://lucene.apache.org/hadoop.

[2] Linux containers Containers (LXC) overview document.: http://lxc.sourceforge.net/lxc.htmlhttp://lxc.sourceforge.net/.

[3] Logistic regression—Wikipedia. : http://en.wikipedia.org/wiki/Logistic_regression.

[4] memcached—a distributed object caching system. : http://memcached.org/.

[5] Scala programming language. : http://www.scala-lang.org/.

[6] The Message Passing Interface (MPI) Standard. : http://www.mcs.anl.gov/research/projects/mpi.

[7] The Next Generation of Apache Hadoop MapReduce. : http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11).*

[9] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P.K. Gunda, and J. Currey, "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language," in *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08),* pp. 1–14.

LISA ’11 Theme— “DevOps: New Challenges, Proven Values”

THOMAS A. LIMONCELLI AND DOUG HUGHES

LISA ’11 Co-Chairs
lisa11chairs@usenix.org

The theme for LISA ’11 is “DevOps: New Challenges, Proven Values.” DevOps is “an umbrella concept that refers to anything that improves the interaction between development and operations” [1]. While usually associated with Web operations, the tools and techniques are now being mainstreamed into the enterprise. Although DevOps is new, it embodies themes long popular at LISA: automation, performance, scaling, collaboration, and cooperation.

There is an important shift happening in system administration, and we felt it was important to acknowledge this change by making it this year’s theme. What is this change? In the status quo, sysadmins obtain software from vendors and struggle with operational issues with varying levels of support. Scaling, disaster recovery, operational efficiency is left as an exercise for the user.

Recently there has been a trend towards self-sourcing. A company providing a Web-based service develops it in-house and the operations work is done by in-house system administrators. Development and Operations work together, with shared responsibility for the success of the whole. This last part bears repeating: shared responsibility.

Let’s call this new way “DevOps.” While we’re at it, let’s acknowledge that it isn’t new. However, it is becoming more frequent and is the dominant paradigm for the high-growth segments of our industry. As a result, new cultural “best practices” are becoming apparent.

While in-house software development is nothing new, the Web-centric world creates opportunities that hadn’t existed before. Likewise, you can manage a new-style, Web-based service with the paradigm of the past, but you would miss out on opportunities that were unavailable before.

For example, the old way often involves packaged, shrink-wrapped software. A new release comes along each year. The effort to ship a new release is huge. You have a printer that makes boxes, a factory that produces media, an assembly line that puts them all together and ships them. Every new release involves a new manual, a new box, and inventory strategies to deal with the old version sitting in the warehouse. All these pieces come together once a year. To make it all happen we use software development methodologies with names like “Waterfall,” “Spiral,” and “Release Trains.” If there are millions of users, there are millions of deployments.

In the Web world, shipping a new release has much lower overhead. There is no physical package. There is, ostensibly, one deployment. This gives birth to frequent releases: weekly, daily, maybe continuously. This, in turn, leads to new software development methodologies like "Agile Development." Yes, you absolutely can use a waterfall model and only update the Web site's software every year, but you would miss out on opportunities that were unavailable before.

Opportunities Arise for Operational Improvements

As mentioned previously, in the old way, sysadmins are solely responsible for operational issues with some or little support from the vendor. Certainly a Web site can be managed that way by treating the in-house developers as the vendor. We can do even better.

DevOps promotes a different culture: developers and operations work together as partners—as a team. The way a manager can bring this about is to make the two groups share responsibility for the operational success of the service.

Previous to DevOps, I'd make feature requests that would benefit the operational efficiency of a service (i.e., make my life easier), and countless times I've seen those feature requests ignored. That attitude changes when the on-call rotation is shared among the developers and the system administrators. Nothing develops empathy for the importance of operational efficiency like a week of pager duty. When informed by operational experience, software development changes in ways that directly benefit the operational efficacy of a company.

It is quite refreshing to leave the "toss each release over the wall" world and enter the DevOps "we're all on the same boat" model. Collaboration between developers and system administrators means that the operational aspects of each new feature are worked out ahead of time. Rather than developing every feature a deployment may need, teams can focus on just the operational features needed by your deployment. Developers have a better appreciation for what information should be logged to ease debugging and what variables need to be exposed to do proper monitoring and metrics.

We're All Programmers Now

In such an environment, system administrators need to become more like developers. Automation becomes critical. Sysadmins have always been "pro-automation" but "who has time to automate anything?" is such a frequent refrain that outsiders would think some of us are anti-automation. There may be justifiable reasons to not automate something, but three of them are disappearing:

1. You can't automate physical work such as installing a new machine.
2. It doesn't make sense to automate something that happens once or rarely.
3. Management isn't funding automation projects, because they don't see the value.

The first objection disappears when using EC2 or other "infrastructure as a service" (IaaS) cloud providers. When installing a new machine is an API call, we're all programmers now.

The second objection disappears because nothing happens once anymore. With old-style packaged software, once it is installed it is installed. The next upgrade might be a year from now. In the Web world, scaling makes very few things "rare." A one-in-a-million error happens hourly and becomes worth fixing. Requests previously done manually must be turned into "self-service" portals so that there is less waiting. If developers frequently need a server's OS reloaded, why should they wait for a system administrator to do that? The portal can verify they own the machine and do the entire process. If developers need another machine, why should they wait for a system administrator to purchase, install, and configure it? The portal can allocate a virtual machine and bill the developers' project code.

The third objection disappears because, in a Web environment, management does see the value, or at least good management does. Velocity becomes important. Uptime becomes important. And, even more importantly, operational efficiency becomes a competitive advantage. These things require the consistency and scale that only automation can achieve.

DevOps reflects a cultural change that reflects the new paradigm. DevOps is a culture. It isn't a job description: you can't hire a "devop." It isn't a technology: you can't buy a software package that provides the "devops service." It isn't a job title: people do not have "devops" on their business card.

DevOps Beyond the Web

DevOps is mature enough that the innovations are now feeding into areas outside Web-based services. LISA is a unique opportunity to apply the lessons of DevOps to traditional enterprise computing, storage administration, security, and network administration.

Traditional computing organizations need the new insights that DevOps culture brings. The stellar uptime of Google, Facebook, and other popular Internet sites has created high expectations for the most simple internal Web app. People want to be able to fill out their expense report forms anytime, even nights and weekends. That was easy when doing so meant a paper form, since paper has incredible uptime. Now such forms are online and we sysadmins are under pressure to make sure they are always available. Packaged software may still ship yearly, but security updates are a constantly flood comparable to the launch schedule of Web sites.

This "mainstreaming" of DevOps is important to us as an industry. It is LISA's great responsibility, as the leader in advancing the state of the art in system administration, to make this happen. DevOps and USENIX LISA embody the same cultural values: automation, performance, scaling, collaboration, and cooperation. These are the values we've always seen at the LISA conference since it began 25 years ago [2].

In a recent phone conversation, Andrew Hume asked Tom to define DevOps. After Tom rambled on for five minutes, Andrew interrupted, "Oh, so they've given a name to the way I've been doing things for years!" He wasn't that far off. Tom reviewed all the presentations from LISA '10 and determined that 27% could easily be classified as "DevOps" and 31% could be classified as "mostly DevOps." Thus it is easy to assert that last year's theme was DevOps but we didn't know it. If we can achieve similar ratios in 2011, the theme will be a success.

LISA '11 will include many new speakers, as well as many familiar faces.

We look forward to seeing your familiar face there too!

References

[1] This definition is attributed to John Allspaw.

[2] By the way, this is the 25th LISA. Happy Silver Anniversary!

Practical Perl Tools

Got My Mojolicious Working

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

Ah, last issue's column. Who can forget it? It had poise. It had verve. It had rhythm. And it certainly had a better introduction than this one. Lest you have forgotten that bygone issue, let met do a little recap. We looked at a very spiffy Web framework called Dancer. Actually, on the official Dancer Web site at perldancer.org they call it a "micro Web application framework," because it is so simple and concise. Based on Ruby's Sinatra framework, it lets you write a script that looks like this:

```
use Dancer;

get '/hello/:name' => sub {
      return "Why, hello there " . params->{name};
};

dance;
```

...and then spin up a tiny Web server that will process requests simply by running the script.

The code looks for incoming GET requests with a path of /hello/{something} and then returns a jovial response of "Why, hello there {something}". The line that begins with "get..." is a route specification. It states that if a request comes in that matches that specification, the associated code should be run. This route-based approach, where the programmer essentially provides a dispatch table, seems to be pretty common in the Web framework world these days. Another pervasive feature for Web frameworks is the ability to generate Web pages through the use of a templating system. We took a brief look at Dancer's support for templates and a helper command-line script that created an entire sample application directory structure which included a place for those templates. At that point, I had to end our time together and leave you with a bit of a cliffhanger by mentioning but never really naming a "competing" Web framework that compared to Dancer. That's what we are here to explore today.

In this column, we're going to look at Mojolicious (http://mojolicio.us, cute, huh?). More precisely, we're going to mostly consider Mojolicious::Lite, the "micro Web framework" part of the larger package, because it is closer in nature to Dancer. Let's look at our first piece of Mojolicious::Lite code:

```
use Mojolicious::Lite;

get '/hello/:name' => sub {
    my $self = shift;
    $self->render(text => "Why, hello there " . $self->param('name'));
};

app->start;
```

Why yes, it does look remarkably like the Dancer example above with a wee bit of object-oriented programming syntax snuck in there. The close resemblance is intentional on my part, but the first bit of code in the Mojolicious::Lite introduction is nearly identical to what I wrote above. There's a bunch more going on here, but if you are comfortable with the Sinatra-like syntax in Dancer, you'll be fine with Mojolicious::Lite as well.

What Is Mojolicious and Where Can I Get One?

One thing I don't want to do with this column is imply that Mojolicious or even Mojolicious::Lite is just Dancer++. It would be possible to write a column that says "Dancer offers this, but Mojolicious::Lite offers that plus this other thing," but I think that would do a disservice to both frameworks. If I did that you wouldn't really get a sense of just how different they are, even just in their basic worldview. So let me step way back for a moment and discuss Mojolicious itself and how it relates to Mojolicious::Lite.

Once upon a time, a German Perl programmer by the name of Sebastian Riedel took over maintenance of a Web application framework called Maypole. Eventually he left that project to found another Web application framework, called Catalyst (Catalyst is probably the most popular of the Perl Web app frameworks). Riedel eventually left the Catalyst project and created yet another framework, called Mojolicious. And that's where our story begins. I have no idea why Riedel is such a serial framework creator, I just know he tends to start and leave good stuff behind in his wake that tends to move the field forward.

One of the first ways Mojolicious distinguishes itself from other frameworks is in its dependencies on other modules outside the Perl core.

It doesn't have any.

"But that's crazy talk!" you say. "What about LWP? How about the parsing of HTTP messages? URL processing? Templates? Cookies? Surely it doesn't implement its own full featured UNIX-optimized preforking async I/O HTTP 1.1 and WebSocket server with IPv6, TLS, Bonjour, epoll, kqueue, and hot deployment support?" That last question is a bit over the top (and if you really said something like that I would probably start edging away slowly), but, yes, Mojolicious does indeed ship with code that implements all of these things.

I'm not entirely sure all of this reimplementation is automatically a good thing. It means that the rest of the Perl world isn't constantly battle-testing Mojolicious's building blocks in the same way other modules do when they rely on common modules such as HTTP parsing libraries. A rejoinder to this concern comes in the Mojolicious FAQ in response to the question, "Why reinvent wheels?":

"Because we can make them rounder."

The same FAQ document also points out that Mojolicious will also optionally use some external Perl modules such as IO::Socket::SSL when it makes sense to "provide advanced functionality."

One plus of this approach is that the functionality found in a Web framework like Mojolicious becomes bounded, not by the restrictions of that framework's external dependencies, but, rather, by the imagination of the framework's author. In Mojolicious's case, it has let the author create a very nice prototyping Web application framework in the Sinatra vein with lots of hidden power under the hood. Mojolicious::Lite is a "micro Web framework built around Mojolicious," according to the documentation. (And in case you were curious, Mojolicious itself is built on top of something called Mojo, which the author calls "a flexible runtime environment for Perl Web frameworks"). Let's focus on some parts of Mojolicious::Lite.

Mojolicious::Lite's Templates

I don't want to rehash the last column's discussion of routes, because they get used in a very similar fashion in Mojolicious::Lite. Once you understand the basic idea, your understanding can be applied in both places without much adaptation. Instead, let's look at a few places where Mojolicious::Lite does things a bit differently, because they will reveal some of the hidden power I mentioned a moment ago.

Templating is one of the areas worth exploring. Mojolicious::Lite has its own template language (although you can use other template engines if you so desire) called Embedded Perl. Templates can either live in files that end with .ep (usually found in a templates directory in your application) or actually embedded in the script itself at the end in a __DATA__ handle. Since most of the introductory documentation demonstrates the latter, let's use that too. Here's a modified version of our last example:

```
use Mojolicious::Lite;

get '/hello/:name' => sub {
   my $self = shift;
   $self->render();
} => 'hello';

app->start;

__DATA__
@@ hello.txt.ep
<%= "Why, hello there $name" %>

@@ hello.html.ep
<html>
<head><title>"Hello"</title></head>
<body>
Why, hello there <strong><%= $name %></strong>
</body>
</html>
```

The first thing you'll notice is that I've added two templates, hello.txt.ep and hello.html.ep, to the __DATA__ section of the script. Since you don't see it used that often in practice let me mention that the idea of a __DATA__ section is actually a Perl, not a Mojolicious thing. In Perl, if you add __DATA__ at the end of the script, that script can read the lines following that marker as if they were coming from a real filehandle (e.g., with while (<__DATA__>) {something...}). Mojolicious::Lite lets you specify multiple "files" in that section by prefixing each section with @@ and the name of the file. It should be mentioned that using the __DATA__ section is just a shortcut. If we wanted to create two separate files and place them in the right templates directory for an application, they would function exactly the same way.

In the templates themselves, you see the use of the <%= Perl_expression %> tag. The contents of that tag are replaced by the result of the enclosed expression after Perl has evaluated it. Here are the other possibilities from the Mojo::Template documentation:

```
<% Inline Perl %>
<%= Perl expression, replaced with result %>
<%== Perl expression, replaced with XML escaped result %>
<%# Comment, useful for debugging %>
% Perl line
%= Perl expression line, replaced with result
%== Perl expression line, replaced with XML escaped result
%# Comment line, useful for debugging
```

Using either the "<" or "%" convention, you can basically embed whatever Perl code you would like into the template (hence the name). The code above just substitutes in the value of $name, but it could just as easily have been a much more complex Perl expression.

While we're talking about the $name variable, I should explain how that variable gets set, since there are a few things going on behind the scenes. In the route specification above, we provided the named placeholder using the syntax ":name". When an incoming request matches that route, the part of the request that matched the placeholder automatically gets extracted and stored in something called "the stash" under that name. The stash is a temporary holding spot for stuff like template values. When the template gets rendered, it looks in the stash for these values.

Data can also be put in the stash by hand; for example, we can also write code such as:

```
$self->stash('editor' => 'rik');
```

and $editor would be available to render in a template.

There are two additional changes between the two previous code samples that reveal more interesting default "smartiness" in Mojolicious::Lite. The first is the addition of the following to the routing specification:

```
} => 'hello';
```

This gave the route a name. Mojolicious::Lite will use that name when deciding what template to render. That's why we didn't have to specify a template in this line:

```
$self->render();
```

When I removed the arguments to the render() call, I brought two Mojolicious::Lite defaults into play. The first is the use of the route name to select the appropriate template basename (the name without the format suffix). The second is an automatic detection of format (i.e., .html or .txt). If I use the following URL in the browser:

```
http://127.0.0.1:3000/hello/dnb.txt
```

the hello.txt.ep template is rendered for me. Similarly, if I use:

```
http://127.0.0.1:3000/hello/dnb.html
```

the hello.html.ep template is chosen.

Mojolicious::Lite's Filigrees

This idea of "guess what I might need to do and make it easy" pervades the whole package. Here's an example of using sessions:

```
use Mojolicious::Lite;

get '/login/:username' => sub {
    my $self = shift;
    $self->session( username => $self->param('username') );
    $self->render();
} => 'login';

get '/hello' => sub {
    my $self = shift;
    my $username = $self->session('username') || '(no one)';
    $self->stash( username => $username );
    $self->render();
} => 'hello';

get '/logout' => sub {
    my $self = shift;
    my $username = $self->session('username') || '(no one)';
    $self->stash( username => $username );
    $self->session( expires => 1 );
    $self->render();
} => 'logout';

app->secret("shhh, I'm hunting wabbits");
app->start;

__DATA__
@@ login.html.ep
Ok, <%= $username %> is logged in.

@@ hello.html.ep
Welcome back  <%= $username %> !

@@ logout.html.ep
<%= $username %> has been logged out.
```

The only really new concept in this example can be found in the use of the session() method calls. Mojolicious::Lite provides built-in session management to help deal with the perpetual question of how to maintain state (e.g., someone's logged-in status) across a set of stateless HTTP requests. Mojolicious::Lite does all the work for you, including generating, exchanging, and expiring HMAC-MD5 signed session cookies. The app->secret() call above simply sets the secret used to sign the cookies. It is a good idea to set your own secret like this so the default secret (the name of the application) is not used.

Here's another piece of code right from the documentation:

```
use Mojolicious::Lite;

any '/upload' => sub {
    my $self = shift;
    if (my $example = $self->req->upload('example')) {
        my $size = $example->size;
        my $name = $example->filename;
        $self->render(text => "Thanks for uploading $size byte file $name.");
    }
};

app->start;
__DATA__

@@ upload.html.ep
<!doctype html><html>
    <head><title>Upload</title></head>
    <body>
        <%= form_for upload =>
                (method => 'post', enctype => 'multipart/form-data') => begin %>
            <%= file_field 'example' %>
            <%= submit_button 'Upload' %>
        <% end %>
    </body>
</html>
```

This code shows how easy it is to perform a file upload using Mojolicious::Lite. In the template, you can see some built-in tag helpers such as "form_for", "file_field", and "submit_button" that make creating a form easier by generating the right HTML. What you can't see from just this code is that Mojolicious::Lite will (1) prevent the user from uploading a file that is larger than some limit you set, and (2) write the incoming data to a temporary file (for files over 250k) during the upload rather than trying to store all the data in memory.

Here are three more cool features that don't really have specific code associated with them:

1. To create tests for your application (you are creating tests, right?), you can create a "t" directory and place your usual Perl tests there. Mojolicious provides a number of Web application testing helpers such as get_ok() (to fetch a Web page and compare the result) and status_is() (to test the response code). It also provides easy ways to parse an existing HTML document (more on this later) that can be used for a more precise test code.
2. If you want to capture a detailed log of your prototype application, it is as simple as creating a "log" directory. Mojolicious::Lite will start to write a log file into that directory with lines such as:

```
Mon May 30 16:51:48 2011 info Mojo::Server::Daemon:297 [20304]: Server
listening (http://*:3000)
Mon May 30 16:52:10 2011 debug Mojolicious::Plugin::RequestTimer:22 [20304]:
GET /login/dnb (Mozilla/5.0 (Macintosh; U;
        Intel Mac OS X 10_6_7; en-us) AppleWebKit/533.21.1 (KHTML, like Gecko)
         Version/5.0.5 Safari/533.21.1).
Mon May 30 16:52:10 2011 debug Mojolicious::Routes:376 [20304]: Dispatching
callback.
Mon May 30 16:52:10 2011 debug Mojolicious::Plugin::EplRenderer:57 [20304]:
Rendering template "login.html.ep" from DATA section.
Mon May 30 16:52:10 2011 debug Mojolicious::Plugin::RequestTimer:44 [20304]:
200 OK (0.004464s, 224.014/s).
```

3. And one more feature that seems obvious once you hear it is possible:

When you start up a prototype application in daemon mode (the one that spins up a test Web server), it can be started with a `--reload` flag, like so:

```
$ perl testapp.pl daemon --reload
```

When started this way, Mojolicious::Lite's test Web server will monitor your testapp.pl file for modification and reload itself with the new contents of that file if it spots any changes. With this approach, making a change, restarting the Web server, making another change, restarting the server, and so on becomes unnecessary and the interaction is much more pleasant. Using the Web server in this mode has some limitations (see the documentation), but most of the time it works great.

What Else?

Once again we are at a place where we've opened the door to peek at a subject but, for space reasons, don't have a chance to fully explore the magic land beyond that door. With Mojolicious this is especially true, because we've only scratched the basic prototyping layer it provides (Mojolicious::Lite). Mojolicious itself is a full-on Web framework that lets you build a full-bore MVC Webapp. You can split up the logic for your application into appropriately sized compartments (each in its own class/package/module). More advanced route handling can be specified to direct the program flow to specific handlers based on conditions (e.g., which browser is being used, invalid input), to ignore parts of the request URL, and so on. There are too many other lovely features to this package (including a cool HTML parser called Mojo::Dom which can use CSS-like selectors to return information) to cover in just one article. Oh, and I haven't mentioned all of the plugins available which make using back-ends like Redis, MongoDB, and Memcached from Mojolicious pretty painless. Set aside some time to look at the documentation at http://mojolicio.us. I think you'll be pleased if you do.

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

Galvin's All Things Enterprise

The State of the Cloud

PETER BAER GALVIN

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

It is with a tear in my eye that I change the name of this long-running column from Pete's All Things Sun to Galvin's All Things Enterprise. The tear comes from a longing for the good ol' days when Sun was a major contributor to technology innovation. With the purchase of Sun by Oracle, the use of the company name Sun no longer makes sense. And while I'm changing some things, why not change the scope of the column as well? In fact, truth be told, some of the previous columns weren't really Sun-focused, but strayed from the course to cover interesting technology topics that were just at the periphery of Sun. So at the prompting of Rik, the editor of *;login:*, both the name and the scope of this column have changed. Certainly, Oracle will be a topic from time to time, but there is a lot of innovation happening in the IT space, and with the new name, it will be within the purview of this column. But what topic should be first for this expanded column? How about the most hyped (or is that over-hyped?) new area of IT innovation: cloud computing?

Everything is cloudy these days. Between old-school vendors having (or claiming to have) cloud-centric products and services and startups that may or may not have the next great cloud thing, it's impossible in the IT space to avoid hearing, reading, getting marketing about, and generally being bludgeoned by cloud. In fact, the use of the word "cloud" in all things technical has caused some to come down with cloud-itis—one more mention of something "cloud" could cause a serious injury (or cause them to seriously injure someone).

Which of course leads me to write this column about cloud. But wait, didn't I just admit that the world is overly cloudy? Indeed. However, this column is not going to introduce some great new cloud thing that you didn't know you couldn't live without until I told you about it. Rather, this is intended to be a pragmatic cloud sanity check. What are IT managers doing about cloud, and what aren't they doing? What seems to be a cloud-based improvement on the old way of doing things, and what is just pie in the sky? And just what is "cloud" anyway? Read on to see my take on all things cloud.

What Is Cloud Computing?

Cloud computing is many things to many companies' marketing departments, and even analysts have trouble agreeing on what constitutes "real" cloud computing. For example, some definitions include some kind of remote access requirement, while others say that the component must be part of a shared infrastructure to be considered cloud. Gartner has the most sane definition [1]: cloud computing is "a style of computing where scalable and elastic IT-enabled capabilities are provided 'as a service' to external customers using Internet technologies." At a more detailed level, I believe a cloud-based solution has to include these aspects:

Elastic/on-demand/scalable—needs to have the ability to rapidly scale to meet potential reasonable demand

Service-based—facilities provided as services rather than ad hoc or fixed implementations

Shared—used by multiple entities concurrently: for example, multiple internal groups or external companies

Metered/monitored—either charge for use or monitored for use (chargeback or viewback)

Internet technologies–based—enabling potential access from anywhere; location independence; secure remote access; and, in general, the benefits brought by Internet technologies

These requirements allow for a general cloud definition to include both public cloud and private cloud versions. *Public cloud* is the more common and the first salvo in the cloud wars, but *private cloud* is also becoming a major player and needs to be part of any cloud discussion. Public cloud is a service provider making resources (CPU cycles, disk blocks, applications, etc.) available from its datacenter, meeting the above requirements. A private cloud solution is similar but executed within a company's datacenter, with the added qualification that it is under their security control.

What of the other forms of cloud that are making the rounds? *Hybrid cloud* is less a cloud form than a cloud strategy. Hybrid is the use of both public and private cloud computing. A given project, for example, might use both, or separate projects might involve one or the other. The consensus seems to be that, rather than there being a mass migration to the public cloud, for example, this hybrid form of cloud computing will be the mainstream for the significant future.

Another major cloud computing form is the *virtual private cloud*. In essence this form gives IT managers more control over the security and manageability of a public cloud solution by segregating part of the public cloud into a more private cloud-like environment. It is between public and private cloud in terms of costs, security, and manageability.

Those of us who are jaded computing veterans recognize in cloud computing many aspects of other previous-generation solutions. In fact, the Wikipedia entry about cloud computing [2] rightly points out that there are aspects of autonomic computing, client-server model, grid computing, utility computing, peer-to-peer, and service-oriented computing. I would go further and say that many of these ideas have been incorporated in thin-client computing, mainframe, online-service providers, and time-sharing computing. If you doubt this, consider that you could rebuild CompuServe (the old online-service provider) by buying CPU cycles from one public cloud provider and disk space from another.

Why Cloud?

If cloud computing is similar to previous computing models, what makes it different? Note that in the definitions given so far, there is nothing about lower cost. However, that is one of the driving factors. The lower cost comes from efficiencies of scale of the large cloud providers, and also from competition. Competition is the key difference between the old models and the cloud model. Not only are the cloud providers competing with one another for price/performance and features, but for the first time, they are providing competition to IT management. This is a seismic change. Never before could a business manager use her credit card to pay for infrastructure to host her new technology-enabled business offering, but that is just what is happening in companies worldwide. Note that this is not necessarily a good thing, as IT management is responsible for SLAs, governance, and policies. For example, a company might have a policy that states that "all tier 1 data must be replicated from the production site to the DR site"; such a policy would more than likely be disregarded by the business manager. Who will be responsible when a problem occurs because of the violation of this policy?

How, then, is IT management going to respond to competition? Some are fighting against it, issuing policies that say cloud facilities cannot be used by business components outside of IT. And sometimes this is for good reason, as discussed below. Others are letting nature take its course, enjoying the surprised look on the business manager's face when they get the unexpectedly large credit card bill ("How could $0.12 per hour add up to that?!"). Others are using that competition to their advantage, harnessing cloud solutions to lower their costs and increase the features, functions, capabilities, rapidity of response, and elasticity available to them by using cloud services to meet business needs. They are moving to provide IT as a service to their customers (employees at their business), and those services are frequently composed of hybrid cloud components.

Of course, with any new computing solution there are those who rush to be at the bleeding edge and those who wait for solutions to mature, see if they stand the test of time, and then use them as they fit into their IT strategy. Equally obvious is that not all cloud technologies, solutions, vendors, and products are right for every IT manager. VDI (Virtual Desktop Infrastructure) is the current case in point. Many sites are exploring VDI, but many are finding that it is proving difficult to execute at any cost savings over their current desktop architecture. Some sites are moving to VDI in spite of this, discovering that it solves other thorny issues such as security and easy (and secure) remote access. Others are evaluating new solutions as they appear, waiting for one that fills their set of needs at their needed price.

That leads to another interesting point about cloud computing—it is without a doubt driving a new, high level of innovation in IT. Startups are being funded to provide cloud solutions, and to provide solutions to the new problems that cloud computing is creating. Existing companies are revamping their existing offerings to make them more pertinent in the cloud epoch (or at least having their marketing spin stories about how their offerings are cloud-relevant).

What's Next

So far this column has defined cloud computing and has given reasons why it is important to IT. In the next edition of Galvin's All Things Enterprise, I'll discuss examples of companies making good uses of cloud solutions, I'll give reasons that others should be or are avoiding the cloud, and I'll finish with a detailed list of IT aspects to consider when deciding whether a given service should be public cloud-based, private cloud-based, hybrid strategy-based, or kept as is on current infrastructure.

References

[1] http://www.gartner.com/it/page.jsp?id=1035013.

[2] http://en.wikipedia.org/wiki/Cloud_computing.

iVoyeur
Crom

DAVE JOSEPHSEN

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

In 2006 I joined a tiny little company that was in the process of moving from the California Bay Area to Texas. Their business was about half database outsourcing and half niche Web site hosting. Their production infrastructure was primarily Linux, but they'd had all sorts of sysadmins, so there were a few pieces of SCO here, some HPUX there, etc.

The various sysadmins had also left their mark on the haphazard bowl of spaghetti that was their back-end processing automation. This unruly mob of code that extracted and imported, encrypted and decrypted, compressed and uncompressed and sent hither and yon the data that was the lifeblood of the company was written in all manner of languages, and never did the same thing the same way twice. When I joined the company no one had a clear idea of what it was all doing, much less how it managed to do it.

There were several hundred scripts in all, written in TCL, Perl, C, shell, and Java. The DBAs knew where to drop things off and where to pick things up, and beyond that nobody wanted to touch any of it. But now that the company was moving, it all needed to get untangled, and the untangling had fallen to me.

It's rarely much fun to inherit another sysadmin's (or, in this case, gaggle of sysadmins') mess, but I was actually kind of fascinated by the problem. It was pretty obvious that all of this stuff was doing the same subset of tasks over and over again. Extract the file, encrypt the file, send the file. Repeat. My plan was to write a library that encompassed all of those tasks, as well as enforce some standardization, and then re-write all of the existing scripts using that library. None of this was exactly rocket science, and transparency was important, so I took the LCD approach and wrote the library in shell.

It was a commendable effort. The library enforced a common runtime directory structure, so that everything was in a predictable place. It included its own logging functions to ensure that all of the logging and error-handling was centralized and in a common format. It even trapped signals and responded accordingly, such that if anyone were to, for example, hit Ctrl-C in the middle of a script execution, the library would gracefully exit. All of the scripts would use cron for scheduling. It solved a lot of the problems that the original bowl of spaghetti presented, and even changed the way I write shell scripts to this day (for the better), but ultimately, I think, the effort was a failure.

There are several reasons I think I missed the mark on this problem, but really they could be summed up by saying that I hadn't given the company a solution. I'd rewritten their automation in the manner I thought it ought to have been done in the first place, but for everyone other than myself, these scripts are still a black box of mystery. Were I to leave the company tomorrow, the admin to replace me would be more likely to write the next script in his or her language of choice (Ruby or Lua, or whatever you kids are using this week) than to dig into my code to learn how my boring, probably obsolete shell library worked. I hadn't added to the mess, but neither had I provided a means to ensure it didn't reoccur. And really that's why the problem existed in the first place.

Also, there were aspects of bad engineering about it. Yes, there was a library of reusable code there, and all those common tasks were represented as functions within it, but I still needed to port the old scripts to new scripts, and those new scripts all still did the same subset of things again and again. So there remained an abhorrent amount of silly code redundancy—100 scripts to call different combinations of the same 15 functions on 100 different files. Had I written a few proof-of-concept scripts instead of being so focused on finishing the magical library of wonder, I would have noticed it earlier. Once the lib was done, I'd ported about two TCL scripts to it before realizing my mistake, but by then I was committed to the design and nearly out of time. I paid for it in the mind-numbing 72-hour port-fest that ensued, feeling stupider and stupider with each newly ported shell script.

Needless to say, the seed of a mental image of the correct answer formed in my mind that night, but as these things go, it was a couple of years before I was able to revisit the problem. That seed had plenty of time to germinate, and I was determined to get it right this time. The library wasn't a bad idea at all, I just wasn't thinking big enough. The correct answer to this problem was, I think, just a single layer of abstraction up from where I'd started. I had written a library to enforce a common way to do things, but I needed a framework, and a set of common interfaces for people to use that library (in a way that didn't force them to write their own shell scripts). I call that framework "Crom." And while Crom is dry fodder for conversation, and only peripherally related to systems monitoring, it's also about all I've worked on for the last several months, so I'm afraid we're stuck with it, dear reader. My apologies.

Crom has a few operational assumptions about your job. First, it assumes that your job can be broken down into tasks. Next, it assumes that you want to schedule those tasks to run on a recurring schedule of some sort. Crom uses the UNIX `at` command to perform the actual job scheduling, and it is written in 100% shell, so it requires only /bin/sh, at, and the usual slew of shell commands like `date`, `cut`, `grep`, and `sed`.

Although the similarities are unintentional, Crom's architecture is quite similar to Nagios [1]. It's a task-specific scheduling and notification engine, but instead of scheduling little monitoring plugins to collect metrics or check availability, Crom schedules individual tasks that make up a larger Job. These tasks deal with some little piece of automation, like loading data into an Oracle database or sending a file via FTP to a remote host. Figure 1 shows a typical Crom job definition.

```
meta{
JOBID=4019
JOBNAME=exampleJob
DESCRIPTION="An example job for the wonderful readers of ;login magazine"
NOTIFYONERRORS='cromerrors@domain.com'
}
```

```
task0{
DESCRIPTION="extract the file from DB1"
TASKTYPE='extract'
SCHEDULE='1 0 * * 2'
SOURCE="$(cat ${CTL}/${JOBID}/db1schema)@DB1:"
ORA_PROC=$CTL/$JOBID/file_extract.proc
ORA_ERROR='halt'
}

task1{
DESCRIPTION="scp the file from coke"
TASKTYPE='pull'
SCHEDULE='runafter:0'
PROTO='sftp'
SOURCE='oracle@DB1.domain.com:/data01/outgoing/Post*'
DESTINATION=%NEXT%
SKEY="${KEYS}/oracle_DB1_dsa"
ARCHIVESOURCE='1'
}

task2{
DESCRIPTION="add a date to the filename"
TASKTYPE='custom'
SCHEDULE='runafter:1'
DESTINATION='%NEXT%'
SOURCE='%THIS%/Post*'
INCLUDE="${CUSTOM}/${JOBID}/rename.sh"
}

task3{
DESCRIPTION="sftp the file to xyz bank"
TASKTYPE='push'
SCHEDULE='runafter:2'
PROTO='sftp'
DESTINATION='dbguser@1.2.3.4:in'
SOURCE='%THIS%/Post*'
DKEY="${KEYS}/${JOBID}/xyz_dsa"
}
```

Figure 1: Crom job definition

Except for the surrounding brackets, each attribute is shell syntax. In fact, when Crom reads in some piece of the job definition, it does so by extracting the section it's interested in between the brackets via sed, and then sourcing it. The attributes are then available as shell variables. Since we source in the attributes, we can use nested execution blocks to hide sensitive info such as passwords. In the example, the SOURCE attribute in task 0 is reading in its Oracle schema and password from an external file using `$()`. Crom inherits its directory structure from my original back-end processing library and provides environment variable shortcuts to useful directories at runtime. These may be used by any script that sources the library. For example, task 0 in Figure 1 is using the ${CTL} shortcut provided by Crom to locate an Oracle procedure file.

Each job is identified by a unique job number, called the JOBID, and each task is numbered sequentially. All tasks are required to have a schedule. There are three valid types: (1) Crom can parse schedules in standard cron syntax using its own parser (also written in shell); (2) a task may be scheduled to be run subsequent to the successful completion of another task with the “runafter” keyword (any number of tasks may “runafter” the same parent task in parallel); and (3) Crom supports the “never” keyword as a valid schedule, for tasks that are never intended to be run automatically (such as break-fix or debug tasks).

The Crom library supports macros in its definition files for those variables that aren’t necessarily known at runtime. For example, task 1 in Figure 1 is making use of the “%NEXT%” macro, which will resolve to “cromhome/run/today/files/4019/2” (since 2 is the number of the next task). Since that specific directory is actually known at runtime, we could have just specified that instead of using the Macro %NEXT%, but the macro is preferable in that if the task is ever renumbered, %NEXT% will continue to work without modification. %NOW% is another macro supported by the library, which will resolve to the current time in seconds since epoch format. Users may specify their own macros by placing their values in a file named for the macro in Crom’s “macro” subdirectory. It’s common practice for tasks to share data with each other by setting up macros in the jobs macro directory.

Each task is required to have a task type, which is similar to a plugin in Nagios. Crom uses the TASKTYPE variable to locate the actual shell script to execute. Since task 0 in Figure 1 is specified as an “extract” task, Crom will check cromhome/bin for an executable named “extract”. If it finds “cromhome/bin/extract”, it will schedule an “at” job at the next occurrence specified by the tasks schedule passing the JOBID as argument 1 and TASKID as argument 2. Expanding Crom is as easy as writing a shell script and placing it in cromhome/bin.

Strictly speaking, the executable is not required to be a shell script: it can be any type of executable, and Crom will gladly schedule it for you. However, Crom provides a litany of useful shell functions for tasks that are shell scripts, such as functions for sourcing in the task definition from the job file, and job control and logging functions. A task that sources the Crom libs can, for example, call the “‘halt” function, which halts the current execution of the task, generates an error to the logs, emails the recipient list specified by the NOTIFYONERROR variable, and prevents any other tasks within the job from being executed or rescheduled. In fact, tasks that source the Crom libs may make eight function calls relating to job handling and logging alone: debug, notify, info, stop, warning, error, nonfatal_error, and halt.

If you have a Java program and want to run it as a task under Crom, the wiser thing to do is to use the “custom” task, which takes the name of a shell script as an attribute called INCLUDE. The custom task will source in the script specified by INCLUDE and will check for a function therein called “runCustom”, which it will run. This way, we don’t need to port our Java code, but we retain full functionality with the job control system at the cost of only a few lines of shell. The custom function can also be used to build tasks that are not easily encompassed by a more generic task type. Renaming a file and setting up a custom macro for a subsequent task to use are good examples.

Crom was written with the expectation that large parts of it would be ripped out and replaced wholesale. For example, it currently reads in its job definitions from files in the cromhome/jobs directory, but the plan has always been to replace the jobs directory with an Oracle database table. The library is, therefore, modular and extensible, and just nice to work with to an extent I find difficult to articulate. Perhaps the best evidence of this is the fact that I find myself using it to write things, such as supporting tools, that I normally wouldn't bother with. Figure 2, for example, is the output of the "cq" tool, which uses library function calls to summarize the scheduling queue. I usually lose interest and move on long before I'd consider writing something like this.

```
JOBID,TASKID    ATID      SCHEDULE
----------------------------------------------
4007,0          2420      Mon May 30 01:00:00 2011 a crom
4007,2          2422      Mon May 30 01:01:00 2011 a crom
4007,4          2425      Mon May 30 01:04:00 2011 a crom
4007,6          2426      Mon May 30 01:10:00 2011 a crom
4007,8          2427      Mon May 30 01:12:00 2011 a crom
4008,0          2309      Tue May 31 08:00:00 2011 a crom
4008,3          2334      Wed Jun 1 08:00:00 2011 a crom
4010,0          2415      Mon May 30 00:15:00 2011 a crom
```

Figure 2: Output from cq, the Crom queue command

Most core functions, and every default behavior in the lib, are overridable by defining a custom function or setting an environment variable. Several of the supporting tools I've written, in fact, make use of override variables. The runtask script, for example, is what I use for manual intervention when something goes wrong. This script takes a JOBID and TASKID as its arguments and uses them to force-run that task immediately, overriding states like halt, which would normally make the task refuse any attempt at execution. Even within the default tasks we've written, the error handling behavior is usually overridable. Task 0 in Figure 1, for example, specifies an ORA_ERROR variable, which is there to override the default behavior of the extract task when it encounters an error running sqlplus (changing it from its default value of "error" to "halt").

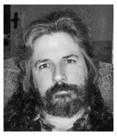
Crom can log to a flat-file log (which rotates daily), syslog (to a user-configurable facility and priority), a FIFO (which we use to push log lines into a database with a separate script), or any combination thereof. The log lines contain all of the information you'd expect, plus a few fields I find especially useful at times, such as the "run number" field, which uniquely identifies each iteration of a task that runs, for example, every other minute all day long. Crom has built-in functions for sending email notification and automatically notifies recipients when a task calls warning, error, nonfatal-error, and halt.

Well, thanks for letting me gush over my shell script. This is about the only tool I've written where I have no doubt I'm reinventing the wheel and it's working so wonderfully I just don't care. It's also the kind of tool that's esoteric enough that I'm not sure if I'm scratching an itch that nobody else has (but again, it's still working so wonderfully that I don't care). If so, I've probably just bored you to death. Sorry about that. Stay tuned for something more monitoring-related next time.

Take it easy.

/dev/random

ROBERT G. 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

The past few weeks have been an adventure for me, with a pinched nerve, lots of doctor visits and tests, and other less than pleasant artifacts of the aging process taking center stage. Parallel to this evolving health drama was an employment-related one. I decided that I needed a new job, and so I put out some feelers, one of which came back positive. I flew out for the strangest interview of my life and on the plane ride home I started thinking about the job interview process in general. I've sat through probably four dozen interviews in my career—on both sides of the table—so I have at least a moderate sample size from personal experience.

I have reached the conclusion that there are really three main categories of interviews: *Traditional*, *Avant-garde*, and *Out There*. In the "traditional" interview they ask questions like, "Where do you want to be in five years?" and "What is your greatest strength/weakness?" The "avant-garde" approach is more relaxed and concentrates on esoteric questions such as, "If you had an infinite research budget, what projects would you work on?" or "You're going to be stranded on a desert island and you can take three tools to help start your own business. Which tools do you choose?" These are a little more creative and geared toward finding employees who think, rather than merely prattle platitudes.

Way beyond the boundaries of custom and best business practices you'll find the "out there" interview. This tragicomic creature is of fairly recent origin and probably a chimera engendered by the collective brain damage of the sixties combined with the corporate insanity that was the dot-com era. The questions you'll encounter in these freak shows are hard, nay impossible, to predict, but I'll give you a general idea. "A man walks into a grocery store and asks the produce clerk for two kumquats and a kiwi fruit. How many pairs of shoes does he own?" "A major hurricane will be hitting your hometown in 24 hours. The local hardware stores are sold out of plywood. Is the school board mostly liberal or conservative?" "Why do men's briefs have a fly in front, but not in back?" "What would ecru smell like?" My advice if you encounter a firm that employs the "out there" interview is to answer in some equally surrealistic fashion and hope they don't try to hire you. Performance reviews in a place like that are bound to be something akin to verbal waterboarding.

I would very much like to say that this segues neatly into my next subject, but that would be a filthy lie because they are no more closely related than an OS vendor's earnings report is to their commitment to releasing secure, thoroughly tested code. Entitlement, ladles and gentlemints, is what I wish to discuss. Past generations, even to a certain extent my own, which is sandwiched somewhere between long-haired hippy peaceniks and Izod-touting yuppies-to-be (really I'm Watergate/disco-era, but I avoid that admission like the pathogenic ruin-every-decent-thing-it-touches plague it was), placed a significant emphasis on the concept of sweat-equity. That is, we encouraged people to scrabble their way up from humble beginnings to positions of influence and affluence by hard work and honest dealings and lionized those who did.

Now, I'm perfectly well aware that every generation thinks the one preceding it tried their darnedest to destroy the world and the one following is soft and pampered and expects everything to be handed to them on a silver platter, but there really is *something* screwy going on here this time. Perhaps it's an artifact of our ever-increasing standard of living, perhaps merely the inevitable result of the confluence of pervasive social media and twenty-four-by-seven inundation by tidbits covering every conceivable aspect of the lives of the world's celebrities, but whatever the contributory factors, there are a great many people under 30 out there who seem to think that success and all that goes with it is somehow magically *owed* to them.

I hate to be yet another bearer of inconvenient truth, but it simply ain't so. If anything, you young'uns may actually have to sweat even more profusely than my lot did, because the economy is in the tank. If at first you don't succeed, that doesn't mean you need to sue someone for damages, though: just try again. You'd be surprised how often that works. When I hit the job market after grad school, of course, we were in the Reagan years and the young revolutionaries weren't worried about much beyond whether to go with the six- or 12-month CD. My generation bridged the awkward gap from sit-ins to youth investment seminars.

I will close this outrageously circuitous ramble by addressing a rather well-known multimedia products vendor's recent rash of security embarrassments. As of this writing there have been nine—that's right, nine—separate incidents connected with this single "mesoscale hack." With any complex product, or in this case suite of products, the occasional lone security vulnerability exploit is understandable, perhaps even inevitable. But this chorus line of gaping flaws is a little beyond the pale, even for our insecurity-apathetic information technology culture. One difference here, though, is that while an American or European company might slough off even an insult of this magnitude as a cost of doing business (which, by the way, it definitely is not, or should not be, anyway), the corporate culture of this particular firm will almost certainly require at least one ritual sacrifice at the executive management level. A noble, if somewhat futile, gesture in a decidedly un-noble period of economic history.

Contrast that with what happens when, say, a major US defense contractor experiences a massive exfiltration of unclassified but highly proprietary military weapons technology data. There is an internal investigation launched, Congress and everyone else concerned is reassured that the problem has been handled and effective damage control measures have been taken, and within a ridiculously short period of time the issue just fades to black. Stock prices aren't even affected over the long term, for Pete's sake. No real consequences always equates to no real corrective steps. Ring around the rosy, pockets full of apathy.

By the way, I didn't get that job—but that's OK. It was definitely an "out there."

Book Reviews

ELIZABETH ZWICKY,
WITH RIK FARROW

R Cookbook

Paul Teetor

O'Reilly, 2011. 395 pp.
ISBN 978-0-596-80915-7

Finally, an R book that actually answers the questions I need to have answered in a way I understand. It answers a lot of other questions, too, ranging from ones where I think, "Ah! I've wanted to do that!" to ones where I think, "I hope I never want to do that."

The problem with R is that almost everything you want to do is very easy as long as you think about it correctly, for R. Which is at right angles to everything you are used to. (For instance, loops are evil. Never loop. You apply the function to an array—well, no, not an array, but unless you already know R you're going to think of it as an array—and it mystically does the right thing.)

Since books about R are written by people who can think in R, they often make sense only after you have managed to make this right angle turn. Somehow Paul Teetor has managed to maintain more perspective than most authors.

The cookbook format is sometimes constraining and artificial ("Problem: You want to install R on your computer"), but once you get past the beginning it generally works OK. Because of the aforementioned otherworldly nature of R, you may find that you have to read parts of it straight through. If you think a vector has a direction and a length (instead of being basically a one-dimensional array), you're not going to be able to leap right to solving your problem. That's not the book's fault; even real cookbooks end up assuming that you know how to boil water at some point.

If your needs for statistics have outgrown your favorite spreadsheet, but you are not a career statistician, you need R, and you probably need this book to go with it.

Everything Is Obvious Once You Know the Answer

Duncan J. Watts

Crown Business, 2011. 324 pp.
ISBN 978-0-385-53168-9

(Full disclosure: Duncan Watts and I share an employer, currently. I've therefore been exposed to more posters for this book than most people, but I believe it has not otherwise influenced me. It does not appear to me to have made any of my colleagues less willing to tell me what they believe to be obvious, sadly.)

I like this book for a couple of reasons. First, I have a strong tendency to believe that everything is probably complex and unintuitive and one ought to measure things before believing in them. Second, I find reading it brings up the basic emotions of watching a high-wire trapeze act. Here is somebody writing an entire book which can be summed up as, "Sociology has so far told us very little about the world, often because the answers to the questions people are asking are in fact unknowable, but it's worth doing anyway." And it's an engaging and I think convincing book. I am in awe at even trying to do this, much less succeeding.

Another way of summing the book up, of more interest to people who hold no strong opinions about sociology, is, "Stop guessing what people are going to do and why, and start reacting to what they actually do." You think sociology is irrelevant? Marketing is applied sociology, and most of it is the sociological equivalent of avoiding black cats and the number 13. That might be funny, except it costs real money.

If you are interested in prediction of human behavior, or sociological ideas such as "everybody is only 6 steps apart" or "some people are influencers and sway the opinions of lots of other people," this is an interesting new take on things. As for me, I plan on keeping it around to whack people with when they tell me how obvious something I've just found out is. Or how something they have just made up is obviously true.

IT Security Interviews Exposed

Chris Butler, Russ Rogers, Mason Ferratt, Greg Miles, Ed Fuller, Chris Hurley, Rob Cameron, and Brian Kirouac

Wiley, 2007. 205 pp.

ISBN 978-0-471-77988-2

Periodically, I wander off to see what's in bookstores. This looked interesting because I've been interviewing candidates recently (for security, but not specifically IT security) and I wanted to see what advice they might be getting, what questions other people use, and whether there were resources that would help candidates avoid popular mistakes.

This book does steer people away from some common and unfortunate mistakes (e.g., it's a resume, not an autobiography—six pages is overkill, particularly if you've only held two jobs). And on most topics, it gives an overview sufficient to help an experienced person think about what areas they might want to brush up on, and what answers they might want to think out.

On the other hand, the years have not been kind to it; there is no mention of Web security at all, and I'm pretty sure I'm one of the few security interviewers on earth who doesn't ask about XSS. I know this because all my interviewees, if they don't know an answer, say hopefully "I think it's an XSS," regardless of the question. (Hint: if we are not discussing Web sites or I have just told you what I think the problem is, that is not the right answer.) A book that doesn't even get into the general vicinity of Web servers (XSS, XSRF) or database servers (SQL injection) is omitting some of the most important and interview-relevant topics in security.

Aside from that, it's inconsistent, with different format and tone for different chapters, the discussions are telegraphic enough to permanently confuse somebody who doesn't understand the territory already, and it rarely gets into questions that distinguish interviewing from exams. Somebody who could answer all the sample questions correctly would come across as somebody with a CISSP and nothing behind it. If they were lucky. Otherwise they might have picked up some of the book's more perplexing stumbles. No, it is not easy to ensure that no wireless client on your network is willing to connect to a rogue access point. No, it is not important to believe that HIPAA is regulation and Sarbanes-Oxley is legislation. (They are both legislation, implemented as regulation, but I had to look that up; nobody cares unless you carefully and definitively get it wrong.) And if I ask you about the main configuration components in a firewall, and you reply "configuration, policy, and objects," I am going to believe you know exactly one firewall configuration system.

Take Control of Media on Your iPad, 2nd Edition

Jeff Carlson

TidBITS Publishing, 2011. 158 pp.

ISBN 978-1-61542-131-2

I was curious about what this series was like, and this seemed like a good topic for a review: simple enough that I can evaluate it, without being so simple as to be mind-numbing. Also, there's something pleasantly ironic about reviewing a book about media on the iPad, as an eBook on an iPad. (It turns out that while it is pleasantly ironic, it makes it impossible to view the instructions and the interface at the same time. This worked out OK for me, but a real novice probably needs a paper copy, or at least to read the book on some other device.)

It was a good experience, all told. The eBook version is formatted to take advantage of the platform, which is rare and convenient, and there's appropriate coverage of the built-in capabilities without totally neglecting the important add-ons (such as GoodReader, the reader I was in fact using to read the book). It told me several things I didn't know and wanted to, and it looked quite useful for its intended audience of basically competent users who may need some help. It was willing to point out useful trivia (how to lock the orientation of your screen and adjust the brightness) without devoting a lot of space to things most people will know.

If you're a contented and knowledgeable iPad user already, it probably won't improve your life by $15 worth (although it did improve my iPad life a bit). On the other hand, it might well be worth it to my father (it certainly would have been before the day I taught him to use smart playlists in iTunes). Although it is a new edition, it is still applicable to first-generation iPads as well as the iPad 2.

The Book of PF, Second Edition

Peter Hansteen

No Starch Press, 2011. 188 pp.

ISBN 978-1-59327-274-6

Tony del Porto reviewed the first edition of this book back in the April 2008 issue of *;login:*, and I was interested in seeing what had changed since then. PF is the OpenBSD firewall and is also available in FreeBSD and NetBSD. PF is configured through a powerful and concise set of rules, and some of the syntax of the rules changed with the release of OpenBSD 4.7 (and FreeBSD 8). And while PF already included support for CARP used for failover, the new version also supports CARP for load balancing.

PF has had the ability to create dynamic rules, something just added to Linux, and you can do some very cool stuff with this. On page 87, you learn how to add the IP address of someone attempting to brute-force SSH to a rule that will block that IP address based on the number of simultaneous connections and the rate of connections.

PF supports IPv6 without the need for a separate configuration file and command, unlike Linux. What is lacking from this book, and the online OpenBSD PF pages, are examples of firewalls using IPv6. In a way, this is not a problem, because simply enabling the routing of IPv6 (in a gateway firewall) is all that is needed to make your existing PF firewall work with IPv6. But there are some things specific to IPv6, such as ICMPv6 (required for determining the MTU, for example), for which examples would be nice.

But this is a clearly written book and well worth the price.

—Rik Farrow

The Silicon Jungle

Shumeet Baluja

Princeton University Press, 2011. 334 pp.

ISBN 978-0-691-14754-3

I was intrigued when a friend mentioned that a book written by a Google employee, about massive data processing, was an exciting novel. And I found myself unable to put the book down as it neared its end. The characters had taken on lives of their own, ones that appeared familiar to me. The big exception here is the antagonist, a greedy multimillionaire, who seemed a bit too simplistic. Then again, I know so few multimillionaires that perhaps Shumeet is being totally accurate here.

This is a story of an intern at a fictional Internet search company which sounds like Google merged with Amazon, Visa, and AT&T. The tech culture is very Google, with all the free caffeinated drinks and food you can ingest, and people working long hours on an eerily familiar campus in Silicon Valley. But this company has access to a much broader swath of data than any company in the world has today. That data, and the ability to process it, is key to the plot.

Stephen, the young intern, lands a highly competitive internship and is selected to work on the hottest internal project, one that can connect information from credit card purchases, email, phone calls, and Web searches to target advertising more precisely than ever before. The interns are an experiment in just how usable the interface to the new software will be, and the interns are soon invading the privacy of unsuspecting people. Stephen, a veteran of a failed startup, has a bit more maturity, but he too gets caught up in the power of the system when he creates a list very similar to one of the US government's terrorist watch lists.

Stephen's girlfriend, Molly, is instrumental in getting him involved with the search company and has her own ties to terrorism, via the unusual path of a doctoral dissertation funded by the DoD. Molly and one other female character add a bit of a balance to this otherwise all-male geek world.

Shumeet's novel is also a speculative tale of what can happen when an organization, whether it is the government or an advertising agency, has access to too much information and the means to process it. The story strongly reminded me of Admiral John Poindexter's failed attempt to create a similar information gathering project, coined Total Information Awareness, back in 2002. With proliferating automated license plate recognition and RFID toll payment devices, and the information available from cell phones (location, contacts, searches), airline reservations lists, and credit cards, TIA would do an even scarier job today. Shumeet is writing about this issue, even as he makes reading about it fun. Yes, there are 16 pages of references at the end, too.

—Rik Farrow

CONFERENCES

Conference Reports

In this issue:

Peter Bailey receiving the Computer Research Association Undergraduate Researcher of the Year award from Matt Welsh at HotOS XIII

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

Napa Valley, California
May 9–11, 2011
Sponsored by USENIX, the Advanced Computing Systems Association, in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

Opening Remarks

HotOS XIII Program Chair: Matt Welsh, Google

Summarized by Rik Farrow (rik@usenix.org)

Matt Welsh opened the workshop with a quick description of how it was structured. Each speaker had only 10 minutes for his or her presentation, with five minutes allotted for questions. Participants could interrupt the speaker during that 10 minutes. At the end of two sessions (five or six presentations), there would be a 45-minute discussion session, where the topics might involve the previous presentations, or anything else that was relevant.

At the end of the workshop, Matt announced some awards. (Yes, I know this is putting the cart before the horse, but I am not certain you will notice these announcements unless I put them up here.)

Matt Welsh, who came from Harvard to work for Google, announced that Peter Bailey, with whom he had worked for eight years at Harvard, had won a Computer Research Association Undergraduate Researcher of the Year award, which includes a 500-pound marble obelisk that had already been delivered to Peter, a certificate, and the support to attend the conference of his choice. One of Mike Freedman's students also won a CRA award this year.

Matt then told us who had won Google Chromebooks by their workshop presentations; Vijay Vasudevan (CMU) with his poster that took a position against the paper he presented; Dave Ackley (U New Mexico) for the most outrageous opinion, best expressed in person, but his paper does nearly as well. They decided to give two best talk awards, one to Mike

Walfish (U of Texas, Austin), who used Prezi, and one to Chris Rossbach (MSR), who got the remaining Chromebook, since Matt thought it would be easier to ship one to Austin than to Microsoft Research.

Putting the Hard Back in Hardware

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

Mind the Gap: Reconnecting Architecture and OS Research

Jeffrey C. Mogul, HP Labs, Palo Alto, CA; Andrew Baumann, Microsoft Research, Redmond, WA; Timothy Roscoe, ETH Zurich; Livio Soares, University of Toronto

Jeff Mogul kicked off the workshop with a talk about a paper that arose from the "Research Vision" session at OSDI '10. The problem is that the computer architecture and OS research communities are drifting apart. New architectures are developed with little regard for the OS, which is considered to be so unknowable that it is a source of "noise" in benchmarks. This is largely due to the gold-standard benchmarks—such as SPLASH, SpecCPU, and PARSEC—which run almost completely in user mode for an extended period of time. By contrast, the state-of-the-art for measuring OS performance on an architecture is limited to little more than system call and page fault micro-benchmarks. A few semi-realistic benchmarks do exist, including SPECWeb and TPC-W, but they don't capture the full variety of applications that run in a realistic system.

The OS researchers in the audience weren't immune to Jeff's criticism. Our unquestioning dedication to developing systems that run on "commodity hardware" means that we are missing the opportunity to ask for new features. If we don't ask, we will end up with features that work when running a single HPC application but are incompatible with the isolation properties that an operating system must provide. For example, a platform might provide low-latency message-passing support using shared memory buffers, but sharing such a facility between multiple processes requires a kernel entry, which effectively erases the latency benefits. Leading by example, Jeff then presented his desiderata, which include cheap inter-core messages, lightweight inter-core notifications, faster syscalls, software-controlled caches, and better performance counters. A common theme was that the architecture shouldn't bake in policy (such as cache coherency) without providing the developer with an escape hatch to try different approaches.

Mike Swift (Wisconsin) opened the Q&A by asking how the OS community could improve its review process for papers that suggest architectural changes. Jeff replied that it is often difficult to assess whether or not a change is feasible, and the purpose of ASPLOS was to have a program committee with a range of experience to aid this. Gernot Heiser (UNSW/NICTA) remarked that some of the architectural critique—of IPIs, in particular—was x86-specific, since ARM and MIPS don't suffer from all of the same problems. John Ousterhout (Stanford) asked how we can make incentives for architecture people to make changes, and pointed out that we should be careful to distinguish between the research community and the people who actually build the hardware. Jeff pointed out that one of the hurdles is that new architectures usually need to run a commodity OS, so there is a chicken-and-egg problem. Finally, Erez Zadok (Stony Brook) lamented that many architectures have a wide range of performance counters in hardware, but OEMs selectively disable many of them in the BIOS. Jeff remarked that some OEMs might be receptive to changing this.

Operating System Implications of Fast, Cheap, Non-Volatile Memory

Katelin Bailey, Luis Ceze, Steven D. Gribble, and Henry M. Levy, University of Washington

Katelin Bailey said that the real-soon-now advent of fast, cheap non-volatile RAM (NVRAM) may have a disruptive effect on OS design. Many of the assumptions in current OS design are based on a two-level memory hierarchy of fast DRAM and slow disks; NVRAM threatens to shake things up, because it potentially combines the speed of DRAM with the persistence of disk—i.e., it offers the best of both worlds. Existing research has focused on incremental steps, such as replacing disk with NVRAM and retaining file system semantics, or using virtual memory to build single-level store that combines RAM and DRAM. But this isn't radical enough: how about replacing *all* of the memory in a system with NVRAM?

Such a system would have many desirable properties. For example, hibernation and reboot would become extremely efficient, because there would be no need to copy state to or from a secondary storage medium. The very fast write performance would also make deterministic record/replay techniques much more practical. However, there are a number of challenges that would need to be addressed: for example, if your entire system image is persistent across reboots, how would you deal with bugs and rolling back to a known-good state? How should sensitive data be treated, now that it could persist for a much longer time? Furthermore, current virtual memory techniques were originally developed with the dual role of enabling swapping (which is no longer necessary) and protection (which is), so the development of a system with

NVRAM everywhere would provide a good opportunity to rethink the assumptions about granularity, for example.

Katelin admitted that the talk raised more questions than it answered, but the audience was on hand to raise even more. John Ousterhout (Stanford) harked back to the 1970s, when every computer effectively had NVRAM in the form of core memory, and he pointed out that nothing changed when DRAM displaced core. Katelin replied that it's still worth exploring our options. Mike Swift (Wisconsin) and Joe Tucek (HP Labs) raised the smartphone question, asking what we could learn from those platforms, but Katelin said that the approaches taken on those devices are relatively conventional. At this point, a waggish audience member pointed out that cell phones reboot every time daylight savings time happens, so they're not there yet. Mothy Roscoe (ETH Zurich) went Back to the Future, pointing out that many of these ideas had been tried before in systems like KeyKOS and Multics, but they hadn't caught on. Katelin said he hoped that fast NVRAM should enable us to do things that weren't possible in those days.

Virtually Cool Ternary Content Addressable Memory

Suparna Bhattacharya, IBM Linux Technology Center and Indian Institute of Science; K. Gopinath, Indian Institute of Science

Suparna Bhattacharya rounded off the hardware session by discussing another exotic form of memory: ternary content-addressable memories (TCAMs). Their associative addressing means that TCAMs have seen a lot of use in caches and high-performance routers, but more exotic uses have been discovered, such as encoding deterministic finite automata, ternary Bloom filters for subset matching, and similarity search algorithms. With progress at this rate, we can expect the range of applications to grow to the point where application developers may want to harness TCAMs, so this talk looked at ways that virtual memory techniques could be used to provide the illusion of vast amounts of associatively addressed memory.

It isn't feasible to build a single huge TCAM, because power consumption and latency increase with the number of keys. Therefore, Suparna discussed various ways that a virtual TCAM could be built from a combination of TCAM and DRAM. The basic idea is to build a cache hierarchy, with the level-1 store implemented in a TCAM and the level-2 store simulating associative lookup in DRAM. The first challenge is choosing a replacement strategy (and an application) that exploits temporal locality, so that as many lookups as possible are served from the TCAM. Spatial locality is less important (since in an associative store, location is not important), but there are some potential wins to be had by compressing keys using the don't care bits. This approach is particularly beneficial if there is "content locality," i.e., keys in a similar range frequently accessed together. Finally, it will be challenging to build an efficient TCAM simulator that uses DRAM, since the don't care bits mean that standard search algorithms do not apply; one possibility is to use part of the TCAM to store a mapping from partitions of the key space to DRAM locations. Suparna ended by echoing the previous talks in this session: it would be useful to engage the architecture community in order to develop TCAMs that are better suited to general programming—for example, by providing better support for multiple matches. She also speculated that the availability of NVRAM might open up new possibilities for TCAMs.

Jeff Mogul (HP Labs) asked how Internet routers—which must store the entire routing table—deal with a limited amount of TCAM, and whether they use similar techniques. Suparna replied that most routing tables try to do static compaction using the don't care bits, but she agreed that there may be tricks that could be picked up from these devices. Mike Freedman (Princeton) asked about applications and whether in this model TCAMs would be part of the general-purpose memory hierarchy. Suparna replied that the intention was to expose (virtual) TCAMs to applications as general-purpose memory, and that there were many search-based applications—for example, in data mining—that could benefit. Matt Welsh (Google) brought the session to a close by remarking that GPUs had become commonplace thanks to 3D gaming, and TCAMs might have a similar "back-door" application that pushes them into widespread use.

Soft Fluffy Clouds

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

The Best of Both Worlds with On-Demand Virtualization

Thawan Kooburat and Michael Swift, University of Wisconsin—Madison

Thawan Kooburat enjoys the advantages of virtualization, but he's concerned that its constant overhead is inhibiting adoption, especially in large datacenters at Google and Facebook, and on resource-constrained devices like your laptop. The idea of "on-demand virtualization" is that you only pay the cost of virtualization—both in terms of performance overhead and limited functionality—when its features are going to be used. Therefore, most of the time the operating system uses native execution, then it slips into virtualized mode *on demand* when the user wants to migrate execution, checkpoint the system state, and so on.

Thawan described how on-demand virtualization is implemented. The basic technique is to use the OS hibernate function (implemented using the TuxOnIce patch to Linux 2.6.35) to create an image of the system state, and then transfer that state into a virtual machine (implemented using KVM). One challenge is that the native and the virtualized hardware profiles will likely be different, with the VMM typically providing a feature set that lags behind native functionality. This is addressed with device hotplug and another level of indirection: logical devices that retain all necessary state and hide the hotplug events from the applications that use these devices. Thawan has a prototype that currently supports one-way conversion from physical to virtual, which takes approximately 90 seconds and succeeds without closing an open SSH connection. Future improvements will include hibernate-to-RAM, which will improve performance, and performing the virtual to physical conversion.

Mike Schroeder (Microsoft) asked if this defeated the purpose of virtualization as a means of providing a defense against security issues. Thawan replied that this is not the aim of on-demand virtualization, which is geared more towards migration and checkpointing. Peter Honeyman (Michigan) asked where to expect the crossover point when the cost of re- and devirtualization becomes greater than the cost of running permanently on a VMM. Thawan answered that it would be workload dependent. Philip Levis (Stanford) raised a concern about what would happen when migrating an OS that used a large amount of local storage on native disks, and Mothy Roscoe (ETH Zurich) pointed out that a paper at the last HotOS had solved the apparently harder problem of migrating between two physical machines with no virtualization involved.

Repair from a Chair: Computer Repair as an Untrusted Cloud Service

Lon Ingram, Ivaylo Popov, Srinath Setty, and Michael Walfish, The University of Texas at Austin

Michael Walfish is dissatisfied with the status quo in computer repair. Today, it resembles television repair, whereby you bring your computer to a retail service that is both inconvenient and insecure. Solutions based on providing remote desktop access are not ideal, because you have to monitor every action by the repairer, or it will be just as insecure as taking your computer to a shop. In this talk, Michael presented "repair from a chair," which uses virtualization technology to make the software components of a computer available to a repairer in a secure fashion. An in-depth study of Geek Squads, Genius Bars, and IT services at UT Austin revealed that the vast majority of repairs are software-only, and so this would be a feasible solution.

The system includes a module called the "repair helper," which lives between the OS and the hypervisor to facilitate repair. According to the paper, the main function of the repair helper is to migrate a copy of the VM to the repairer with private data scrubbed; there is a large design space to explore here. In the talk, Michael focused on the idea of using "action graphs" to represent the changes made by a repairer, and hence provide integrity guarantees. The hope is that repairs could be encoded in a canonical representation, which could then be signed by the repairer for assurance and auditing purposes. The action graph representation would also help to maintain availability of the machine while under repair: the customer could continue to use the machine, and changes by the customer and the repairer could be merged using a process that is analogous to git rebasing.

The talk provoked a lot of discussion and was awarded one of the Best Talk prizes at the end of the workshop. Mike Swift (Wisconsin) was first up to ask whether on-demand virtualization (from the previous talk) would be ideal for this. He also had a real question about what fraction of repairs would be difficult to handle, and how hypervisor device driver problems might be handled in the cloud. Michael replied that configuration errors would be in scope, but he hadn't considered hypervisor issues, since it was assumed that the customer wouldn't (or wouldn't be able to) mess with the hypervisor configuration. Jeff Mogul (HP Labs) took a different tack, suggesting that, if all the repairs were canonical and could be signed, the repair service could just apply all known repairs indiscriminately. Michael countered that there might still be some human intelligence required to choose the correct ordering. Then Jeff raised the specter of having to trust "canonical compositions," but Michael replied that this is not necessary if there is an auditable log. Finally, Brad Chen (Google) characterized this as an "automatic update" problem, and asked whether this would cease to be a problem when applications are cloud-based. Michael replied that as soon as devices become used for content creation, rather than consumption, configuration issues will start to arise again.

This marked the end of the formal Q&A, but this talk was the subject of much debate in the discussion/open mike session that follows below.

Structuring the Unstructured Middle with Chunk Computing

Justin Mazzola Paluska, Hubert Pham, and Steve Ward, MIT Computer Science and Artificial Intelligence Laboratory

Justin Mazzola Paluska gave an intriguing talk about a new construct that promises to unify parallel programming for GPGPUs, massively multicore systems, clusters, and clouds. At present, the structures used to represent programs and the structures of different execution platforms are orthogonal, and unstructured assembly code does a poor job of åtaking advantage of different, very specialized machines. "Chunks" are the solution: a chunk is a fixed-size block in memory that abstracts program structure, and chunks are mapped individually onto machine structure. Each chunk has a fixed number of slots, each of which is fixed size. Each slot is typed, and it can contain a scalar value or a link to another chunk. One idea is that making links explicit exposes structure in the chunk graph, and the developer is forced into this by the relatively small size of a chunk.

The chunk graph creates many opportunities and challenges for improving parallel programs. First, a link is allowed to cross architectural boundaries, and chunks can migrate between processing elements, which helps in a heterogeneous multicore system. However, this creates a distributed garbage collection problem and requires a policy to decide which chunks should be migrated. Another feature of the model is that threads start out being represented by a single chunk with a link to a (possibly linked-list) stack of chunks, which in turn may be linked to function chunks or object chunks. The links can be used to compute a distance and size metric within a given thread, which helps the system decide which chunks should be co-located. For example, a distance-k neighborhood of the thread object would indicate the important chunks to co-locate, and overlapping neighborhoods would enable synchronization and contention to be inferred. The main hope, however, is that there will be many distant threads that can run without interference and can be scheduled to avoid false sharing and contention.

Dave Ackley (New Mexico)—who would go on to make a name for himself at the workshop with an outrageous programming model of his own—was concerned about the chunk graph turning into a "huge ball of high-dimensional goo." Justin countered that unused chunks would not need to be loaded in, and NVRAM could be useful to help with this. Aleks Budzynowski (UNSW/NICTA) was more worried about the amount of policy that seemed to be going on at the OS level, and would prefer to see more work being done at the language level or in the compiler. Justin replied that this is another way to experiment with the same issues, and the chunk model is an attempt to force the compiler into giving the OS something to which it can usefully apply policies. Dave Holland (Harvard) saw this as a graph clustering, which is a known hard problem, but Justin replied that hopefully he doesn't have to solve the general problem, if it is possible to use some heuristics at runtime, such as sending paths around. Finally, Mothy Roscoe (ETH Zurich) was unconvinced that there is a one-size-fits-all solution for the huge number of different scales, but Justin said he'd had positive experience with cloud and cluster computing (which is the easiest experimental platform). The reason for a one-size-fits-all solution is that Justin had seen schematic pictures resembling chunk graphs over and over in different venues, and he wanted to extract some common abstraction that could be useful.

Discussion/Open Mike

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

By now the audience was fired up, and Matt Welsh (Google) opened the floor to anyone with something to say. Matt used chair's prerogative to make the first point about NVRAM: he likes the ability to wipe a computer's memory on reboot, because it's the only way to get it to a known-good state. Katelin Bailey (Washington) replied that rebooting wouldn't go away in the non-volatile future, but the aim was to separate the notion of resetting from the power cycle. Jeff Mogul (HP Labs) pointed out that this is a perfect example of decoupling mechanisms that don't belong together, as he had proposed in the first talk. Dave Andersen (CMU) was worried about the effect of random bit flips, but Margo Seltzer (Harvard) said that these are very unlikely in practice.

Geoff Challen (SUNY Buffalo) remarked that it was good to see many hardware people in the audience, which should help to address Jeff Mogul's criticism that the communities don't talk anymore. Mark Hempstead (Drexel), a self-confessed computer architect, announced that it was great to see a move towards better communication between the communities, and he asked people to send him C code that he could run. Mark raised a bone of contention: his aim is to have as few cycles in the OS as possible. Mothy Roscoe (ETH Zurich) disagreed, saying that many applications intentionally spend a long time in the OS, and this illustrates what hardware designers don't understand about operating systems. Mothy's real desire is hardware that does less stuff in hardware and just provides fast mechanisms that software can use. Jeff Mogul agreed with Mothy, telling Mark that, for example, fast cache coherence in the hardware is all very well, but sometimes there is a better policy for a given workload, and it would be desirable if we could implement that in software without having to trick the hardware into doing our bidding. Steve Hand (Cambridge) reminisced about the glory days of software/hardware co-design and mused that Intel should buy Microsoft or vice versa, to take us back to those days. Steve also praised FPGAs, which have become relatively easy to program, thereby allowing more people to try their hand at hardware design. Joe Tucek (HP Labs) mourned the loss of software-controlled TLBs. Margo Seltzer suggested that we need to pitch to industry, rather than other researchers, and asked what the virtualization researchers did to get hardware support in modern instruction sets. Matt Welsh—tongue firmly in cheek—suggested that the answer was to build something that is useful but really slow without hardware support.

Aleks Budzynowski (UNSW/NICTA) turned the discussion to repair-from-the-chair, asking whether anything had been done to cut down the amount of state that must be sent to the repairer. Michael Walfish (UT-Austin) replied that techniques based on selectively faulting-in state to the repairer would work. Mike Swift (Wisconsin) was more attached to the idea of remote desktop solutions, but Michael replied that protecting against a malicious repairer was the real aim of the project, and where that was implemented really didn't matter. Mike Freedman (Princeton) suggested that the Geek Squad could provide a piece of software for the customer to install, which could be configured to allow access to different settings, but Michael was concerned about the cognitive overhead of configuration on non-technical users. Dave Andersen reckoned that the problem could be solved by Microsoft engineering a better access policy control panel. Mothy saw it more as a problem of liability if somebody were to make a mistake, and the "right answer" would only be found by talking to financial and legal people.

Petros Maniatis (Intel) took Mothy's point about non-technical issues and brought us back to discussing architecture. One of the overriding concerns for a processor company is whether adding a feature will get the company sued or cause bad PR. Steve Hand also mentioned the issue of backwards compatibility, which is often necessary and can inhibit innovation. Brad Chen (Google) suggested that our job is to discover technical choices, and present them to the business people; he mentioned Android and ChromeOS as two very different solutions to similar problems. Dave Holland (Harvard) pointed out that lawyers are trained to look for risk and not make policy, so we shouldn't worry about that so much, although Matt Welsh replied that that is easier to say in a university. Mike Freedman ended the discussion by remarking that he had spoken to a number of law professors who are in favor of technical solutions, because things are much slower to change in the legal and policy fields.

Panel: Cloud Computing

Panelists: Mendel Rosenblum, Stanford; Rebecca Isaacs, Microsoft Research; John Wilkes, Google; Ion Stoica, UC Berkeley.

Summarized by Lon Ingram (lawnsea@cs.utexas.edu)

John introduced the panel session by saying that Matt Welsh had asked them to fight, but they found that they agreed too much on the fundamental academic questions in cloud computing to do so. The panel chose to instead discuss two subjects that they did disagree on: (1) what should academics do that is not useless and (2) what should industry do that is not worthless.

The panelists offered brief introductory remarks, and John completed the introductions with a bid for the Most Controversial Opinion prize by declaring that he never wanted to read another paper submission that talks about improving Hadoop performance by 10%. Discussion then began in earnest, with the panel taking questions from the audience.

The conversation covered a broad range of topics, but a recurring theme was multiple pleas from academics for industry to release large anonymized datasets that would be useful for understanding what workloads datacenters see at scale. The industry representatives responded that this was unlikely to happen and that anonymizing such datasets is far harder than one would expect. Rebecca proposed a possible solution: academics should run their own commercial cloud platform as a way to generate such datasets themselves.

The panel and the audience also discussed the difficulties academics face evaluating proposed solutions without access to the kind of scale that industry sees. Mike Freedman of Princeton asked for examples of algorithms that looked good in the small but failed at scale. John replied that it typically isn't O(n) that is the problem but, rather, the complications introduced by interactions with other components and operational concerns—upgrading a system while it's in operation, for example. Ion added that academics need to understand how to evaluate solutions without running them at scale.

Matt Welsh from Google launched the final discussion of the session by asking how to get industry to open up more and how industry can help train the next generation. John suggested that those working in industry should find an academic and tell them about a problem they have—talking to them until they understand the problem.

We're Going to Need More Wine

Summarized by Srinath Setty (Srinath@cs.utexas.edu)

Macho: Programming with Man Pages

Anthony Cozzie, Murph Finnicum, and Samuel T. King, University of Illinois

Anthony Cozzie started the talk by pointing out a hard truth about programming: programming is hard and programmers make errors when they write code. Then he described the architecture of Macho, a system that can automatically generate Java programs. Macho takes the description of the functionality in a natural language as input and then uses a database of code snippets to stitch together a piece of code with the functionality specified in the natural language. Macho also includes an automated debugger to test the generated code using a set of examples.

Margo Seltzer from Harvard asked about the progress made in the project. Cozzie acknowledged that the problem is hard and the module involving the database is the hard problem. Brad Chen from Google suggested that it would be very useful if Macho generated a specification along with the implementation. Cozzie agreed. Joe Tucek (HP Labs) asked about the amount of time taken for generating code. Cozzie replied that the ls example takes about 20 minutes.

Pursue Robust Indefinite Scalability

David H. Ackley and Daniel C. Cannon, The University of New Mexico

In the second of the two Best Talks, David Ackley pointed out the conflict between efficiency and robustness in computer systems. He went on to propose a computational model, Movable Feast Machine, to achieve indefinite scalability. However, this approach sacrifices the following three properties in the current system's architectures: first, fixed-width addresses and unique node names; second, logarithmic global communication cost; and third, clock and phase synchronization.

In the proposed design, the Movable Feast Machine consists of a 2D grid in which each tile contains a processor with a fixed amount of volatile and non-volatile memory. Each processor can communicate with its nearest neighbor processors via point-to-point links. The computation model for the proposed machine consists of a set of "event windows" that involve a group of tiles communicating with each other to perform the computation. Note that many non-overlapping event windows can exist concurrently. One of the critiques for this proposed architecture is that the hardware costs will be too high for cost-effective computation.

Mike Dahlin (University of Texas at Austin) asked about the rationale behind choosing small atom sizes. Ackley answered that the smaller sizes provide fine-gained mobility, which is essential for indefinite scalability. Dave Anderson (Carnegie Mellon University) asked about the advantages of Movable Feast Machine's local propagation restriction. David replied that local propagation enables expressiveness in the proposed architecture. Erez Zadok (Stony Brook) asked whether he had looked at any newer computing models to see whether anything matched. David replied that his PhD work was in neural networks a thousand years ago, and this is his attempt to start again from scratch. Michael Walfish (UT-Austin) asked about the types of computations that can be represented in the proposed computation model. Dave said any computation under the stochastic flow-sorting category can be represented on Movable Feast Machines. Toby Murray (NICTA/UNSW) asked if there is a way to quantify the error in output generated for the computations run on the proposed architecture. David acknowledged that quantifying error is a hard problem and one could reduce error by replication and repetition.

Hear Ye, Hear Ye

Summarized by Suparna Bhattacharya (suparna@csa.iisc.ernet.in)

*Benchmarking File System Benchmarking: It *IS* Rocket Science*

Vasily Tarasov, Saumitra Bhanage, and Erez Zadok, Stony Brook University; Margo Seltzer, Harvard University

Vasily Tarasov began his talk by citing a recent study which found that research conclusions in medicine often contain misleading findings with a heavy focus on exciting results to the exclusion of other aspects, and noted that similar observations could be made about the state of filesystem benchmarking. He argued for an improved evaluation approach which adequately reflects the complex multi-dimensional character of file-system behavior. As a follow-up to their previous ACM TOS (Transactions on Storage) paper, "A Nine-Year Study of File System and Storage Benchmarking," he told how he and his co-authors surveyed 100 file system papers from 2009 and 2010 and found a wide range of benchmarks used, with little standardization, e.g., as many as 74 ad hoc benchmarks and 24 custom traces. Even among the standard benchmarks used, many were based on compilation or small file operations, effectively stressing CPU or memory more than on-disk layout.

As a possible way forward, Vasily proposed creating standardized benchmarks for common filesystem dimensions such as on-disk layout, prefetching, and in-cache performance. In addition he emphasized the need for reporting results in terms of curves and distributions across a range of parameters instead of a single number, since filesystem behavior can be sensitive to even small changes in the environment. To illustrate how widely conclusions from benchmarking may be impacted by the choice of evaluation approach, he presented an interesting case study comparing the graphs of random read throughput (using filebench) of a 410 MB file across three file systems (ext2, ext3, and XFS) as measured at 10 second intervals. Initially, the performance is I/O-bound and eventually, when the file is completely in the page cache, it becomes CPU/memory-bound. At both these extremes, performance is similar for all three file systems, but in the transition range, which involves a 10-fold jump in throughput between the interval from 200 to 800 seconds, the differences between file systems can vary widely (up to as much as an order of magnitude) depending on the time when measurements are made. This can result in radically different conclusions from point comparisons. Likewise, a 3D plot of latency histograms collected periodically for ext2 random reads reveals a bimodal kind of characteristic, with a 1000-fold difference between the modes. Average results make very little sense in such situations.

Someone raised the concern that it might be very tough to ensure that the dimensions are orthogonal to each other. Vasily responded that indeed isolating dimensions is important but sometimes hard; however, even without orthogonality, we can still ensure coverage. Phil Levis (Stanford) felt that the comparison with medicine might be misleading since, unlike medicine, file systems do not involve human subjects.

John Ousterhout observed that the real question is not just one of capturing data but a need to understand and explain what is actually going on, e.g., the reason for different modalities in the graph. In the ensuing discussion, Jeff Mogul argued that the purpose of benchmarking is comparison, not understanding. With this approach it isn't clear how one would compare these multi-dimensional result distributions. Margo Seltzer responded that there is no one uni-dimensional comparison that works, because the weighting may not be same for all uses. While this means more work for the reader, it ensures that results are less biased. Erez Zadok observed that the networking community uses CDFs more than the storage community. Jeff Mogul asked whether they explain why one CDF is better than another. Margo Seltzer reiterated that there is no single preferred answer; it depends on what we are trying to achieve. Vasily observed that often such benchmarking really comes down to benchmarketing. Someone remarked that having a marketing target can help, especially in pushing improvements over time, and asked whether we should redefine benchmarks as a composition of performance curves and a purpose-specific utility function.

David Holland remarked that as a consumer of benchmarks we still don't know what the good choices are. The state of file-system (FS) benchmarking in the OS community is abysmal; we need an official set of FS benchmarks. Erez Zadok responded that among benchmarking tools, they found filebench to be nice and hence forked and fixed it. Now it supports two dozen random distributions, can handle multimodal distributions, and uses a data generator instead of merely writing zeros as some other benchmarks do.

Multicore OS Benchmarks: We Can Do Better

Ihor Kuz, ETH Zurich, NICTA, and the University of New South Wales; Zachary Anderson, Pravin Shinde, and Timothy Roscoe, ETH Zurich

Ihor Kuz observed that there is a fundamental problem with existing multicore OS benchmarks—they measure scalability of applications but do not evaluate how well the OS manages performance isolation between different applications. In this respect, they fail to expose performance effects of what is arguably the central purpose of an OS: that of allocating and sharing resources across applications. To address this concern, he proposed a systematic benchmarking approach that employs a mix of application workloads running concurrently. The mix is carefully chosen in a way that (1) exercises multiple system resources without overcommitting any resource and (2) is performance-sensitive to the availability of resources.

An application-specific goodness function is used to perform a sensitivity analysis of the performance of each candidate application variant (choice of application parameters) with respect to various machine resources: e.g., CPU, cache, memory, disk, and network. For example, a Web browser is partly sensitive to network bandwidth, with the rendering of Web pages being CPU-sensitive. On the other hand, the goodness metric of a virus scanner might be the number of files scanned, which is disk-bound. The design of the optimal (maximally sensitive) mix is posed as an integer linear programming optimization problem, based on resource usage and sensitivity, subject to the constraint of avoiding resource overcommit. Intuitively, the optimal solution is a mix of application variants that use resources that they are most sensitive to. Once the results from running an optimal mix on an OS have been obtained, several evaluations may be performed. For example, the performance difference in running an application unmixed and mixed can highlight potential problems in the system. Different operating systems might have a different optimal mix; comparing performance at these points can indicate how well each OS manages its optimal mix.

Ihor concluded with some comments on the status of the work. Currently they have tried this with Linux micro-benchmarks; they need to run it with real applications. The approach assumes a constant resource usage, hence will need to be extended to account for bursty applications. Further, it uses a static mix, while in desktop scenarios, application mixes are dynamic.

Michael Dahlin asked whether the optimal mix gains in stressing the OS while sacrificing completeness, making it difficult to compare results (unlike typical OLTP benchmarks). What if the optimal mix is not even realistic? Ihor responded that one can play around with parameters and constraints of the ILP formulation to restrict solutions to realistic or sensible combinations rather than irrelevant mixes. Livio Soares suggested including OS abstractions of resources in addition to raw resources and Ihor agreed. Someone raised a concern about the difficulty of stating and proving that various resources can be scheduled together without overcommit in tricky situations. Ihor accepted that one may run into this issue for real applications but observed that OSes need to handle such situations, so we should be able to test for this.

It's Time for Low Latency

Stephen M. Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K. Ousterhout, Stanford University

Steve Rumble made a case for rearchitecting systems for low latency communication in datacenters, anticipating realizability of up to two orders-of-magnitude improvement in RPC round-trip times and the significant impact this can have on enabling future Web applications. He began by highlighting the increasing demand for low latency as foreseen by constraints faced today by applications like Facebook, which randomly access many pieces of non-local interdependent data in fast DRAM-based storage for each small request. Commodity network bandwidth has increased by a factor of 3000 in the past 30 years, while latency has only decreased by a factor of 30; high latency limits Facebook to 100–150 dependent data accesses per page request. Working around such constraints not only adds to application complexity but also renders certain features non-viable.

Steve then presented a component-wise breakup of the high 300–500us RPC latency in current datacenters. He observed that because of the small distances between servers within a datacenter, the limiting factor is not the propagation delay (< 2us) but the delays across multiple hop switches (10 hops with 10–30us/hop) and a comparable delay in the NIC (10–128us) and OS stack (60us). He then argued that recent hardware improvements have brought us to the cusp of low latency. The time is right for the OS community to initiate a rethinking of the stack and architecture to reduce the rest of the overhead. 100ns latency switches and 1us latency NICs are already available in the HPC space, with Fulcrum Microsystems and Mellanox pushing the boundary to sub-500ns switches in the commodity Ethernet space. Steve predicted that this means that 5–10us round-trip times are within reach in the short term by addressing OS/protocol overheads while defining a simpler API structure that has a different distribution of responsibility between the OS, application, and the NIC than Infiniband/RDMA or U-Net. Since a datacenter is a closed ecosystem, it is even possible to experiment with new protocols that can scale low latency to 100K+ nodes instead of living with TCP. Steve projected that even lower latencies are possible in the long term; below 10us, transferring data between the NIC and the system would become a bottleneck, but a round-trip latency of 1us is achievable in 5–10 years by re-architecting systems to transmit/receive data directly from the CPU cache.

The talk generated a lot of questions both during the Q&A and the discussion session that followed. Matt Welsh remarked that we've been through similar work in the past which failed, but not due to technical reasons—the NIC was the bottleneck back then too. Could those ideas (from active messaging/U-Net) be applied now or is there something fundamentally different? The response was that today we have massive datacenter applications that need this, and low latency is becoming practical in commodity space. Matt followed up by noting that we knew how to get good performance under ideal conditions but the programming model at that time was awful—the sheer amount of engineering needed to get stable performance over time was a challenge. It was mentioned that many SIGCOMM papers had appeared on the chained RPC and scatter-gather problem, but no one cared before. Further, while we can make it faster, maybe XML/SOAP is not the most efficient way to dispatch requests—we will run up against the propagation wall sometime. Mike Schroeder noted that commodity support does not matter all that much, since there is a need within a datacenter. He mentioned that they were seeing problems with packet switching and might need circuit switching instead. John Ousterhout remarked that the community was too influenced by the success of MapReduce, which is bandwidth -oriented; there are other applications, such as realtime analysis of graphs with no locality, that really need low latency.

There was a question about why the DRAM isn't directly put on the NIC, since the CPU is not really used; Steve responded that it is essentially the same, only the CPU is programmable. Joseph Tucek remarked that infiniband costs only $300/port which is not that expensive. Michael Swift wondered if there was a case for saving data persistently at low latency, especially with NVRAM, but no suggestions came up. Michael Dahlin observed that the fact that there are about 150 dependent data-access steps for a Facebook request was intriguing, and asked whether it was the ratio of latency to overhead that mattered and if benchmarks could be designed to capture this. Perhaps the cool stuff did not matter because it got hidden by other overheads. Prabal Dutta asked why the netFPGA project was not considered a fabric to explore these questions. Steve responded that he didn't think that switches are an issue and that future problems in NICs will only appear after we solve the other problems to get to 10us latency. Timothy Roscoe commented that the problem is not the design of NICs (modern NICs are pretty good), but in interfacing with the application after it gets the data, especially as NIC latencies are getting close to DRAM latency.

Discussion/Open Mike

No report is available for this session.

Data Still Matters

Summarized by Vasily Tarasov (tarasov@vasily.name)

Disk-Locality in Datacenter Computing Considered Irrelevant

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, University of California, Berkeley

Ganesh Ananthanarayanan talked about the changes in the notion of disk locality in data-intensive computing. Disk locality is exploited at all levels of the storage stack: applications, file systems, disks. The fundamental reason why corresponding optimization methods work is that disk bandwidth is significantly larger than network bandwidth. However, this statement is less true nowadays. Off-rack, rack-local, and local disks all perform almost the same. Designers still need to care about RAM locality, however. But datasets are huge in data-intensive applications (e.g., 200 times larger than available RAM size in Facebook). Can anything be done about that? It turns out that for 96% of the jobs, all the required data can fit in the RAM. It is just that current caching policies (such us LRU) cannot predict well which data to put in the RAM. Ganesh concluded that software needs to be more intelligent in deciding which data to put to the cache and which to evict.

Gregory Ganger of CMU said that this is a great example of a collective action problem. If everybody ignored caching, this would place tremendous demand on the network. Ganesh replied that in any case the network bandwidth is so high that disk locality becomes less important. Someone asked what were the 96% of jobs doing? Maybe they were CPU-bound, and Facebook just runs applications incorrectly? Ganesh agreed that it would be great to have this information but they do not have it. Someone commented that it is very easy to get a one-rack node with 12 locally attached disks. The author responded that according to his calculations one needs at least 50 disks per node.

Optimizing Data Partitioning for Data-Parallel Computing

Qifa Ke, Vijayan Prabhakaran, Yinglian Xie, and Yuan Yu, Microsoft Research Silicon Valley; Jingyue Wu and Junfeng Yang, Columbia University

Qifa Ke discussed intelligent data partitioning for performing distributed computations. When one needs to run some computation across several nodes, the job needs to be divided between these nodes. As part of this process, the data needs to be partitioned. What is the optimal number of partitions and the partition function? The simplest (and quite common) method is to partition data using a hash function, but data skew can happen in this case. In addition, computation skew can occur; even if data is of the same size, computation time is not the same for all partitions. Moreover, balanced workload does not always mean optimal performance, and the authors worked on determining the best partitioning scheme given the data and application. Data is not structured in this case, unlike with databases, which makes the problem more difficult. The authors were looking for a compact data representation. Code is user-defined as well, with different languages and execution modes. The authors proposed a three-stage approach: model the partition scheme, estimate performance, and find the scheme that provides optimal performance.

Darren Martin of Cambridge asked whether the authors plan to do partitioning online or offline. Qifa replied, Both. Another person asked what happens if the optimal solution is 50 partitions but one has only 49 nodes. Qifa said that at the moment they consider only an ideal case. Somebody suggested they use AI techniques for the partitioning problem.

Disks Are Like Snowflakes: No Two Are Alike

Elie Krevat, Carnegie Mellon University; Joseph Tucek, HP Labs; Gregory R. Ganger, Carnegie Mellon University

A lot of today's systems and techniques rely on the idea that two identically labeled pieces of hardware will perform identically, but this is not true anymore. Elie Krevat presented a study showing that modern disk drives, even if their makes and models match, perform differently. In this study, the authors looked at three generations of disk drives: 2002, 2006, and 2008 vintages. The throughput of disk drives produced in 2002 was the same. In 2006 the variance in performance reached 10%, and it increased to 20% in 2008.

The reason for this behavior is a new and "almost undocumented" feature of disk drives: adaptive zoning. Zone Bit Recording (ZBR) has been around for many years. This technology allows vendors to put more sectors on the outer tracks of a platter. However, before now, zones' boundaries were fixed by the specification of the disk. Now, on the other hand, disk manufacturers test every individual read-write head with respect to the data rate that it can sustain. Then they assign zone boundaries according to this information. Interestingly, this happens even within the disk: different platters have different zoning.

Margo Seltzer said that her research group has been aware of similar problems with other components for a long time. But nobody cared. Why should they care now? Elie responded that there are systems that can neglect this, but others should care. Michael Schroeder of MSR pointed out that pundits say that the rise of SSD drives will make this a moot point. Elie responded that SSDs are still quite expensive and that even flash drives can have variations in their performance. Philip Levis questioned if the cache will amortize this problem. Elie agreed that this can happen as long as you can prefetch data in time. But in the paper, the authors used different, streaming workloads.

Watts Up, Joules?

Summarized by Vasily Tarasov (tarasov@vasily.name)

The Case for Power-Agile Computing

Geoffrey Challen, MIT, SUNY Buffalo; Mark Hempstead, Drexel University

This presentation was unlike any other talk at the workshop—it was a whole show! I'll try to describe it, but you really had to be there to truly appreciate it. First, the title slide appeared but the presenter seemed to be missing. After a period of growing uncertainty in the audience, the second author, Mark Hempstead, stood up and said that he would have to give the talk, but that he had not seen the slides before. He pressed the space button on the laptop and what the audience saw on the screen was a genie lamp. Over the laughs of the crowd, Mark humbly confessed that he was not aware of the purpose of this slide. Maybe we need to rub the lamp in order for a genie to appear, he said. He tried it… and Geoffrey Challen, first author, ran into the room in a golden hat and a vest over his naked torso shouting "Shazam! Shazam! Shazam!" The audience roared.

The rest of the talk was a conversation between the genie (Geoffrey) and the genius (Mark) during which they designed extremely power-efficient systems that can scale to anything from a cell phone to a production server. The idea is based on the availability of more and more components with differing computational power and levels of energy consumption. Additionally, these components have become cheaper. So why not put several components of varying power on the same device and switch between them as necessary? This can provide very smooth scaling.

Mike Schroeder (MSR) asked if this could be applied in datacenters. Mark said that is definitely possible and there are some projects that already try to do this. Peter Bailis (Harvard) asked about the programming model in such environments. Geoffrey said that there are ways to design convenient programming models for such systems. He gave an example of fat binaries that support several platforms. Prabal Dutta (U. Michigan) asked about the components that already include some method of scaling—for example, CPU frequency scaling. The authors replied that their design can reuse such features. One can switch to a more powerful CPU only if all levels in the currently working CPU are used up.

Mobile Apps: It's Time to Move Up to CondOS

David Chu, Aman Kansal, and Jie Liu, Microsoft Research Redmond; Feng Zhao, Microsoft Research Asia

David Chu noted two tendencies in mobile devices: (1) they are highly programmable and (2) more and more sensors are installed on these devices. As a result, programs that use sensors are becoming very widespread. Currently, they access sensors through inflexible custom interfaces. The approach the authors suggest is CondOS, an operating system that provides a unified interface for all sensors and applications. The OS will convert data to CDUs (Context Data Units) that are returned to applications. The benefit is that applications can perform a wider variety of tasks: for example, preload calendars when a user comes into the office or auto-unlock passwords when a user is at home.

Justin Pulaski (MIT) observed that the pervasive computing community has tried to do this for a long time already, but they are struggling to come up with a proper programming model. David replied that at the moment their interface is just a single syscall to get CDUs. Another person wondered why not employ user-level solutions such as the Linux D-BUS? David said that this should not be necessary with a kernel solution, but there should be some unified interface for all programs.

Free Lunch: Exploiting Renewable Energy for Computing

Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper, Computer Laboratory, University of Cambridge

Sherif Akoush proposed moving computation and data processes to the places where green energy is generated. Governments may push industry toward being greener. Some companies have already installed solar panels near their datacenters. However, the amount of solar and wind power changes over time in a specific geographic region. If one can find two regions so that at least in one of them at any moment of time there is enough sun or wind to generate the required amount of energy, then one can migrate data and computation processes between corresponding datacenters dynamically. Migration can happen in the form of VM migration. The challenges one will have to address are storage synchronization, predictive VM migration, scheduling, and planning.

The authors did a case study in which they picked two datacenters, one in Africa and one in Australia. The downtime was only 0.5 seconds per migration, totaling 415 seconds per year, which corresponds to a very solid SLA. The cost of migration is 57.5 kJ/migration, which is also very low.

Aman Kansal (MSR) pointed out that a lot of hardware will be idling in this case. Sherif replied that energy will very soon be more expensive than hardware. A lot of people in the audience did not believe that, saying that power should become really expensive in order for this technique to start to make sense. A related question was whether the authors neglected the cost evaluation. Sherif said that at the moment they do not have a good cost model. Another concern was high latencies. The author agreed that for some applications this approach will not work. Mike Freedman asked why they were using VM migrations. Sherif answered that for some application types, where the working set is small, VMs make sense.

Discussion/Open Mike

Summarized by Rik Farrow (rik@usenix.org)

The half hour of open discussion at first stayed focused on energy saving, the topic of the previous session. Dan Wallach wondered where else we could apply the genie. Geoff said that they had focused on improving the energy footprint of a single machine, but you could consider clusters and clouds. Mark Hempstead pointed out that the cost of transitioning processes or VMs between systems or DCs needed to be taken into account. Jeff Mogul mentioned that energy and computing is where computer security was ten years ago. Security is hard to get right, and accounting is the Achilles' heel of these things. The energy cost to produce a laptop is the same as the cost of using it two years. Mark responded that he hoped we would read his paper carefully, as they were careful. Geoff actually agreed with Jeff, in that we have been using voltage scaling for ten years, and Windows still does this so poorly it is better just to turn off the laptop. Mark mentioned that if we are really going to consider scalable computing, we need to consider the entire lifecycle. Mike Schecter said that people building datacenters are watching out for their own interests, but even they do not have control of all costs, including lifecycle costs.

John Ousterhout displayed a slide from RAMCloud (a Stanford project that replaces large disk storage with DRAM in server clusters). John pointed out that while disks have 16,667 times more capacity, latency has improved much less (twice as fast), while transfer rate is 50 times better than it was in the mid-'80s. But because capacity has far outstripped latency and bandwidth, reading an entire disk, using small blocks at random addresses, has become 8333 times worse. Just reading an entire disk sequentially can take 30 hours. Peter Honeyman said that the same thing is happening with memory, but John replied that memory is still much faster. Mike Swift noted that it is faster to read from a remote cache, outside the network, than to read from the local disk. The speed of doing computation over distance is the fundamental issue. Someone from Google pointed out that DCs are expected to last for 15–20 years, and using optimistically chosen costs are just going to get you laughed at.

Gernot Heiser mentioned that with DVFS it is very difficult to get even 10% power savings, as operating voltages have dropped to close to 1 volt. Gernot also pointed out that the Thumb instruction set in the ARM chip does not save energy. It is a subset of the regular ARM ISA, but you need to execute more instructions to get the same work done. You just get a smaller memory footprint.

Nobody Likes Surprises

Summarized by Suparna Bhattacharya (suparna@csa.iisc.ernet.in)

Debug Determinism: The Sweet Spot for Replay-Based Debugging

Cristian Zamfir, EPFL, Switzerland; Gautam Altekar, University of California, Berkeley; George Candea, EPFL, Switzerland; Ion Stoica, University of California, Berkeley

Replay-based debugging is a useful technique for tracking down hard to reproduce non-deterministic bugs which may otherwise take days or months to diagnose. The high runtime overhead involved in ensuring deterministic record-replay, however, is a major barrier to making these tools practical for production use.

Cristian Zamfir argued for a new model of determinism, called "debug determinism," which specifies that a system should at a minimum reproduce the failure and the root cause of the failure in order to be useful for debugging. Thus, debug determinism maximizes debugging utility, yet it is a relaxed-determinism model that has the potential to be achieved with low in-production overhead. He observed that existing relaxed deterministic replay approaches such as output determinism and failure determinism may end up sacrificing debugging utility in the process of reducing runtime overhead. For example, an output-deterministic system may only record the output but not the input context or data race that is the root cause of the failure. Debug determinism, on the other hand, relaxes determinism while ensuring that both the original failure and the root cause can be reproduced.

How might this be achieved? One could apply high-fidelity recording during portions of execution where root causes and failures are suspected—the key difficulty, of course, is that these are not known a priori. Hence, static analysis or domain knowledge is required to guess the location of possible root causes. In the case study presented for Hypertable, the authors relied on previous reports that control plane code tends to be responsible for most program failures, but only a small portion of the execution time. Thus, one approach is to record with high fidelity just the control plane. Cristian proposed a metric called debugging fidelity (DF) to assess different approaches with respect to their debugging utility. For example, DF is 1 when both the failure and the original root cause can be reproduced (e.g., when both are in the control plane in the Hypertable example) and it is 1/3 when there are 3 possible root causes for a reproducible failure and the system may reproduce one of the root causes that is different from the original cause.

Jeff Mogul asked whether it would be useful to implement a two-phase approach involving a run in high performance (relaxed determinism) mode followed by other runs with high fidelity with respect to the possible root cause of the failure. Cristian pointed out that replay debugging systems are typically targeted at failures that occur infrequently and are hard to reproduce; therefore a two-phase approach may not work well in these cases. Mike Schroeder wondered whether low fidelity might be better, since it is good to know all the root causes, in order to fix them all. Cristian responded that finding all root causes for a failure may take a long time and may be more difficult to scale. However, such a system would have higher "debugging effectiveness," which is a different metric from debugging fidelity.

One participant asked for a clarification on how one can know up-front where the root causes are. Cristian replied that one could over-approximate where root causes are likely to be based on a heuristics or static analysis, then record those parts of the execution with high fidelity. For instance, one might be able to statically over-approximate where all the data races are. Some of these data races may be benign; due to the over-approximation, they would be recorded as well, yet the system would achieve debug determinism for failures caused by data race bugs.

Non-deterministic Parallelism Considered Useful

Derek G. Murray and Steven Hand, University of Cambridge Computer Laboratory

In contrast with much recent work that treats non-determinism as a source of undesirable problems in parallel programming, Derek Murray made a case for extending distributed execution engines to enable explicit support for non-deterministic execution in applications that can benefit from it. He began his presentation by explaining how distributed execution engines (e.g., MapReduce) take care of a lot of parallel programming drudgery, including parallelization, synchronization, scheduling, load balancing, communication, and fault tolerance. It is the last of these that requires deterministic execution. Thus non-determinism comes at the cost of trading off transparent fault tolerance. However, he argued that more efficient and versatile programs can be built if non-determinism is supported as a first-class abstraction in distributed computing engines, and that this could be achieved without forcing additional complexity on computations that do not involve non-determinism. He presented examples like branch-and-bound and applications with irregular-sized parallel sub-trees; these can be speeded up significantly by reducing wasted work, using primitives like asynchronous signals for work shedding and non-deterministic select to continue execution without synchronization delays.

Since the main challenge with implementing non-determinism lies in dealing with faults, Derek discussed a possible range of policies, from a conservative but expensive record and replay to explicit error/exception handling by applications, to the other extreme of a fail-everything all-or-nothing approach. One of the more interesting alternatives proposed was that of bounded non-deterministic annotations for computations that have deterministic outputs, but which may be implemented internally using non-deterministic steps for efficiency. The paper also describes how tainting could be used to differentiate non-deterministically generated outputs from deterministic references to restrict the impact.

Mike Schroeder asked about the extent to which the authors have managed to act on these observations. Derek responded that they need to understand the distribution of failures before concluding what the appropriate solution should look like. Cristian Zamfir wondered how one would deal with undesirable non-determinism such as a bug in the JVM or the kernel. Derek clarified that they were not trying to deal with those kinds of problems, but were focused on explicit user-level non-determinism. The main message here is that currently the problem of handling non-determinism has been pushed to lower layers of the system; instead, we should pull back some of it to higher layers where it may be cheaper to handle and enable more flexibility.

Finding Concurrency Errors in Sequential Code—OS-level, In-vivo Model Checking of Process Races

Oren Laadan, Chia-Che Tsai, Nicolas Viennot, Chris Blinn, Peter Senyao Du, Junfeng Yang, and Jason Nieh, Columbia University

This intriguing title marked the last talk of the session on non-determinism. Oren Laadan highlighted an important problem that has received very little attention in the systems community compared to the active research on thread races. This is the existence of process races, or races which occur when multiple processes access shared OS resources without proper synchronization, e.g., non-determinism in the results of ps aux | grep XYZ or a shutdown script unmounting a file system before another process writes its data. Using results from their survey of sampled race reports for common Linux distributions, he pointed out that process races are numerous and growing over the years. They can also be dangerous, resulting in data loss and security vulnerabilities. Diagnosing process races is challenging, however, because of: (1) the diversity in scope (involving multiple programs written in different languages with complex interactions involving a variety of heterogeneous resources); (2) the need for a race detection algorithm that can handle these complex and often underspecified interactions between system calls and resources; (3) the difficulty of ensuring coverage due to dependencies on elusive conditions such as timing, environment configuration, and usage scenarios; and (4) a high likelihood of false positives or benign races.

Oren described their solution to the problem: RacePro, a system which combines lightweight online in-kernel record/replay (to transparently track access to shared resource accesses at the OS level) with an offline exploration engine that analyzes the record (using model checking) to detect potential process races. While the first piece addresses the scope challenge, the second addresses coverage. The algorithm challenge for race detection is solved by mapping this to an equivalent memory race detection problem which treats resources like memory locations and system calls like memory read/write. The last step of the solution is an offline validation using a live replay of a modified version of the recording that forces candidate race conditions to help rule out false positives. With their preliminary implementation they have detected 14 races, including 4 that result in a data loss, 5 that result in a crash, and 5 security vulnerabilities. Of all the races detected by the exploration engine, only 3–10% proved harmful, showing that the validation step is crucial.

There were several questions about what the underlying recording scheme actually captures and the assumptions made. Oren explained that they record all system calls and the partial order of their access to resources. Responding to a question from Marcos about whether they rely on a model of the OS for knowledge of what the shared resources are, Oren mentioned that their record replay mechanism is based on Scribe, their earlier work on a transparent lightweight application execution replay, published at SIGMETRICS '10. The basic resources are decided up front—e.g., IPC, files, inodes (not every single lock in the kernel), and partial ordering for a resource are recorded/effected by tracing internal kernel function accesses. Since Scribe can replay any application, including one that is multi-process and multi-threaded, RacePro can detect process races that involve threads as well. David Holland asked how robust the mechanism is to the kernel that's not working properly. The answer was that the approach assumes a correctly working kernel.

The Tin Foil Hat Session

Summarized by Srinath Setty (Srinath@cs.utexas.edu)

Privacy Revelations for Web and Mobile Apps

D. Wetherall and D. Choffnes, University of Washington; B. Greenstein, Intel Labs; S. Han and P. Hornyack, University of Washington; J. Jung, Intel Labs; S. Schechter, Microsoft Research; X. Wang, University of Washington

Right now, the research community's work can be divided into the following two categories: first, creating clever attacks to expose privacy risks, and second, devising narrow mechanisms to prevent a class of privacy risks. David Wetherall argued that the research community needs to go beyond these two classes of work and devise operating system mechanisms for privacy revelations. Privacy revelations will track how a user's information spreads in applications and will present that information to its users. The authors argue that this information will enable users to improve privacy if it can be presented as application-level concepts.

Yinglian Xie (MSR) pointed out that the problem is more than transparency: users need to know how their data gets used outside. Wetherall agreed. Matt Welsh from Google asked about the incentives for OS developers and application developers to support privacy revelations. Wetherall said that the work is not to disallow apps from tracking/collecting users' information but to expose that fact to the users. John Wilkes (Google) suggested that privacy revelations should go beyond what the authors defined: the operating systems should point out information about the ways in which the tracked information gets used. Wetherall agreed.

Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection

Petros Maniatis, Intel Labs Berkeley; Devdatta Akhawe, University of California, Berkeley; Kevin Fall, Intel Labs Berkeley; Elaine Shi, University of California, Berkeley and PARC; Dawn Song, University of California, Berkeley

Petros Maniatis began with a story about health data. His foot was injured in Palo Alto, and then he was hit by an ambulance, re-injuring the same foot, while in the UK. It would have been useful to have data from the medical work done in California while in the UK. But it is important to maintain control over our own health data.

Maniatis presented the secure data capsules vision: the owner of data sets a policy; policy is enforced during its lifetime, and data provenance is maintained throughout.

Then Maniatis presented the challenges involved in realizing this vision. First, the vision requires tracking information, which is known to be expensive in practice; devising efficient mechanisms to track the flow of information is a challenge. Second, the vision requires us to devise composable and meaningful policy definitions. Third, covert channels are a serious threat and need to be addressed.

Toby Murray (NICTA/UNSW) pointed out that microkernels are good for secure data capsules. Then Toby asked if naming is going to be the hard problem. Maniatis agreed but pointed out that it needs to be solved in order to realize the proposed vision. An audience member pointed out that Palladium at MSR, with a similar vision, had problems with displaying output on commodity hardware, and asked if this is going to be a problem in this work. Maniatis said there are solutions to the secure display problem if there is hardware manufacturer support. Michael Swift (University of Wisconsin) asked if it is going to be a problem to create policies to handle medical data before the data gets used. Maniatis pointed out that there are a couple of ways to handle this: the system could have policy violation budgeted to address the unknown data usage information, or a quorum of entities could decide the policy dynamically at runtime.

Making Programs Forget: Enforcing Lifetime for Sensitive Data

Jayanthkumar Kannan, Google Inc.; Gautam Altekar, University of California, Berkeley; Petros Maniatis and Byung-Gon Chun, Intel Labs Berkeley

Gautam Altekar explained that their idea is to create OS mechanisms to ensure that sensitive data is not retrievable after a defined data lifetime date has expired. This mechanism should not require support from applications. Altekar presented their initial work, state reincarnation, in which an operating system rolls back the application's state, replaces sensitive information with equivalent non-sensitive information, and rolls forward the application. State reincarnation eliminates any sensitive data from the system after its lifetime, but the challenge in achieving this is to derive equivalent non-sensitive data during the process. Altekar pointed out that output deterministic replay (SOSP '09) can be used to solve this problem in many cases, but overheads are going to be high. The talk also presented overheads by recording information at user-level: for bash, the slowdown was 1.2 times.

An audience member asked whether the goal could be achieved by simply going back in time. Altekar pointed out that that proposed fix would disrupt the application and the user. John Ousterhout from Stanford asked if this would increase risk by recording information. Altekar answered that users have to trust the recording system to not leak information. Timothy Roscoe from ETH Zurich asked if this is going to be used, since users may not like to record all their actions. The answer was to reduce the costs to make it favorable for the users to use it.

Discussion/Open Mike

No report is available for this session.

MacGyver Would Be Proud

Summarized by Sherif Akoush (sa497@cam.ac.uk)

Exploiting MISD Performance Opportunities in Multi-core Systems

Patrick G. Bridges, Donour Sizemore, and Scott Levy, University of New Mexico

Patrick Bridges presented opportunities to increase the speed of fixed-size workloads proportional to the processor count. He argued for strong scaling in systems software, and he gave the example of a single TCP connection as why we need it. For small MTUs, TCP synchronization across multiple cores is a bottleneck and it kills performance. Multiple-instruction/multiple-data (MIMD) approaches do not solve this problem, as they require coordinating activities between cores.

The alternative approach is to use a multiple-instruction/single-data (MISD) execution model based on the replication of sequential code across cores. In other words, synchronization is being replaced by replicating sequential work to guarantee consistency. They have implemented the system, and the initial result is that their model scales well for TCP receive processing. Patrick concluded by giving other examples, such as high-throughput file systems, in which the MISD approach would be beneficial.

Timothy Roscoe from ETH Zurich asked whether TCP is the interesting case in this approach and if it has any sequential component that can be replicated across cores to achieve scaling. Patrick answered that TCP state information such as window size, congestion control, and flow control is basically the sequential code that needs to be consistent across cores. He believes that TCP would be the killer application, to speed up a single connection flow proportional to the number of cores. Mike Swift asked about other applications that would fit this model. Patrick said that moving a large chunk of data in the DB world would be interesting as well.

More Intervention Now!

Moises Goldszmidt and Rebecca Isaacs, Microsoft Research

Moises Goldszmidt argues for what-if scenarios for data-parallel systems (e.g., MapReduce and Dryad). Unfortunately, this cannot be done by passive observations only; active interventions are required to learn the causality of different parts of the system. The proposed approach makes use of well-developed mathematical models, theories, and engineering.

The approach relies on passive observations to build the confounding factors that require further active interventions. Then, a Bayesian network is developed and executed which determines the experimentations needed. Statistics and machine-learning techniques provide a set of new rules that can be used for active interventions.

Matt Welsh said that he was the reviewer that said you are reinventing control theory, something that was done for 50 years. Can you apply stuff that was done 30 years ago? Moises answered that they actually combined different models to come up with a new formulation that can be used to decide on the active interventions. Rodrigo Fonseca (Brown) asked how the blueprint (i.e., causality) is captured, and if it is captured wrong, how this might affect the conclusions. Moises answered that the blueprint is constructed from passive observations (e.g., data sizes from nodes in the cluster that are running the tasks), while active interventions are what actually correct any wrong assumptions in the blueprint.

make world

Christopher Smowton and Steven Hand, University of Cambridge Computer Laboratory

Christopher Smowton argued that programs such as Firefox, OpenOffice, and Eclipse are rubbish since they repeat work that is done in either the current or the previous session. Developers of these programs never consider specialization of the software, because it is a challenging task. A spell-checker, for example, converts a local/global dictionary into a machine-readable form every time before checking. An easy optimization is to do this conversion only once per session. Alternatively, the program can be manually rewritten to save its intermediate results.

The paper proposes a more efficient technique based on global optimization: automated specialization of programs by partial evaluation. In the specific example of the spell-checker, this technique treats the dictionary as a constant propagated through the rest of the program. A prototype has been implemented as a proof of concept (not for the spell-checker) which can eliminate the redundant work of counting words in a file. However, there are challenges in making this adaptive optimization efficient and between multiple processes.

Phil Levis (Stanford) asked how this adaptive optimization would work for specialized paths in the current user's home directory. Christopher answered that this can be mitigated by either pushing this challenge to the user or writing a wrapper script that can check whether there is a specialization version available. Petros Maniatis (Intel Labs Berkeley) asked how this approach compares to speculative execution. Christopher answered that what he is proposing is complementary, as it can eliminate some decisions that are not needed ahead of the speculative execution. An audience member commented that the proposed approach can be augmented to model deferential execution of multiple program invocations over time to identify common state which is useful. Dave Holland (Harvard) wondered about what happens if the content of the file changes during execution. The reply was that, for the time being, it will be left to the developer to take the correct action.

Poster Session

Summarized by Lon Ingram (lawnsea@cs.utexas.edu)

The Best of Both Worlds with On-Demand Virtualization

Thawan Kooburat and Michael Swift, University of Wisconsin—Madison

Kooburat and Swift propose on-demand virtualization, where users run natively most of the time to reap the full performance of their hardware, but can switch to running in a virtual machine when needed. They save the state of the OS and running processes through hibernation, use hotplugging to transition devices from physical to virtual hardware, and employ logical devices to preserve device bindings, which allows network connections to be maintained.

Mobile Apps: It's Time to Move Up to CondOS

David Chu, Aman Kansal, and Jie Liu, Microsoft Research Redmond; Feng Zhao, Microsoft Research Asia

Chu presented his team's vision for a new kind of mobile OS service. The OS would provide applications with context signals—whether the user is standing or sitting, for example, or in a loud or quiet environment—in addition to raw sensor data. They claim that such an OS would better protect the user's privacy and use resources more efficiently, among other advantages.

Seeking Efficient Data-Intensive Computing

Elie Krevat and Tomer Shiran, Carnegie Mellon University; Eric A. Anderson, Joseph Tucek, and Jay J. Wylie, HP Labs; Gregory R. Ganger, Carnegie Mellon University

Krevat and his team investigated what inefficiencies affect data-intensive scientific computing (DISC), which they define as large-scale computations over big datasets. They used a simple model of DISC and a library called Parallel DataSeries to look for performance problems.

Detern: Robustly and Efficiently Determinizing Threads

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

Yang and his team built on recent work in the field of deterministic multithreading, making it more robust and efficient. Their system caches thread schedules and reuses them; it also uses a hybrid schedule that takes advantage of the fact that there are typically relatively few races during the execution of the program.

Execution Synthesis: A Technique for Automated Software Debugging

Cristian Zamfir and George Candea, Ecole Fédérale de Lausanne

Zamfir and Candea created a method for automatically finding a path through a program that reproduces a reported bug. Their technique uses a focused path search based on a combination of heuristics and symbolic execution to reach a target failure state. It incurs no runtime overhead and can find deadlocks and race conditions.

Memento: In-Memory Caching for Datacenters

Ganesh Ananthanarayanan, Ali Ghodsi, and Andrew Wang, University of California, Berkeley; Dhruba Borthakur, Facebook; Srikanth Kandula, Microsoft Research; Scott Shenker and Ion Stoica, University of California, Berkeley

The Memento team are working on a memory cache for data-intensive workloads in datacenters. They observed that most jobs are small and require all of their data to be cached to reap performance benefits. Large jobs, on the other hand, experience linear improvement as their working set is cached. Traditional caching disciplines ignore the all-or-nothing constraint on small jobs. Memento categorizes jobs by size and tries to ensure that this constraint is met so that large jobs don't starve small ones.

Pervasive Detection of Process Races in Deployed Systems

Oren Laadan, Chia-Che Tsai, Nicolas Viennot, Chris Blinn, Peter Senyao Du, and Junfeng Yang, Columbia University

This project uses a recording of process interactions through the system call interface to detect process races. Once a race is detected, the system re-executes the processes until immediately before the racing syscalls. It then resumes execution, but with the loser of the race executing before the winner. This technique allows them to reduce false positives by ignoring benign races.

Sirikata: Design and Implementation of a Next Generation Metaverse

Philip Levis, Stanford University; Michael J. Freedman, Princeton University; Ewen Cheslack-Postava, Daniel Reiter Horn, Behram F.T. Mistree, and Tahir Azim, Stanford University; Jeff Terrace, Princeton University; Bhupesh Chandra, Stanford University; Xiaozhou Li, Princeton University

Sirikata is a project to actually build a usable virtual world. The challenges presented by such an undertaking are unique and daunting. The project is well into its implementation, with 12 undergraduate students building a virtual city called Merustadt over the summer of 2011.

SPECTRE: Speculation to Hide Communication Latency

J.P. Martin, C. Rossbach, and M. Isard, Microsoft Research SVC

Programs that share mutable state and sequential algorithms are harder to run efficiently on multiple machines. SPECTRE uses prefetching and speculative execution to run sequential algorithms in parallel, rolling back if a conflict is encountered. It is currently running as a prototype on a small cluster.

T2M: Converting I/O Traces to Workload Models

Vasily Tarasov, Santhosh Kumar Koundinya, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College

T2M is an effort to create benchmarks from I/O traces. Traces are broken into chunks based on what model will be used to analyze the chunks. The chunks are then modeled and a workload model is output, which can be used as a benchmark in the future.

Why a Vector OS Is a Bad Idea

Vijay Vasudevan and David Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Awarded Best Poster!

Vasudevan presented the winning poster, which discussed the downsides to the Vector OS project that he discussed in the final session of the workshop. Two problems identified on the poster were the difficulty of programming to an explicit vector interface, illustrated with code showing the extra work required, and latency penalties paid when code diverges. The poster also included a space for audience members to fill in their own objections to the scheme.

Wild and Crazy Ideas Session

Summarized by Thawan Kooburat (kooburat@cs.wisc.edu)

Matt Welsh (Google) presented MEME OS, which is designed to appeal to the Internet generation. This generation does not understand the messages displayed via the text-based terminal. He proposed the use of funny images from the Web as a way to report output or error messages to users.

Margo Seltzer (Harvard) conducted a poll on how people carry their mobile phones. She found that those who carry their phones in their pockets are mostly men. This means that women may not carry the phone with them when leaving their desk to do small errands. Research on mobile phones should also take women's behaviors into account, since they represent the other half of the demographic.

Jeffrey Mogul (HP Labs) proposed a journal for reproduced results in OS research. He also came up with several ideas to provide incentive for people to work on this journal. For example, submitting a paper to this journal should be required to get tenure. The authors of any system paper need to put down $1000 on paper submission, which will go to the reviewers if the result is refuted within two years. Many people responded that other research communities—the database community, for example—already have mechanisms such as reproducibility committees to verify published works.

Dan Wallach (Rice University) complained about the current submission process, in which papers get into the loop of submit-reject-revise. He proposed that all papers should get accepted immediately as tech reports. This sparked a debate where people discussed the submission process of other conferences such as SIGMOD and VLDB. Others also raised the idea of removing anonymous review or using crowdsourcing instead of peer review.

Mike Walfish (University of Texas at Austin) showed a YouTube video where a penguin is taught to go shopping. He suggested that robots should be used to perform mundane tasks like this. He presented his work in building robots which are easy to program and able to perform simple tasks such as getting a cup of coffee. He proposed a model where people can download pre-programmed tasks to their robots from places like AppStore.

David Anderson (CMU) suggested that systems research is about dealing with constraints imposed by hardware. Previously, we have been able to use many abstractions to hide some hardware details such as uniform memory and sequential computation. However, as we are hitting the physical limit of physical devices, we will start to throw away these abstractions. He believed that we will ultimately reach the end of scaling of physical devices and we should accept this fact. Because of this, we should think carefully about which abstractions to throw away and in which order, so that programmers will continue to survive despite these changes.

Joseph Tucek (HP) explained that the next-generation computer system is just a machine with a different ratio of hardware resources. He proposed that cutting-edge research can be carried out by putting together machines that simulate this ratio. For example, we can pair a 386 processor with a 10 Gbps network to mimic the future Terabit network.

Prove It!

Summarized by Sherif Akoush (sa497@cam.ac.uk)

What If You Could Actually Trust Your Kernel?

Gernot Heiser, Leonid Ryzhyk, Michael von Tessin, and Aleksander Budzynowski, NICTA and University of New South Wales

Gernot Heiser presented an seL4 microkernel that is formally proven to be functionally correct. The kernel is free from crashes, bugs, and similar safety issues. Interesting applications are for the purposes of better virtual machine monitors and isolating Web browsers. Trusted platform modules (TPM) can also be made practical by the use of a trusted kernel with a trusted verified loader.

Taking home banking as an example, TPM is practically useless, as it forces the users to boot into a special banking configuration that will kill any other concurrent access to other machine features. Late launch/DRTM is also practically useless, as it does not allow for interrupts, DMA, or multiprocessing. The proposed solution is to load the banking application in a mini OS that is also loaded with a verified loader on top of a verified seL4 kernel. The user's standard OS is still working in parallel and is not affected. Additionally, DBMS would not need synchronous log writes, as it is guaranteed with a verified kernel that the OS will not crash. In this case, there is no tradeoff between performance and reliability.

John Ousterhout (Stanford) asked if they had found any issues in seL4 since the SOSP '09 paper. Gernot replied that they had found a few proof bugs, around specification, configuration, and initialization. The only way around that is to complete the proof chain for the security parts. Brad Chen (Google) asked whether there is more than isolation that can be gained by a verified kernel and how application correctness can be guaranteed. Gernot answered that the guaranteed kernel functionality can be leveraged to ensure user-level component interfaces and this is something they are currently working on. Mike Swift asked what happens if the memory fails, and Gernot replied that people trust their RAID systems today. He also said that the military would like to have triply redundant memory for some applications.

Provable Security: How Feasible Is It?

Gerwin Klein, Toby Murray, Peter Gammie, Thomas Sewell, and Simon Winwood, NICTA and University of New South Wales

Toby Murray argued that provable security for a real system is feasible but certainly not easy. Real proofs are done by machines and can provide you with unexpected insights into high-level security issues such as integrity and confidentiality. Real systems are big and often written in C or assembler, not in a language that is designed to be proofed.

Toby provided seL4 as an example of a proofed kernel that enforces integrity. It is a machine-checked proof with 10,000 lines of proof-script code. However, timing channels are still too hard to be proofed and require a very detailed model of the underlying hardware. Additionally, systems like Linux cannot be proofed easily, as they have large trusted components.

Steven Hand (University of Cambridge) asked what is required if changes are made to the kernel. Toby answered that it depends on the level of modification done to the kernel; the correctness proofs rely on a number of invariants that have been proved about the kernel, and most of the work in proving correctness involves proving these invariants. So changes that do not break the invariants or introduce new ones require little work; however, ones that do require more work.

John Ousterhout (Stanford) asked about the number of lines of code seL4 has and how the effort required for the proof scales with the number of lines. Toby replied that seL4 has 8,600 LOC and noted that according to his experience, the proof should scale more than linearly (about square) with the size. Brad Chen (Google) asked how important the missing specification for hardware is. Toby answered that it is really important and they are actually working on modeling the hardware.

Toward Practical and Unconditional Verification of Remote Computations

Srinath Setty, Andrew J. Blumberg, and Michael Walfish, The University of Texas at Austin

Srinath Setty presented a practical and unconditional verification of remote computations which is useful in cloud and volunteer computing. Basically, the client needs to verify that the server has executed the code correctly without redoing the computation. One solution is to use probabilistically checkable proofs (PCPs), but PCPs are currently just applied in theory. The purpose of this paper is to make them practically possible (i.e., position the challenge as a systems problem). They refined PCP via arithmetic circuits instead of Boolean circuits, to make the system efficient, and implemented the design to demonstrate its practicality.

The prototype achieves savings of 10 orders of magnitude by using the refinements presented. It is based on a divide and conquer strategy: dividing the problem into smaller parallel parts that the server checks simultaneously and then verifies. However, more refinements are still required to reduce the storage cost and support floating-point operations, for example.

Steven Hand (University of Cambridge) asked why we should use PCPs instead of replication, as replication is simpler to verify computations. Srinath replied that replication assumes a threshold on the number of faulty servers, but PCPs provide stronger guarantees. Mike Freedman (Princeton) asked whether the optimizations that have been made can be generalized to the implementation of other computations. The answer was that these optimizations can be applied to any computation expressed as a circuit. In principle, a compiler can be used to translate a circuit representation from a high-level specification.

MOMMIE Knows Best: Systematic Optimizations for Verifiable Distributed Algorithms

Petros Maniatis, Intel Labs Berkeley; Michael Dietz, Rice University; Charalampos Papamanthou, Brown University

Petros Maniatis argued for an approach that guarantees the development of both algorithmic logic (for verification) and optimizations (for efficient implementation). Abstractions are great, but developers usually end up modifying the actual implementation code when systems are built. The actual code is therefore too difficult to be verified for correctness. Moreover, we should not worry about this level of implementation detail.

The proposed middleware, MOMMIE, is a high-level language that can be used to compose the system. This abstraction can then be translated to a specification (TLA+) for formal verification or to an optimized program (C++) for execution. In this way, proofs carry over into implementation without having to rebuild everything from scratch if the system changes.

A MOMMIE statement is the fundamental building block and is composed of an issuer, a verifier, an auditor, and C-structs. The program looks like event-condition-action, where actions have assignment, loops, and variables (i.e., imperative code). A prototype is available but it is still in its early stages.

Mike Freedman (Princeton) asked whether the designer tells MOMMIE which parts of the algorithm should go to the formal proof and which parts are for actual optimizations. Petros answered that they focused on abstraction so that anything that is composable can be proved in isolation and some aspects are mapped manually to implementation detail. Toby Murray asked whether there are any restrictions on the algorithm formulated in MOMMIE. Petros replied that there are some restrictions: for example, it cannot allow for arbitrary loops, because the goal is to reduce the amount of work a designer has to do. Others wondered how this work differs from other systems and protocols. Petros replied that MOMMIE provides a granularity (middleware) that is not found in any other system.

OS Design Isn't Dead; It's Just the Last Session of the Workshop

Summarized by Thawan Kooburat (kooburat@cs.wisc.edu)

The Case for VOS: The Vector Operating System

Vijay Vasudevan and David G. Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Vijay raised the fact that in a Web server each request often executes a similar sequence of operations, such as accepting a network connection, opening a file, etc. Thus, a lot of redundant and identical work is performed when servicing requests in parallel using multiple cores. He proposed a vector system call as a mechanism to increase system efficiency. Vectorization allows batching of system calls, which reduces kernel crossing overhead, but, more importantly, it allows redundant work to be eliminated. Examples include reducing pathname resolutions, using SSE instructions in hash calculations, and looking up data structures more efficiently.

Vijay demonstrated the benefits of a vector system call while performing memory protection at the speed of millions of operations per second. This was achieved by vectorizing the mprotect system call. First, vectorization batches up several system calls into one, similar to FlexSC. Then it eliminates redundant TLB flushes and algorithmically exploits vector abstractions by sorting requests based on page address, which reduces memory allocation overhead. These optimizations provided a factor-of-three improvement in the mprotect rate: 30% of the improvement was attributed to avoiding system call overhead, while the other 70% came from the vector opportunities deeper in the stack, emphasizing the need for vectorization at all levels. Next, Vijay described the challenge of dealing with divergent execution paths. He proposed a solution based on either forking extra threads and rejoining them when execution paths converge, or using lightweight message passing between function calls. However, deciding when to fork and join execution is application-specific and remains a challenge. Finally, he described one way of building a vector OS by restructuring the OS as a staged event system. This allows the programmer to write sequential code and let the system handle vectorization.

Erez Zadok (Stony Brook) asked about the difficulty of dealing with errors when using vector system calls. Vijay replied that this task can be simplified by using an event-based model, as it allows programmers to handle errors individually. Andrew Baumann (Microsoft) pointed out that other OS designs explicitly avoided synchronization overhead by executing work redundantly in parallel. Vijay responded that eliminating redundancy at the cost of serialization improves efficiency, especially for I/O-bound operations in a highly parallel Web server. Mike Schroeder (Microsoft) brought up concerns about latency, since the system may have to wait to batch up requests. Vijay said that existing techniques such as interrupt coalescing already batch up requests at the network layer before they arrive at the application, providing an opportunity for vector execution to improve efficiency without adding significantly more latency.

Operating Systems Must Support GPU Abstractions

Christopher J. Rossbach and Jon Currey, Microsoft Research; Emmett Witchel, The University of Texas at Austin

Christopher raised the fact that the GPU, as a general-purpose computing device, is underutilized because it is treated as an I/O device. He strengthened his argument by showing that the CPU has a much richer set of abstractions, such as process and pipe, than the GPU, which has only ioctl as the main interface. He described several issues caused by the lack of a proper abstraction. First, there is no fairness or isolation guarantee from the kernel. Second, the absence of a kernel-facing interface means the kernel cannot use GPU directly. Third, he presented two experiments to highlight CPU/GPU performance isolation and scheduling problems.

Fourth, Christopher talked about his gestural interface program. He tried to decompose the program into a collection of programs connecting via pipes. However, this design has poor performance as a result of the unnecessary data movement. With existing GPU abstractions, this overhead cannot be removed.

Christopher emphasized that general-purpose GPUs need more abstraction, similar to what the CPU has. It needs many APIs to support functions such as scheduling and inter-process communication. The right abstraction should enable program composition and eliminate unnecessary data movement between CPU and GPU. The proposed abstraction is based on a dataflow programming model. First, PTasks represent a computation executing on a GPU. They have priority to allow the kernel to enforce fairness. They are also connected via ports and channels. These specialized channels allow programmers to eliminate unnecessary data movement when an opportunity arises. Finally, he revisited his gestural interface program to show how these abstractions solve the problem.

Erez Zadok (Stony Brook) suggested that if the GPU is incorporated into the CPU like the floating-point coprocessor was, this problem may go away. Christopher responded that having better hardware is also one of the solutions, but he would like to have a solution that works with existing hardware. Philip Levis (Stanford) argued that this problem is irrelevant, as the CPU is moving to multicore and becoming more heterogeneous. Christopher explained that dataflow is still important since it may be the right model for any type of accelerator. Brad Chen (Google) brought up concerns regarding GPU security issues and believed that it should not be tightly integrated with the OS. Christopher replied that better support from hardware, such as allowing context switching and better specification, can mitigate the problem.

Multicore OSes: Looking Forward from 1991, er, 2011

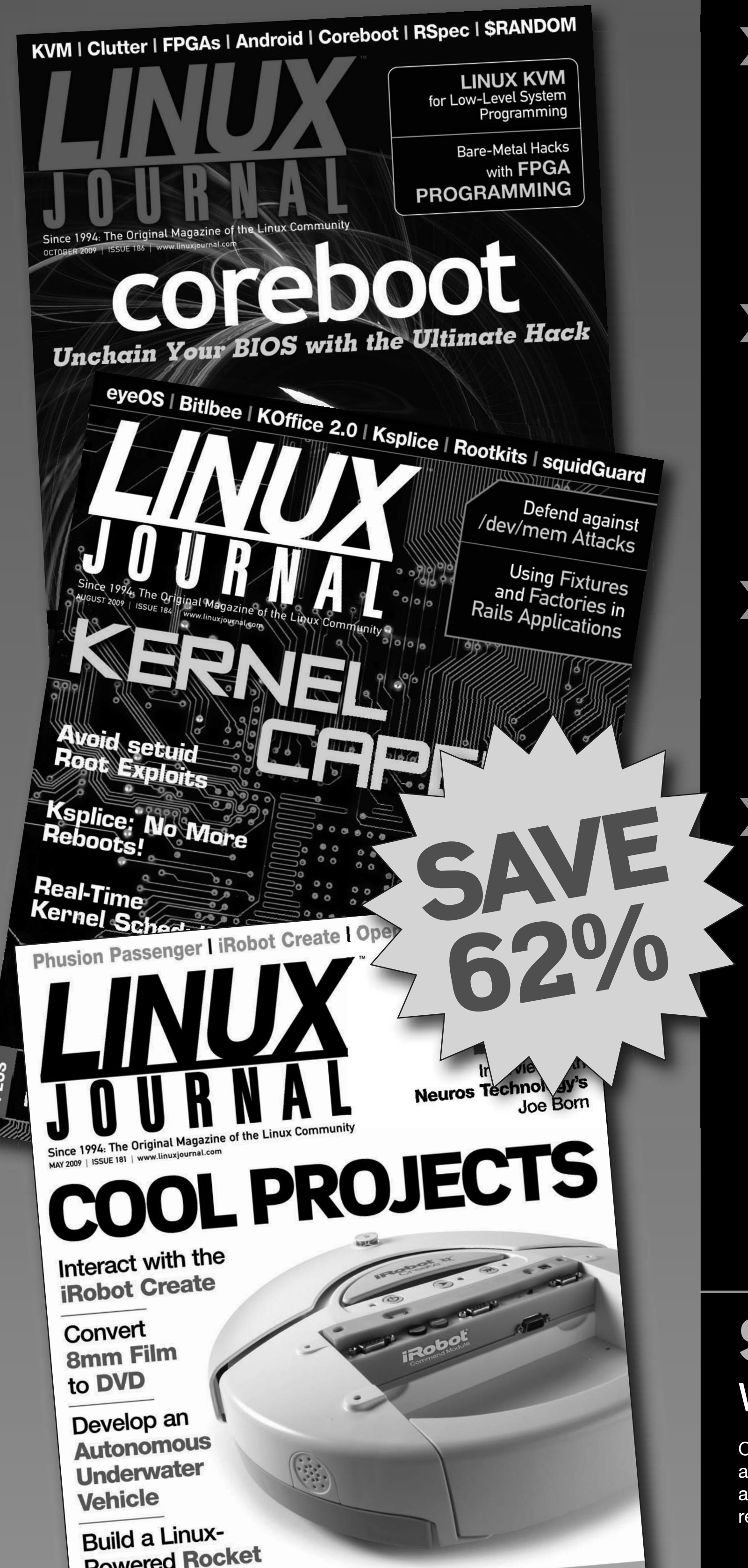
David A. Holland and Margo I. Seltzer, Harvard University

David complained about multicore systems. Hardware is not getting faster and we are forced to adopt parallel programming, which makes good scalability very difficult to achieve. However, these are the same challenges that people who worked with supercomputers faced around 1991. The experience gained from this shows that machines with thousands of cores will need to adopt the shared-nothing architecture. We also learned that message passing is the right model for programming these machines. However, instead of using MPI, a lightweight message channel is already available in languages such as Go and Erlang. Since it is based on a shared-nothing architecture, it has the potential to achieve good scalability. Unlike sending a MPI and network RPC packet, sending a lightweight message is comparable to a procedure call.

David talked about how to restructure the kernel to use messages. He presented a diagram which shows the kernel running on a separate core instead of underneath the application. In shared-nothing architecture this is possible, since there is no need to protect processes from overwriting each other. He also discussed several challenges in building such a system. For example, relying on a hardware-based channel can lead to a similar issue that people relying on Infiniband encountered. Virtual memory may also look entirely different, since there is no kernel running underneath. In addition, this model may encourage programmers to write too many threads. Finally, fault-tolerance and scheduling may become issues as well.

Timothy Roscoe (ETH Zurich) argued that some form of kernel is still required to run together with the application in order to perform privileged tasks on its behalf. Hence, only OS services are needed to be restructured around message passing and run on a different core. David responded that hardware-based channels can remove the need for running the kernel in the same core. Joseph Tucek (HP) raised the point that a NUMA machine with 1000 cores is already available today from SGI and is used by NASA. David Andersen (CMU) commented that this system tries to achieve Mach-like message passing with Go-like language support. He also mentioned that people usually wrap an RPC-like interface around message passing. David confirmed the point about language support and argued that a lightweight message can have as low overhead as a function call. On the other hand, RPC is too heavyweight to replace every function call in a program.

If You Use Linux, You Should Be Reading LINUX JOURNAL™



» In-depth information providing a full 360-degree look at featured topics relating to Linux

» Tools, tips and tricks you will use today as well as relevant information for the future

» Advice and inspiration for getting the most out of your Linux system

» Instructional how-tos will save you time and money

Get *Linux Journal* delivered to your door monthly for 1 year for only $29.50! Plus, you will receive a free gift with your subscription.

SUBSCRIBE NOW AT:

WWW.LINUXJOURNAL.COM/SUBSCRIBE

Offer valid in US only. Newsstand price per issue is $5.99 USD; Canada/Mexico annual price is $39.50 USD; International annual price is $69.50. Free gift valued at $5.99. Prepaid in US funds. First issue will arrive in 4-6 weeks. Sign up for, renew, or manage your subscription on-line, www.linuxjournal.com/subscribe.

VMware Global Education and Research

Academic Programs from VMware

VMware® has created rich global education and research offering with two distinct programs to allow access to the latest virtualization technologies for both research and instruction purposes:

- **VMware Academic Program (VMAP):** If you represent an academic institution, VMAP can provide access to cutting-edge virtualization and cloud technology, along with access to experts for research and teaching purposes.
  - Free and Discounted Licenses
  - Source Code Access
  - Research Funding
  - Conference Sponsorships
  - Research Collaboration
  - Guest Lectures
  - Scholar in Residence Program
  - VMware Fellowship

- **VMware IT Academy Program:** VMware IT Academy membership can help you get your VMware Certified Professional certification with access to VMware eLearning and instructor-led course materials. More details are available at http://www.vmware.com/partners/programs/vap/

Researchers can request source code and software tools to assist with research. More details are available at **http://www.vmware.com/partners/academic/program-overview.html**

"We view research and education on virtualization and cloud computing as critical to computing and IT, now and especially in the future. The VMware Academic Program provides us (and others) with invaluable resources and connections, helping us push the frontiers in these areas."

Dr. Greg Ganger
Professor in the Electrical and Computer Engineering & School of Computer Science
Carnegie Mellon University

VMware Labs is our home for collaboration. Here, users can check out the latest innovations coming out of VMware and share feedback and ideas directly with our engineers. At labs.vmware.com, you can find 'Flings' (cool new tools and utilities), publications, videos, and more! Learn more and visit http://labs.vmware.com/ to read more about what VMware education and research offers.

VMware Inc.
3401 Hillview Avenue
Palo Alto, CA 94304
vmap-ask@vmware.com

usenix

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

LISA '11

25TH LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE

December 4–9, 2011, Boston, MA

SPONSORED BY
usenix
IN COOPERATION WITH LOPSA and SNIA

Come to LISA '11 for training and face time with experts in the sysadmin community.

The theme for LISA '11 is "DevOps: New Challenges, Proven Values."

LISA '11 will feature:

6 days of training on topics including:

- Virtualization
- Security
- Configuration management
- And more!

Plus a 3-day Technical Program:

- Invited Talks
- Paper Presentations
- Guru Is In Sessions
- Practice and Experience Reports
- Vendor Exhibition
- Workshops
- Posters and WiPs

Stay Connected... http://www.usenix.org/facebook http://twitter.com/LISAConference

Find out more at www.usenix.org/lisa11/lg