usenix ;login:

FEBRUARY 2012 VOL. 37, NO. 1

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix ASSOCIATION

UPCOMING EVENTS

In Cooperation: EuroSys 2012

SPONSORED BY ACM SIGOPS IN COOPERATION WITH USENIX

April 10–13, 2012, Bern, Switzerland
http://eurosys2012.unibe.ch

2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '12)

CO-LOCATED WITH NSDI '12

April 24, 2012, San Jose, CA, USA
http://www.usenix.org/hotice12

5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '12)

CO-LOCATED WITH NSDI '12

April 24, 2012, San Jose, CA, USA
http://www.usenix.org/leet12

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, USA
http://www.usenix.org/nsdi12

In Cooperation: 5th Annual International Systems and Storage Conference (SYSTOR 2012)

IN COOPERATION WITH ACM SIGOPS (PENDING) AND USENIX

June 4–6, 2012, Haifa, Israel
http://www.research.ibm.com/haifa/conferences/systor2012

4th USENIX Workshop on Hot Topics in Parallelism (HotPar '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, ACM SIGARCH, AND ACM SIGPLAN

June 7–8, 2012, Berkeley, CA, USA
http://www.usenix.org/hotpar12

2012 USENIX Federated Conferences Week

June 12–15, 2012, Boston, MA, USA

2012 USENIX Annual Technical Conference (USENIX ATC '12)

June 13–15, 2012
http://www.usenix.org/atc12

3rd USENIX Conference on Web Application Development (WebApps '12)

June 13–14, 2012
http://www.usenix.org/webapps12

4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '12)

June 12–13, 2012
http://www.usenix.org/hotcloud12
Submissions due: March 8, 2012

4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '12)

June 13–14, 2012
http://www.usenix.org/hotstorage12
Submissions due: March 12, 2012

4th USENIX Workshop on the Theory and Practice of Provenance (TaPP '12)

June 14–15, 2012
http://www.usenix.org/tapp12
Submissions due: March 31, 2012

6th Workshop on Networked Systems for Developing Regions (NSDR '12)

June 15, 2012

21st USENIX Security Symposium (USENIX Security '12)

August 8–10, 2012, Bellevue, WA, USA
http://www.usenix.org/sec12

15th Workshop on Cyber Security Experimentation and Test (CSET '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6, 2012, Bellevue, WA, USA
http://www.usenix.org/cset12
Submissions due: April 19, 2012

3rd USENIX Workshop on Health Security and Privacy (HealthSec '12)

CO-LOCATED WITH USENIX SECURITY '12

August 6–7, 2012, Bellevue, WA, USA
http://www.usenix.org/healthsec12
Submissions due: April 10, 2012

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
Arnold Gatilao
Jane-Ellen Long
Casey Henderson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

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

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

usenix ;login:

FEBRUARY 2012, VOL. 37, NO. 1

Musings

RIK FARROW

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

I just finished reading about how humans managed to survive, just barely, a robot revolution [1]. A quick and easy read, it was just what I needed after immersing myself in file systems and the cloud. And fortunately, after a grim beginning, the book focuses more on the puny humans' successes than on their failures.

We already live in a world where robots are becoming more common. Some examples, such as factory robots, are obvious. But others, such as Google's self-driving Prius [2], don't look at all like robots. But then, neither do factory robots look at all like Robbie the Robot, nor do the robots that help to care for old folks that are gaining acceptance in Japan.

Thinking about robots, not the virus-infected bloodthirsty type, got me to thinking about automation and autonomous systems. For several years, autonomic systems were the rage in some parts of the sysadmin world, while being anathema in others. Many sysadmins felt that bringing autonomy into their networks would make the sysadmin redundant. That won't happen, any more than the fifth-generation programming languages [3] managed to do away with programmers back in the 1980s.

Automation

If we go back to the 1950s, we have early examples of computerized automation that go well beyond card-punch readers. ADP, for example, was selling Software as a Service (SaaS) by 1957, with their payroll-processing business. ADP quickly expanded to offer other services, all SaaS.

To stick to something more in the world of system management, I began managing multiple systems in 1985 and used UUCP [4] to share data between these systems. Each system was installed independently, configured independently, and updated independently. UUCP was a serial line protocol, and I had wired the office where I consulted with lots of telephone wire terminating with RS-232 connectors. Any change, such as adding a user or a new host, involved manually editing files on each existing host.

After Ethernet arrived, at a blistering 10 megabits/second, we had real networking at last, but still no automatic administration. We could use Sun's Network Information Service, but only on the Suns.

But as TCP/IP networking evolved, we began to see what might not at first look like automation, but certainly worked that way. DNS is a great example, as it replaced fetching and installing the /etc/hosts file with a system where updates appeared

almost immediately. Having MX records really helped deal a death blow to UUCP's bangpath addressing, where you needed to know every host along the path if you wanted to send email to someone.

DHCP is another wonderful example of network automation. Instead of having to configure each system with an IP address, DHCP does that for you, and more. Note that IPv6 does support automatic address assignment, but fails to provide the other key features that DHCP currently serves (such as host and domain name, and the addresses of key network services).

In 1994, Sun Microsystems introduced Jumpstart: instead of installing each system, you used network boot and network installation of computers. We now have similar systems, such as Kickstart, for installing Linux systems.

None of this automation put sysadmins out of work, as far as I know.

The Cloud

While ADP got an early start as a "cloud" provider, almost 50 years before the buzzword became popular, the cloud is a real game changer today. I attended the High Performance Transaction Systems (HPTS) workshop in October (see the reports in this issue) and heard many fascinating talks about how the ability to host your applications with various cloud providers is transformative. Yes, this too has to do with automation.

Instead of gangs of sysadmins installing system images, we now have Web interfaces that allow us to point-and-click our way through the process. There are more elaborate versions of this, such as the offerings of VMware, which allow you to build and control your own clouds. There are also custom interfaces, built on top of cloud providers' interfaces, that make installing your system images, complete with your own applications and data, as easy as filling in Web forms.

Adrian Cockcroft, during HPTS, told the fascinating story of how Netflix has migrated away from datacenters using SQL databases into the cloud [5]. Adrian described several reasons for the migration: reliability, flexibility, and cost. That a cloud-hosted service could be more reliable than a traditional datacenter service surprised me. But if you read more about the Netflix story, you too will begin to understand how this works.

The NoSQL (not-only SQL [6]) databases also support a movement to both clouds, and to farms of low-cost servers. NoSQL-style databases are good when you need to scale beyond what can be done with a single server, no matter how many processors, gigabytes of RAM, and SANS you throw at the problem. HPTS also had a debate contrasting scaling up (the SQL database model) with scaling out (the farm of servers running one of many varieties of NoSQL).

Robots and Jobs

Today, people roam the hot and noisy aisles between racks in datacenters, replacing hard disks, power supplies, and entire systems. Datacenters are noisy environments, and not the nicest places to work. As James Hamilton (Amazon) said during his HPTS talk, if your datacenters look nice enough that you invite people to see them, you have paid too much for your datacenter [7]. It seems to be that datacenter system wranglers might someday be replaced with robots.

Robots would also make much safer drivers, instead of the multi-taskers we have today who reserve most of their attention for the person on the far end of a cell phone connection.

The Lineup

I was also able to attend the SNIA Storage Developers Conference [8] this September, where I learned some amazing things that I might have missed had I not gone. Even though SDC is long over, I did ask several people to write about some of the topics covered that I felt would be relevant to USENIX members.

First up, Josef Bacik (Red Hat) explains Btrfs. I had first heard about Btrfs (pronounced "butter FS") four years ago, during a Linux File System workshop running alongside FAST. At the time, I thought that Btrfs would be the Linux clone of Sun's ZFS, but, if anything, Btrfs is a new file system that resembles Sun's, but with simpler administration and the features of ZFS. When I learned that the "Btr" in Btrfs referred to B-trees (as used in Mac file systems), I asked Josef to include an explanation of B-trees as well. Knowing that Btrfs uses B-trees for almost everything makes developing Btrfs easier and administering it easy, and it helps you decide which of your file-system needs best match Btrfs instead of ext4 or XFS.

I didn't meet Rob Chansler at SDC, but he too has a filesystem-related article. Rob has worked on HDFS development at Yahoo!, and wanted to share his experience with very large HDFS installations. HDFS is a distributed file system designed to work with Hadoop, and Yahoo! is not just the initial developer of Hadoop but one of its larger users as well.

I encountered Chris Hertel in the hallway at SDC, shortly before he was scheduled to speak. I had met Chris many years before in Minneapolis, where I was teaching Linux security. While catching up with Chris, I learned that he is a Samba developer, had written the book on the SMB protocols, and had become familiar with SMBv2. SMB, Microsoft's file sharing protocol, had pretty much stagnated for many years. SMBv2 has changed that, with many times the performance, and many of the features one expects to find in modern file-sharing services. Chris not only explains the advantages of using SMBv2 but also shows Microsoft in a new perspective which I think will surprise you. It certainly did me, as did the live demonstration of SMBv2.2 beta between a bunch of disks over bonded channels with automated failover.

Alex McDonald (NetApp) finally provided something that I had long searched for: an explanation of NFSv4, along with motivation for moving to it from NFSv3. NFSv4 has been around for years now, but adoption has been glacial. Too many people have spent too many years learning how to deal with the eccentricities of NFSv3 and are not anxious to try something new. As you read Alex's article, you will realize that some of the features appearing in beta in SMBv2.2 have long been a part of NFSv4. NFSv4 makes managing NFS simpler, its security better, and its performance much faster, as well as adding support for HPC. Alex's article also contains a link to his SDC white paper on migrating to NFSv4 from v3.

I also met Stuart Kendrick at SDC, although he was not there as a presenter. Stuart also learned about the improved performance of SMBv2, and this appears in his article. But the focus of his article is not SMB, but discovering where the problem might lie in client-server transactions. Stuart describes, and provides scripts for, slicing up client-server pie, a nice visible method for seeing whether it is the client,

the server, or the network, that is responsible for bad performance. Note that Stuart has written for *;login:* before.

I met Adrian Cockcroft (Netflix) at HPTS. Adrian was very excited about Netflix's move from big iron in datacenters to Amazon's AWS and EBS cloud services. I talked with him during the reception, listened to his HPTS talk, read his other online slide presentations, and read his Netflix blog. And I still had more questions, which Adrian took some time out of his crazy-busy schedule to answer.

David Blank-Edelman provided me with some timely help. David discusses Perl modules for manipulating spreadsheets and CSVs (Comman Separated Values). I was very happy about this, as someone had just dumped an Excel spreadsheet in my lap, suggesting that I extract some useful data from it. Somehow I have never learned much about spreadsheets, including how to extract just two columns from many rows. David also explains, for the Perl-inspired and spreadsheet-averse, how you can write to spreadsheets as well—just what I needed.

Dave Josephsen continues with the explanation of Graphite that he started in the December 2011 issue. Graphite is one of a trio of tools for monitoring a large number of systems, and one that Dave waxes enthusiastic about because it fulfills many of the items on his long list of features desired in distributed monitoring systems. Dave explains how to integrate Graphite with Nagios and Ganglia, as well as linking Graphite up to a number of other packages useful in analyzing monitoring data. Very cool.

Dave Beazley presents his first *;login:* column, with a reprise of his earlier *;login:* article that covers Python 3. Dave has many years of experience writing and teaching Python, and has plans to include columns about data analysis, libraries, the language itself, and system programming. In this column, Dave focuses on features of Python 3 that he particularly appreciates, contrasting them with how they might have appeared in Python 2.

Robert Ferrell muses about the process of writing columns, especially when he can't find his muse.

Elizabeth Zwicky reviews a pile of books, covering skills for software architects, five big data titles, and programming Pig. Sam Stover covers two books, one about MongoDB with Python, and the second a different take on *Privacy and Big Data*, a book Elizabeth also reviewed.

We conclude this issue with reports from the HPTS workshop. We worked on producing these reports because we felt that many of the issues addressed in this workshop would be relevant to USENIX members.

While I was searching for a link to Wilson's book about the robot revolution, I learned that in a year or so Steven Spielberg will have a movie out based on the book. I am sure the movie will have lots of cool special effects, and likely much better character development than Wilson's book had (after all, Wilson got his PhD in robotics), but it really was a fun read, even if the robots in it were not just murderous, but a little like Nazis from the 1940s.

We will certainly be living in a world with many more robots, ones that will perform work that human beings generally dislike doing. But that also begs the question: once robots have replaced factory workers, stoop labor, seamstresses, and even system wranglers, how are all those people going to make a living? When the problem was *just* factory workers, with many of those jobs going to poorer nations,

the white collar workers weren't too worried. But those days may soon be over. Or, to put it another way, when was the last time you called a business and a human answered the phone? The robots are taking over.

Good thing most of them don't have guns [9].

References

[1] Daniel Wilson, *Robopocalypse* (Doubleday,. 2011).

[2] Google's self-driving Prius: http://www.greencarreports.com/news/1067485_how-googles-self-driving-car-works.

[3] Fifth-generation programming language: https://secure.wikimedia.org/wikipedia/en/wiki/Fifth-generation_programming_language.

[4] UUCP Project: http://www.uucp.org/info.html.

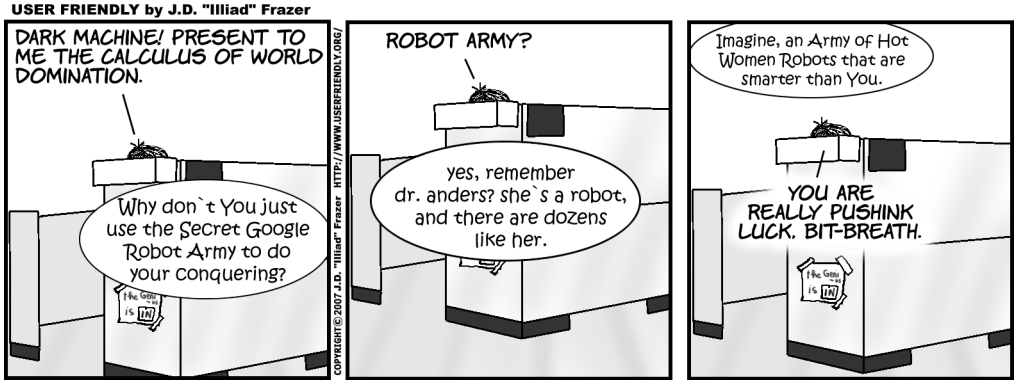
[5] Adrian Cockcroft, "Migrating Netflix from Datacenter Oracle to Global Cassandra" : http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra.

[6] Greg Burd, "NoSQL," *;login:*, vol. 36, no. 5: http://db.usenix.org/publications/login/2011-10/openpdfs/Burd.pdf.

[7] Photo tour of the Facebook datacenter: http://scobleizer.com/2011/04/16/photo-tour-of-facebooks-new-datacenter/.

[8] SNIA SDC 2011: http://www.snia.org/events/storage-developer2011.

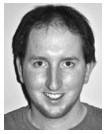
[9] Samsung SGR-A1: https://secure.wikimedia.org/wikipedia/en/wiki/Samsung_SGR-A1.



Btrfs

The Swiss Army Knife of Storage

JOSEF BACIK



Josef is the lead developer on Btrfs for Red Hat. He cut his teeth on the clustered file system GFS2, moving on to help maintain ext3 for Red Hat until Btrfs was publicly announced in 2007.
josef@redhat.com

Btrfs is a new file system for Linux that has been under development for four years now and is based on Ohad Rodeh's copy-on-write B-tree. Its aim is to bring more efficient storage management and better data integrity features to Linux. It has been designed to offer advanced features such as built-in RAID support, snapshotting, compression, and encryption. Btrfs also checksums all metadata and will checksum data with the option to turn off data checksumming. In this article I explain a bit about how Btrfs is designed and how you can use these new capabilities to your advantage.

Historically, storage management on Linux has been disjointed. You have the traditional mdadm RAID or the newer dmraid if you wish to set up software RAID. There is LVM for setting up basic storage management capable of having separate volumes from a common storage pool as well as the ability to logically group disks into one storage pool. Then on top of your storage pools you have your file system, usually an ext variant or XFS. The drawback of this approach is that it can get complicated when you want to change your setup. For example, if you want to add another disk to your logical volume in order to add more space to your home file system, you first must initialize and add that disk to your LVM volume group, then extend your logical volume, and only then extend your file system. Btrfs aims to fix this by handling all of this work for you. You simply run the command

```
# btrfs device add <device> <file system>
```

and you are done.

The same thing can be said about snapshotting. With LVM you must have free space in your volume group to create an overflow logical volume which will hold any of the changes to the source logical volume, and if this becomes full the volume becomes disabled. With Btrfs you are still limited to the free space in the file system, but you do not have to plan ahead and leave enough space in your file system in order to do snapshots. A simple df will tell you whether you have enough space to handle the changes to the source volume. Btrfs simply creates a new root and copies the source root information into the new root, allowing snapshot creation on Btrfs to take essentially the same time no matter how large the source volume.

Btrfs's B-tree

Btrfs breaks its metadata up into several B-trees. A B-tree is made up of nodes and leaves and has one or more levels. Information is stored in the tree and organized

by a key. Nodes contain the smallest key and the disk location of the node or leaf in the next level down. Leaves contain the actual data of the tree (see Figure 1).

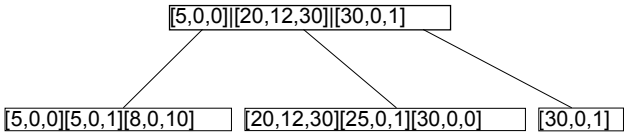


Figure 1: An example of a B-tree with four nodes and two levels

The top level, referred to as the root, acts just like a node. The entries in each node will tell you the first key in the node or leaf below it. In this example each key has three values, which is specific to Btrfs. We break the key up into objectID, the most important part of the key; the type, the second most important; and then the offset, which is the least important. So, as you can see in the above example, we have [30,0,0], which is smaller than [30,0,1]. This is important because for things such as files, we set the objectID to the inode number, and then any inode-specific information can also have the inode number as its objectID, allowing us to specify a different type and offset. Then any metadata related to the inode will be stored close to the actual inode information.

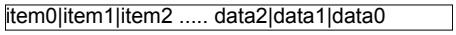


Figure 2: A leaf containing keys and data

The leaf's items contain the key and the size and offset of the data within the leaf (Figure 2). The items grow from the front of the leaf toward the end, and the data grows from the end of the leaf toward the front. Items can have an arbitrary data size, so you could potentially have one item in a leaf and have the rest of the leaf taken up with data.

This is a great advantage to Btrfs when it comes to dealing with small files. All Linux file systems address storage in arbitrary block sizes (e.g., 4 kilobytes). That has traditionally meant that if you create a file that is less than 4 kilobytes you will be wasting the leftover space. With Btrfs we can stash these smaller files directly into our B-tree leaves, so you will have the inode information and the data in the same block, which gives Btrfs a tremendous performance advantage when creating and reading small files.

The following is a basic list of the B-trees you get with a newly created file system:

- Tree root B-tree: This keeps the location of the roots of all of the various B-trees.
- Chunk B-tree: This keeps track of which chunks of the devices are allocated and to what type.
- Extent B-tree: This tree keeps track of all of the extents allocated for the system and their reference counts.
- Checksum B-tree: This tree stores the checksums of all of the data extents in the file system.
- File system B-tree: This holds the actual filesystem information, the file, and directory information.

We have all of these various B-trees to allow us a great deal of flexibility. For example, instead of having to come up with ways to stash extent reference count

information alongside file information, we simply store the two different sets of data in different trees. This makes everything easier for the developers and gives us a nice set of rules for offline error recovery. If we know how each tree should look generally, it makes it trivial to build tools to put things back together if some sort of catastrophic failure occurs.

Snapshotting

Btrfs's snapshotting is simple to use and understand. The snapshots will show up as normal directories under the snapshotted directory, and you can cd into it and walk around like in a normal directory. By default, all snapshots are writeable in Btrfs, but you can create read-only snapshots if you so choose. Read-only snapshots are great if you are just going to take a snapshot for a backup and then delete it once the backup completes. Writeable snapshots are handy because you can do things such as snapshot your file system before performing a system update; if the update breaks your system, you can reboot into the snapshot and use it like your normal file system.

When you create a new Btrfs file system, the root directory is a subvolume. Snapshots can only be taken of subvolumes, because a subvolume is the representation of the root of a completely different filesystem tree, and you can only snapshot a filesystem tree. The simplest way to think of this would be to create a subvolume for /home, so you could snapshot / and /home independently of each other. So you could run the following command to create a subvolume:

```
btrfs subvolume create /home
```

And then at some point down the road when you need to snapshot /home for a backup, you simply run

```
btrfs subvolume snapshot /home/ /home-snap
```

Once you are done with your backup, you can delete the snapshot with the command

```
btrfs subvolume delete /home-snap/
```

The hard work of unlinking the snapshot tree is done in the background, so you may notice I/O happening on a seemingly idle box; this is just Btrfs cleaning up the old snapshot. If you have a lot of snapshots or don't remember which directories you created as subvolumes, you can run the command

```
# btrfs subvolume list /mnt/btrfs-test/
ID 267 top level 5 path home
ID 268 top level 5 path snap-home
ID 270 top level 5 path home/josef
```

This doesn't differentiate between a snapshot and a normal subvolume, so you should probably name your snapshots consistently so that later on you can tell which is which.

Future Proofing

Btrfs uses 64 bits wherever possible to handle the various identifiers within the B-trees. This means that Btrfs can handle up to 2^{64} inodes, minus a couple of hundred for special items. This is a per filesystem tree limit, so you can create multiple subvolumes within the same file system and get even more inodes. Since

you can have a total of 2^{64} subvolumes, you could potentially have 2^{128} inodes in one file system, minus a negligible amount for reserved objects. This is scalability far above what could previously be achieved with a Linux file system.

The use of 64 bits also applies to how Btrfs addresses its disk space, enabling it to address up to 8 exabytes of storage. This makes Btrfs very future proof; it will be useful for many years to come as our storage capacities increase.

Directories

Directories and files look the same on disk in Btrfs, which is in keeping with the UNIX way of doing things. The ext file system variants have to pre-allocate their inode space when making the file system, so you are limited to the number of files you can create once you create the file system. With Btrfs we add a couple of items to the B-tree when you create a new file, which limits you only by the amount of metadata space you have in your file system.

If you have ever created thousands of files in a directory on an ext file system and then deleted the files, you may have noticed that doing an ls on the directory would take much longer than you'd expect given that there may only be a few files in the directory. You may have even had to run this command:

```
e2fsck -D /dev/sda1
```

to re-optimize your directories in ext. This is due to a flaw in how the directory indexes are stored in ext: they cannot be shrunk. So once you add thousands of files and the internal directory index tree grows to a large size, it will not shrink back down as you remove files. This is not the case with Btrfs. In Btrfs we store a file index next to the directory inode within the file system B-tree. The B-tree will grow and shrink as necessary, so if you create a billion files in a directory and then remove all of them, an ls will take only as long as if you had just created the directory.

Btrfs also has an index for each file that is based on the name of the file. This is handy because instead of having to search through the containing directory's file index for a match, we simply hash the name of the file and search the B-tree for this hash value. This item is stored next to the inode item of the file, so looking up the name will usually read in the same block that contains all of the important information you need. Again, this limits the amount of I/O that needs to be done to accomplish basic tasks.

Space Allocation

Like many other modern file systems, Btrfs uses delayed allocation to allow for better disk allocation. This means that Btrfs will only allocate space on the disk when the system decides it needs to get rid of dirty pages, so you end up with much larger allocations being made and much larger chunks of sequential data, which makes reading the data back faster.

Btrfs allocates space on its disks by allocating chunks, usually in 1 gigabyte chunks for data and 256 megabyte chunks for metadata. A chunk will have a specific profile associated with it: for example, it can be allocated for either data or metadata and then also have a RAID profile component. Once a chunk is allocated for either data or metadata, that space can only be used for one or the other. This allows Btrfs to have different allocation profiles for metadata and data.

For example, say you have a four-disk setup and you want to mirror your metadata but stripe your data. You can make your file system and specify RAID1 for metadata and RAID0 for data. Then whenever you write your metadata it will be mirrored across all of your disks, but when you write your data it will only be striped across the disks.

This split of metadata and data can be confusing to some users. A user may see that she has 10 gigabytes of data on her 16 gigabyte file system but only has 2 gigabytes free. In order to help clear up the confusion, Btrfs has its own df command which will show exactly how the space on the file system is used. Here is an example output from a full 7 gigabyte file system:

```
# btrfs filesystem df /mnt/btrfs-test/
Data: total=6.74GB, used=6.74GB
System: total=4.00MB, used=4.00KB
Metadata: total=264.00MB, used=121.34MB
```

This only shows allocated chunks and their usage amount. So with the above file system, if I add a disk with

```
# btrfs device add /dev/sdc /mnt/btrfs-test/
```

and then re-run `btrfs filesystem df`, I will see basically the same thing:

```
# btrfs filesystem df /mnt/btrfs-test/
Data: total=6.74GB, used=6.74GB
System: total=4.00MB, used=4.00KB
Metadata: total=264.00MB, used=121.35MB
```

This is because the new disk I added has not been allocated for either data or metadata. So I can use another command, `btrfs filesystem show`, and see the following:

```
# btrfs filesystem show /dev/sdb
Label: none uuid: 5eb80e04-26b9-4bb2-bd0f-a90a94464d6b
        Total devices 2 FS bytes used 6.86GB
        devid 1 size 7.00GB used 7.00GB path /dev/sdb
        devid 2 size 2.73TB used 0.00 path /dev/sdc
```

The size value is the size of the disk, and the used value is the size of the chunks allocated on that disk. So the new disk is 2.73 TB but hasn't had any chunks allocated from the disk, potentially allowing 2.73 TB of free space for allocation. You will see this reflected in the normal df command:

```
# df -h
Filesystem   Size   Used   Avail  Use %   Mounted on
/dev/sdb     2.8T   6.9G   2.8T   1%      /mnt/btrfs-test
```

Once you add a device, it is generally a good idea to run a balance on the file system with the command:

```
# btrfs filesystem balance /mnt/btrfs-test
```

This command, which can be run at any time, is used to redistribute space and reclaim any wasted space. If you add a disk, running balance will make sure everything is spread evenly across the disks.

Checksumming

Since Btrfs does checksum all of its data, it uses several worker threads to offload this work. When writing big chunks of data, the work will be split up among all of the processors on the system to calculate the checksums of the chunks. The same happens for reading: on completion of the read, the pages are handed off to worker threads which calculate and verify the checksums of the data so that the work is spread out, and this makes checksumming a much smaller performance hit than normal. For metadata checksumming, the checksum is calculated at write time as well, but is stored at the front of the metadata block.

Checksumming is great because it keeps the file system from crashing the box by reading bogus data, and also allows users to know that they need to be looking for a new hard drive or new memory. If you have a RAID profile that gives you multiple copies of the same data or metadata, such as RAID1 or RAID10, Btrfs will automatically try one of the other mirrors so that it can find a valid block. If it does find a valid block, everything will continue on as normal and the application will be none the wiser. The checksum mismatch will be logged so the user or administrator can be aware of the problem. If there are no other mirrors or all of the other mirrors are corrupt as well, Btrfs will return an error and the application will deal with it accordingly.

Compression

Btrfs currently supports two compression methods, zlib and lzo, with lzo being the default. You simply mount the file system with

```
mount -o compress
```

and any new writes will be compressed. Sometimes small writes will not compress well and will actually require more space compressed than uncompressed. Btrfs will notice this sort of behavior and turn off compression on the file in an effort to give the user the best possible space usage while using compression. Sometimes this is not what the user wants, however, so it can be changed by using the mount option:

```
mount -o compress-force
```

This option will force Btrfs to always compress the data no matter how it looks when compressed. Generally speaking, Btrfs does a good job balancing what should and shouldn't be compressed. The benefit of this compression infrastructure is that it is well abstracted, which makes adding support for new compression algorithms relatively easy, and hopefully it will be used to add encryption support in the future.

Solid State Drives

Solid state drives are changing how we think about storage, and Btrfs is no exception. Btrfs will automatically detect if it is on an SSD and will appropriately adjust how it allocates space. On spinning disks it is important to get good data locality, that is, to store related data as close together as possible to reduce seeking. With SSDs this is not as much of an issue, since the seek penalty is almost nothing. So instead of Btrfs wasting CPU cycles trying to get good data locality on an SSD, it will simply keep track of the last used free space for an allocation and start its

search there. Btrfs also supports TRIM, but this is turned off by default until more vendors can handle it reliably and quickly.

Filesystem Consistency

Traditional Linux file systems have used journals to ensure metadata consistency after crashes or power failures. In the case of ext this means all metadata is written twice, once to the journal and then to its final destination. In the case of XFS this usually means that a small record of what has changed is written to the journal, and eventually the changed block is written to disk. If the machine crashes or experiences a power failure, these journals have to be read on mount and re-run onto the file system to make sure nothing was lost. With Btrfs everything is copied on write. That means whenever we modify a block, we allocate a new location on disk for it, make our modification, write it to the new location, and then free the old location. You either get the change or you don't, so you don't have to log the change or replay anything the next time you mount the file system after a failure—the file system will always be consistent.

Journaled file systems can only ensure metadata consistency by writing to the journal. If your application uses fsync() to ensure data integrity, any other metadata changes that have happened recently must also be written to the journal. If you have other threads on the system modifying lots of metadata, you will have inconsistent fsync() times on a journaled file system. On Btrfs there's a special B-tree called a tree log that we use for fsync(). Any time you call fsync() on a file, Btrfs will go through and find all of the metadata that is required for that given file, copy it to the tree log, and write out the tree log. Because other threads that are modifying the metadata on the system will not affect the application doing fsync(), the application should see consistent fsync() performance. The only exception is if there are multiple applications doing fsync() on the same file system. They will all be logged to the same tree log, but this is the same as on a journaled file system.

Performance

Btrfs strives to have great performance in all workloads, but some workloads work better than others. One area where Btrfs has problems is with random overwrite workloads (i.e., writing to a file and then writing over a part of that file, and doing this often and randomly). Because of the copy-on-write design, this will lead to bad fragmentation and could result in slow cold cache reading. There is work to fix this shortcoming, and you can mount with the option autodefrag and Btrfs will notice this behavior and attempt to defragment the file in the background.

Btrfs also has quite a bit of latency associated with doing direct I/O to files. This, coupled with the copy-on-write nature of the file system, means that any enterprise database workload is likely to be slower on Btrfs than on XFS or ext4.

Btrfs tries to provide the most consistent performance possible as the file system fills up. For example, Figure 3 (next page) shows a workload where 16 threads are creating 512-kilobyte files across 512 subdirectories on a 7 gigabyte disk with ext4, Btrfs, and XFS.

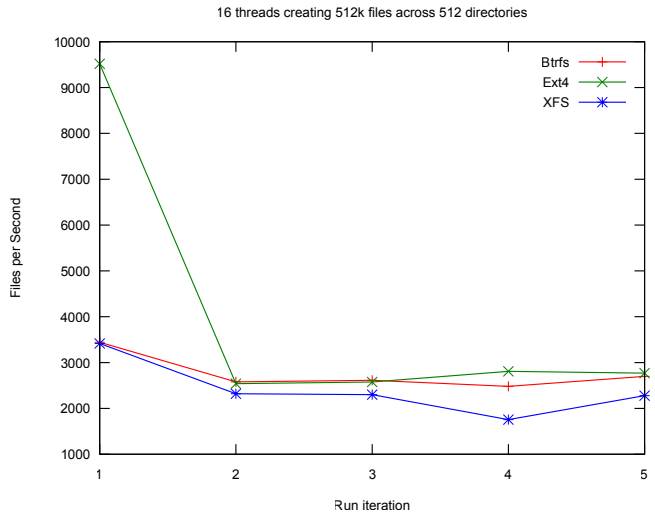


Figure 3: Btrfs, ext4, and XFS performance comparison

With small files Btrfs can really shine, since it will inline the data into the metadata. So you get a graph that looks like Figure 4.

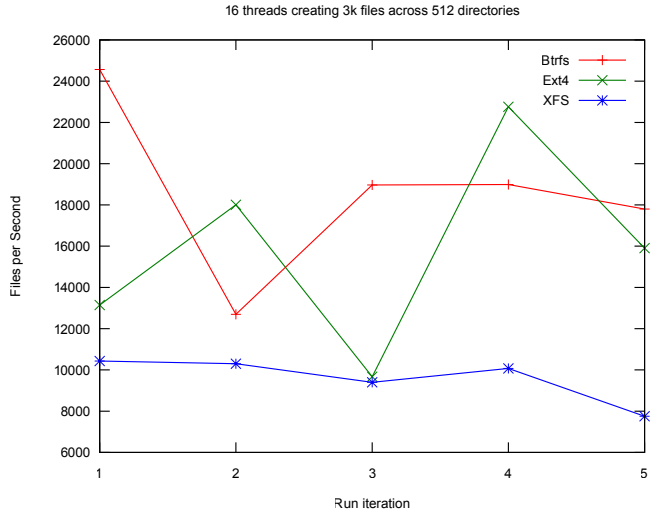


Figure 4: Btrfs, ext4, and XFS performance comparison using small files

Streaming writes onto Btrfs should be close to disk speeds. For example, writing 20 gigabytes directly to my local disk gives me a speed of 148 MB/s, and then writing to the same disk with Btrfs on top gives me 145 MB/s. Btrfs does very well at saturating the link to the disk.

Since Btrfs is still under heavy development, much of the effort is focused on finishing features and fixing stability issues. Performance is very much an important part of development, but, unlike XFS and ext4, Btrfs has not had years of widespread use to hammer out the kinks and optimize performance. In most common

workloads, Btrfs should perform much like its counterparts, but there is still a lot of work that needs to be done before it is a fair comparison.

Acknowledgments

Thanks to Chris Mason and Rik Farrow for checking the accuracy of this article and reading all of my drafts.

Data Availability and Durability with the Hadoop Distributed File System

ROBERT J. CHANSLER

Robert J. Chansler is Senior Manager of Hadoop Infrastructure at LinkedIn. This work draws on Rob's experience as manager of the HDFS development team at Yahoo!. A Caltech graduate, Rob earned a PhD in computer science at Carnegie Mellon University investigating distributed systems. After a detour through compilers, printing systems, electronic commerce, and network management, Rob returned to distributed systems, where many problems were still familiar but all the numbers had two or three more zeros.
rchansler@yahoo.com

The Hadoop Distributed File System at Yahoo! stores 40 petabytes of application data across 30,000 nodes. The most conventional strategy for data protection—just make a copy somewhere else—is not practical for such large data sets. To be a good custodian of this much data, HDFS must continuously manage the number of replicas for each block, test the integrity of blocks, balance the usage of resources as the hardware infrastructure changes, report status to administrators, and be on guard for the unexpected. Furthermore, the system administrators must ensure that thousands of hosts are operational, have network connectivity, and are executing the HDFS Name Node and Data Node applications. How well has this worked in practice? HDFS is (almost) always available and (almost) never loses data.

A Survey of HDFS Availability

The Grid Operations team tracks each cluster loss-of-service incident. Many of these incidents are "caused" by facilities other than the file system. But for some incidents, the immediate problem is that HDFS is unavailable or not performing well. Sometimes the initial incident report originates with a user frustrated that HDFS is not performing as well as expected. Automated tools are better at monitoring whether individual host machines and network switches are operational. Yahoo! uses Nagios for alerting. An internal tool called Simon is used for time series collection and presentation from Nagios; Ganglia is a popular tool used elsewhere for this purpose. HDFS itself provides several statistics streams available via JMX and collected by Simon. An introduction to using Nagios and Ganglia for cluster monitoring is available from IBM developerWorks (see Resources). The logs from cluster hosts—both the operating system logs and the HDFS application logs—can be used for fault diagnosis.

Table 1 summarizes all of the "grid down" events recorded during the 500 days preceding 5 April 2011, where loss of HDFS service is the initial or primary complaint. The table includes some data about the aggregate size of the clusters for perspective. This somewhat arbitrary interval was chosen to be long enough to collect an interesting data set, but not so long as to include much data from when HDFS was—to be frank—not so good. Roughly, this corresponds to the deployment Hadoop 0.20 versions.

The HDFS Name Node is a Java application. As such, there is always concern that garbage collection might interrupt service. Indeed, garbage collection (GC) represents 44% of the reported loss-of-service incidents. In a GC event, HDFS is

unavailable for a few (less than 10) minutes. The system resumes normal operation without intervention. The nature of the Java Virtual Machine is such that there can be no promise that GC will not interrupt service. Most GC events are close to the threshold of observability, about five minutes. During mid-2010, the biggest grids (about 4000 nodes) had short interruptions once or twice a week, although only one report was generated each month. Careful provisioning and some thoughtful software changes have reduced the frequency of interruptions, so that today big clusters have fewer than one event every other month. GC issues were nagging problems from time to time, but among HDFS developers the consensus is that HDFS has benefited from the choice to implement the system in Java. Development efficiency was higher, and inconvenient service interruptions that did occur were not the kind of faults that might threaten the integrity of the file system.

Table 2 summarizes each of the "Other HDFS Down Incidents"; HDFS was unavailable for an hour or two while the fault was repaired and the cluster restarted. HDFS ought to be excused from 16 of the events. There is not much the file system can do if LDAP, NFS, power, or a switch fails. Two of the events are attributed to software faults where code did the wrong thing. Nine events occurred when system storage resources were exhausted. Either the data nodes ran out of available space or the name system could not create additional names or blocks. User application behavior was implicated in five cases, where either the rate of requests was extraordinary or the required response to a request was exceptionally large. There is still no good protection from the former, but the latter problem is eliminated. There is no convincing explanation for the remaining four cases. The longest unexcused absence was 3.2 hours.

Clusters 19 April 2011	**GC**	**Other HDFS Down Incidents**	**Nodes**	**Blocks ($\times 10^6$)**	**New Files Per Day ($\times 10^6$)**
Research clusters	15	22	10,404	299.2	16.1
Production clusters	9	14	19,720	357.1	30.6
Totals	24	36	30,124	656.3	46.7

Table 1: Loss of HDFS service incidents for 500 days preceding 5 April 2011. Scheduled maintenance events are not included. Multiple Hadoop clusters have been aggregated into two classes: "research clusters," generally available to the engineering community, and "production clusters," restricted to approved jobs and serving more time-critical processes.

Four continuing initiatives have helped to reduce the number of incidents and the duration of incidents that do occur.

1. Operations now have better guidance on what the practical limits are and can better manage the clusters to avoid encountering the hard limits.
2. Better-provisioned servers have been tested with higher practical limits.
3. The time necessary to restart the name system has been reduced. This involved reducing the minimum necessary time to restart the name system, and better tools to validate the integrity of the system before resuming service.
4. The garbage collection parameters for the Java VM have been more carefully tuned.

Is HDFS getting better? There were seven unexcused absences in the last 100 days of this survey, but only two in the earliest 100 days. But HDFS has over twice as many nodes and clusters under management at the end of the survey as at the beginning, and the introduction of the Security feature means HDFS is more dependent on infrastructure like LDAP and Kerberos servers. Certainly garbage collection is much better, as explained previously. And if resource usage is carefully managed, the likelihood of a surprise is reduced by half.

HDFS can be satisfyingly resistant to multiple insults. In one recent incident, a journal storage volume was lost when the volume filled up under extraordinary client demand and a rack switch died. HDFS continued service and re-created the missing block replicas, although there was a user complaint that the file system was slow.

Down Incidents	Number	Details
Garbage collection	24	JVM "promotion failures" interrupted service.
Excused absence	16	Loss of power, hardware failure, misconfiguration, operation error, network faults, and the like; HDFS restarted.
Bugs	2	Software did wrong; system was restarted without software change.
Space/objects exhausted	9	Insufficient resources for users; sometimes system continued when resources were freed.
User applications	5	HDFS was unable to service demand effectively, but system recovered without intervention when excess demand was removed.
Unknown	4	No convincing explanation; HDFS restarted.

Table 2: Loss-of-service incidents categorized. The apparent cause of the incident was determined by inspection of logs and reports from the Operations team.

An initiative is underway to bring conventional "high availability" to the HDFS name system. The general idea is that a second host is provisioned and able to execute the name system application with only a modest delay (a few seconds) and without interrupting user applications. This would not remedy all of the incidents that have been described here. The loss of a critical infrastructure component would not be remediated, and some operational problems of the name system (resource exhaustion, say) would just be transferred to the other host with no benefit. Leaving aside the question of garbage collection events, 7 of the 16 excused absences would be remediated as would 1 of the 4 unattributed events. Whether to failover to the second host in case of a garbage collection event is a difficult policy question. The decision rests on whether the transfer really is a low-risk activity, and to what extent the transfer really is transparent to user applications. But if transfer in the case of garbage collection is a Good Thing, the number of incidents will not be reduced, but the duration of the interruption will be bounded by the transfer time.

The larger context for this discussion considers not just adventitious interruptions of the file system, but also the many other causes of service interruptions for a cluster. With our problem child as an example (a large cluster with an especially diverse user community), there were 13 reports of file system interruptions, but a total of 86 cluster service interruptions when scheduled maintenance activities are included along with surprises attributed to the job system and other facilities. Twelve of the maintenance windows were for deployment of Hadoop. If Hadoop—including HDFS—were engineered for continuous service during software updates, many of these interruptions could be eliminated. Also, some of the maintenance windows were scheduled to perform administrative functions (e.g., change VM size) that could have been performed with less interruption if a high availability solution had already been at hand.

A Survey of HDFS Data Durability

Data stored in HDFS may become unavailable either because it is not possible to access the bytes on disk that are the data, or else the bytes have inadvertently been changed so that the bytes accessed are not the data originally stored. Since data storage in HDFS is organized as files composed of a sequence of large blocks (256 megabytes is typical), missing data is manifest as "lost" blocks. As the typical file has one or two blocks, the loss of a block is the same as the loss of a file for present purposes.

The failure of a data server does not result in the loss of blocks, as each block has replicas at other nodes. The usual case is that there are two other replicas. But if the application program requested exactly one replica, then loss of the node hosting the single replica must necessarily result in the loss of the block. The failure of a rack switch does not result in the loss of blocks, as each block has a replica at some other rack. The core switches connecting racks are provisioned so that component failures do not isolate a slice of nodes across racks or among multiple racks. Since each node hosts about 50,000 block replicas, it's near certainty that the loss of a slice of the cluster—or multiple racks—will cause data to be unavailable. Fortunately, in the case of switch failures, the switch will be repaired and the lost block replicas will be available again.

When a node fails, new replicas are created for blocks where a replica was hosted on the failed node. All is well if the re-replication process keeps up with the occasional failure of a node. (One percent of nodes fail each month.) Of course, it might just happen that a number of nodes fail during a short interval. How likely is the loss of blocks due to *uncorrelated* failures of multiple nodes? A statistical model (Table 3, next page) suggests that it is improbable that any cluster has observed the loss of a block with three replicas due to uncorrelated failure of nodes. Nor is there any record of this having happened. There can be *correlated* failures of nodes. Experience suggests that several nodes of a cluster will not survive a power-on restart. In that case a few (~10) blocks with three replicas will be lost, consistent with the statistical model for simultaneous failures.

Probability of losing a block on a large (4000 nodes) cluster	
In the next 24 hours	5.7×10^{-7}
In the next 365 days	2.1×10^{-4}

Table 3: Probability model for predicting data loss due to uncorrelated failure of nodes. The details of the model are in the HDFS issue tracking system; a link can be found in the Resources section of this article.

The remaining opportunity for losing blocks is when HDFS just does the wrong thing. HDFS has certainly matured in the past couple of years, from when one bad circumstance might cause a block to be lost to where at least a couple of bad circumstances must occur before the loss of a block is threatened. Two aspects of the HDFS architecture are especially pertinent. First, the strategy of replicating blocks is critically dependent on the correctness of the replication monitor. Second, if a client application begins writing a file but later abandons it, the writer's lease for the file must be recovered. In the former case, the number of different circumstances that might be the proximate cause for needing a new replica is large, and so making the correct diagnosis as to which replica ought to be replaced is difficult logic. In the latter case, the intervening hour presents a long interval when various misfortunes can happen, again complicating the logic.

One type of catastrophic failure requires special mention. If the file system metadata were ever lost, all data in the entire cluster would be compromised. Briefly, the metadata is actively managed by two hosts, the data is written to multiple volumes on different hosts, the integrity is periodically tested, and snapshots are retained for long periods. There is no record of having lost any piece of system metadata in the past two years. If the name system server fails and cannot be immediately restarted, a new host can begin service using the saved metadata. The metadata is managed by a write-ahead log, and so name system server failure will not cause loss of even the most recent metadata records.

A Review of 2009

Table 4 summarizes the record of lost blocks for the clusters managed by the Grid Operations team during 2009. This data has the weakness that reporting is certainly incomplete. With that in mind, the total number of lost blocks in 2009 is likely to be less than 1.5 times the number reported here. To provide some perspective, the table indicates an estimate of the number of blocks managed by the clusters and the number of new files created each day (recall a typical file is one or two blocks).

Clusters October 2009	Blocks Reported Lost	Nodes	Blocks ($\times10^6$)	New Files Per day ($\times10^6$)
Research clusters	631	11,890	222	10.9
Production clusters	10	5830	107	5.7
Totals	641	17,720	329	16.6

Table 4: Reported loss-of-data events for 2009. In one curious case, an unexplained fault changed a host directory into a regular file. The greater number of nodes in Table 1 is due to the increased usage of Hadoop resources during 2010 and early 2011.

In one of the research clusters, for a short period (a few days), client applications were abandoning an extraordinary number of files. An error in recovering the lease for a small fraction (10^{-4}) resulted in losing the block being written. This fault accounted for 533 of the total number of lost blocks. On another cluster, a single user experimented with setting the replication factor for many files to one. He lost his gamble—and 91 blocks—when a node failed. On other clusters, two blocks were lost when a data node reported replicas as too big; four blocks were lost when nodes failed during a cluster cold start; four blocks were lost due to a faulty write pipeline recovery; an unreported (but tiny) number of blocks were lost when nodes returned to service with corrupt replicas. For 2009, maybe two dozen blocks were lost due to all other software faults.

A Review of 2010

It was not possible to repeat this analysis for 2010—well, not impossible, but considerably less convenient, because the apparent loss of blocks became dominated by a new operating procedure. There is never too much space, and when space is short, the third replica of a block looks to be an excess of caution. Blocks with two replicas are, in fact, pretty durable. Furthermore, in our practice it is not uncommon for a file in one cluster to already have a copy on another cluster. If each of those files has only two replicas at each of two clusters, space is saved while durability is enhanced. Therefore we moved to a practice whereby many files have replication set to two. Any two nodes are likely to each have a replica of some block. When many blocks have only two replicas, if any two nodes fail to start, some blocks (in proportion to the fraction of blocks with replication two) will be lost. In this context, it is important to consider two sources of *correlated* failures of data nodes.

A power-on restart of a big cluster begins by switching on each of the cluster hosts, and if the hosts successfully start the operating system, the administrator will issue commands to begin execution of the HDFS Data Node and Name Node applications. In practice, a few dozen nodes will not immediately restart. Also, consider a circumstance where a host can continue service but cannot restart the operating system or Data Node application. This might occur if the system disk volume has failed, for instance. In this case even a warm restart of the cluster (without power interruption) will cause the un-restartable nodes to fail together.

When many blocks have only two replicas, if several nodes fail to join the cluster after a cluster start the number of lost blocks will be in proportion to the number of pairs of failed nodes. The result may be thousands of missing blocks. It is probable that those blocks can be recovered by copying from another cluster or by persuading the failed nodes to join the cluster, but that effort requires a day—or more—of work by an administrator. We are investigating automated recovery! This process dominates reports of lost blocks; no one pays attention to other possible causes anymore. For over a year, no one has done a block-by-block analysis of missing blocks as was done in 2009. The new disk fail-in-place feature of HDFS helps to alleviate the problem of un-restartable nodes.

But what of those problems that were previously known causes of missing blocks? All have been fixed in Hadoop 0.20.204. Only one new cause of missing blocks has been discovered in 2010. A data node that is unable to reliably read data from other nodes might attempt to read each replica of a block and report each replica as corrupt. In this case, the block looks to be missing even though all existing replicas are

good. It is possible for an administrator to recover from such false reports. HDFS has adopted the policy that a report of a corrupt block is only accepted from a node that has read a good replica.

Know Your Neighbors

Can users do anything to improve the robustness and availability of their own data? There is a wide variance among clusters of incidence of service or data unavailability. The clusters that host a more experimental user community experience more untoward events. The "production" clusters with restricted access are more robust than the "research" clusters available to all engineers. Especially with the introduction of permissions two years ago, HDFS users have substantial protection from the carelessness of others. The most immediate threat to data availability comes from an application job that exhausts some cluster resource. The quota features of HDFS are partial protections. How to yet better manage system resources is an area of active investigation. That said, HDFS operated by skilled administrators is a reliable custodian of Yahoo!'s petabytes.

Resources

K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010), May 2010.

Konstantin V. Shvachko, "HDFS Scalability: The Limits to Growth," *;login:*, vol. 35, no. 2, April 2010: http://www.usenix.org/publications/login/2010-04/openpdfs/shvachko.pdf.

Konstantin V. Shvachko, "Apache Hadoop: The Scalability Update," *;login:*, vol. 36, no. 3, June 2011: http://www.usenix.org/publications/login/2011-06/openpdfs/Shvachko.pdf.

Using Ganglia and Nagios: http://www.ibm.com/developerworks/linux/library/l-ganglia-nagios-1/.

HDFS tracking system: "HDFS-2535: A Model for Data Durability": https://issues.apache.org/jira/browse/HDFS-2535.

Server Message Block in the Age of Microsoft Glasnost

CHRISTOPHER R. HERTEL

Christopher R. Hertel is a long-haul member of the Samba Team and co-founder of the jCIFS project. He is also the author of *Implementing CIFS—The Common Internet File System,* the only developer's guide to the SMB/CIFS protocol suite. Not too long ago, he had the opportunity to work directly with Microsoft's File Server team when the company he founded, ubiqx Consulting, Inc., was tapped to write Microsoft's official SMB/CIFS specifications. Chris has also been adjunct faculty at the University of Minnesota College of Continuing Education (CCE) and is currently a member of the CCE IT Infrastructure Advisory Board.
crh@ubiqx.com

The EU anti-trust case against Microsoft concluded in late 2007. Related or not, that's when things started to change. One pleasant surprise for third-party developers was the release of hundreds of specifications covering Windows file formats, system internals, and protocols. Microsoft was opening up. Four years later, about 400 specifications have been published. It took a while for some of those documents to appear, mostly because Microsoft didn't actually have them all written yet. And now they have surprised us again. Well before the beta release of Windows 8, they have provided preview documentation for an overhauled and compelling new version of the venerable Server Message Block Protocol: SMB2.2. This is going to be epic.

A Gathering of Storage Geeks

Once a year, typically in September and typically somewhere near San Jose, California, the Storage Networking Industry Association (SNIA) hosts the Storage Developer Conference (SDC). The season and location don't really matter all that much, since many of those who attend never leave the confines of the hotel until the conference is over. As its name implies, the SDC is aimed at data storage engineers.

As a sideshow to the SDC, the SNIA hosts the annual SMB/CIFS/SMB2 Plugfest, an opportunity to bring together different products from different vendors and open source groups and do some heavy-duty interoperability testing.

The Plugfest is housed in its own conference room. The room is filled with network cables and switches, computers, tables with black tablecloths, a whiteboard, racks of storage equipment (with fans that sound like jet engines), and engineers. A badge is required to get in, but for those who participate there are only a few simple rules: Don't take stuff that's not yours, don't talk about your competitor's test results outside the Plugfest room, but do take a shower at least once every two days. They're serious about that last rule. They need to be. It gets intense in there.

The Ice Age

The Plugfest has been an annual event for more than a dozen years—longer, in fact, than the SDC—but for most of those years Microsoft, the "owner" of the SMB protocol suite, was conspicuous by their absence. While they were in the throes of legal battles (with not one but two major world governments) they kept their SMB code monkeys locked in their cages in Redmond.

During that time, SMB also suffered a dearth of documentation. There were some vintage OS/2 and DOS docs, and an unpolished, incomplete draft specification that Microsoft had presented to the IETF in 1996/97. The SNIA made a valiant attempt to create an updated version of the IETF draft, but without a solid commitment from Microsoft it too suffered from inaccuracies and omissions. There was also an implementer's guide (written by yours truly) that leveraged the collected knowledge of the third-party SMB development community. Since Microsoft's engineers weren't allowed out to play with the rest of the development community, and since there were no official specifications, SMB remained a protocol shrouded in myth and obscurity.

To make matters worse, when Microsoft lost in the US courts they furthered the divide by creating a licensing program that provided some of their competitors with access to Microsoft-internal information. That licensing program originally required signing an NDA (and possibly paying a fee, but we don't know for sure, because that part fell under the NDA). A couple of companies opted in, several others opted out, and open source groups such as the Samba Team were left without any options. The licensing program effectively split the SMB development community.

Those were the dark times—ash fell from the sky, a biting frost covered the land, the sun was only seen dimly behind the clouds, and SMB developers became melodramatic and prone to hyperbole. The Plugfest almost came to an end.

After Microsoft conceded in the EU in late 2007, things changed again. This time, however, Microsoft surprised just about everyone (who was paying attention) by going in completely the opposite direction. One of the first moves they made was to publicly release hundreds of documents covering Windows file formats, protocols, and software internals. They simply started posting them on the Web. Then they unlocked the cages and let their code monkeys out. Not only did they send engineers to the Plugfest, they went so far as to sponsor it so that others—notably smaller companies and open source groups—could afford to attend as well.

It was like springtime. The ice melted, the sun shone, the divide was healed, and the community started to come back together. In 2008, when Microsoft first sent a contingent to the Plugfest, a total of 14 organizations participated. By 2011 there were 27, almost twice as many.

Quick Sync: SMB and CIFS

Just to make sure we're all on the same page here, the SMB protocol is what Windows systems use to connect the Q: or Z: "drive" to a shared directory on a server—possibly a Windows server or, thanks to those storage engineers at the Plugfest, it could be any of several third-party implementations.

The naming could use a little clarification as well. Just before the start of the SMB Ice Age, when Microsoft submitted the draft specification to the IETF, they gave the protocol a marketing upgrade. That is, they renamed it. They decided that they needed to have the word "Internet" in the acronym, so they re-dubbed it Common Internet File System: CIFS. These days, the CIFS name is used interchangeably with SMB. They essentially mean the same thing.

SMB Evolution

In the computing industry, we measure time in nanoseconds. By that standard SMB is a fossil. It was originally created by IBM (not Microsoft!) in the 1980s as a network file protocol for PC-DOS, yet somehow it survives today as a key compo-

nent of Microsoft's Windows products. Not only is it still around, it is probably the most widely used and most successful network file system there is. Apple AFP? Novell Netware? Gone by the wayside. The only real competition still out there is NFS, which is as old and wizened as SMB.

All Windows systems, of course, support SMB. In addition, every major NAS platform and every major server OS (as well as many of the minor ones) supports SMB. Mainframes and desktops and tablets and cell phones support SMB. It's an interoperability requirement these days. If you find a NAS appliance for the home or enterprise or anything in between that doesn't support SMB, you should buy a lottery ticket—it's your lucky day.

Despite its importance, development of the SMB protocol remained relatively stagnant for years. In fact, SMB hasn't changed much at all since it was ported from OS/2 to Windows NT back in the early 1990s. When Windows 2000 was being developed, there were rumors that it would have a completely overhauled version of SMB to go along with the new CIFS name, but the overhaul didn't actually happen then. Windows 2000 introduced a few enhancements to access additional Windows OS features, improvements to the authentication subsystem, and changes to the entourage of protocols that support SMB. At the core, however, the SMB protocol remained the same.

Somewhere in the deep of the dark times, Microsoft finally decided that they needed something better. So, as part of Windows Vista, they introduced SMB version 2 (SMB2). SMB2 is a complete rewrite. The number of commands supported has been cut to 25% of its predecessor's, a reduction in bulk made possible by consolidating redundant commands and (hurray!) jettisoning support for DOS and OS/2 semantics. SMB2 also offers some useful new features, though none of those features are immediately visible to the user; SMB2 is a transparent upgrade. It is so transparent, in fact, that users and system administrators often don't even know it's there. It runs automatically instead of SMB if both the client and server agree to use it.

So SMB2 was a major cleanup but not a paradigm shift. A few additional features were added in SMB version 2.1, which appeared with Windows 7, but those were seen as incremental improvements. One major step forward, however, was documentation. Windows 7 came out after Microsoft had started to open up. That, and the realization that Windows XP was the only supported Windows OS that did not include SMB2 encouraged third-party developers to start thinking of SMB version 1 as the old reliable family minivan—it still runs, but it's too big and it's getting a little rusty around the edges. Time to move up to something sportier and more efficient, but we'll keep the old one parked in the back until it stops running entirely.

Call Me Triceratops

There are several presentation tracks at the SDC, covering all manner of storage technologies, from the mechanics of disk drives to the problems of defining standards for content-addressable archive metadata. There are also, of course, keynote speeches. At the 2010 SDC, a keynote slot opened up at the last minute and the organizers asked Samba Team luminary Jeremy Allison to fill the gap. Never fear, Jeremy's here. Somehow, he quickly threw together a remarkably memorable talk in which he asserted that the NFS and SMB developers in the room, himself included, were a herd of dinosaurs all trying to eat one another before the asteroid collision and the rise of the rodents.

Jeremy pointed out that most of the new and interesting work in network file systems was being done to create targeted solutions like Gluster, Hadoop HDFS, Ceph, and any number of other efforts aimed at solving specific sets of problems rather than trying to do a good enough job at everything to handle every use-case. General-purpose network file systems, he asserted, were a dying breed. A handful of them are still out there, but (as noted above) the mainstream market has narrowed its interest down to just two: NFS and SMB.

The Age of the Dinosaurs

During the dark times, while Microsoft was hibernating, the Internet Engineering Task Force (IETF) was busy overhauling NFS and doing so out in the open as part of a standards process. NFSv4 and 4.1 were getting snazzy new features such as parallel data access and transport over RDMA. SMB was getting left behind.

That was something else that changed when Microsoft let their engineers out to play with the other engineers. They saw the bright shiny things going into NFS, and they saw that the Samba Team had somehow managed to implement SMB clustering. Perhaps Microsoft had been planning an advanced feature set for SMB2 from the very start, but it wasn't until they announced version 2.2 at the 2011 SDC that it became clear that they, too, had been hard at work re-evaluating the market for general-purpose network file systems.

One year after Jeremy's keynote (which they made a point of acknowledging), Microsoft gave their own SDC keynote titled "The Future of File Protocols: SMB 2.2 in the Data Center." The presentation was full of surprises. In addition to providing an overview of new SMB2.2 features in Windows 8 and giving a live demo that actually worked, they also announced the availability of preview documentation so that others could start building implementations right away.

Here's some of what will be in SMB2.2:

- SMB2.2 supports clustering: Multiple SMB servers can be connected to multiple application servers, providing scalability as well as failover.
- SMB2.2 supports multi-path: There can be multiple connections between the client and the server.
- SMB over RDMA: SMB2.2 I/O commands can use machine-to-machine remote direct memory access (RDMA) to perform read/write operations over iWarp, InfiniBand, or plain old Ethernet.
- SMB2.2 provides lightweight WAN acceleration: This feature, called BranchCache, was introduced in SMB2.1, but needs to be enabled in order to be used. SMB2.2 supports an updated version of BranchCache that allows for improved content caching algorithms.

The most surprising thing about these features is that they were demonstrated live using a pre-alpha version of Windows 8 that had been compiled the night before by a guy named Dave, and the system never crashed. (It's also surprising that we know Microsoft SMB developers by name, even if only first name. That was never the case back in the dark times.) It's hard enough to get a live demo working with production code, so using pre-alpha code is downright scary. Just to make things more fun, they were connecting and disconnecting cables during the keynote to show how well the system failed over and scaled.

It was a very impressive demonstration, and fun to watch in the way that a daredevil act is fun to watch. For Windows users, SMB2.2 will be a major—and

visible—step forward. However, a quick look through the list of key features shows that there is nothing actually new here, technology-wise. SMB2.2 is running fast to catch up.

The Battle for the Niche

So if Jeremy is right, what we are about to witness is a battle for dominance between two large tyrannosaurs in what remains of the general-purpose file protocol ecosystem. NFS is ahead on features, but NFS implementers have been slow to create products, and users have quite reasonably been slower to adopt them. SMB has the advantage of Windows 8—when that comes out, both clients and servers will have SMB2.2 already installed and ready to deploy. It will simply "be there."

What this means for developers is that there is a lot of work to be done. On the NFS side, more v4 implementations need to become available on more platforms. Soon. On the SMB side, there are not enough third-party SMB2 implementations, and those that do exist will need to be updated to handle the fancy SMB2.2 feature set.

And there is one more logistical difference to consider. The rumor among Silicon Valley recruiters is that you can put up a sign saying "NFS Coders Needed" and a line will form, but that there simply aren't enough SMB developers to be found. Universities often teach basic NFS internals, and the specifications have long been available, so NFS know-how is a relatively common skill.

Not so with SMB. Most SMB developers are either senior engineers who have spent years working with the protocol and have learned it from the ground up, or senior engineers who became senior working on some other protocol and were then tapped by their employer to learn SMB. There are a few, but very few, newcomers in the SMB world.

Place your bets, the stakes are high. Data storage isn't the most glamorous field in modern computing, but it is fundamental. What happens in this arena in the next few years will have implications. Rodents take note: the dinosaurs still rule.

Resources

Christopher R. Hertel, *Implementing CIFS—The Common Internet File System* (online book), 2003: http://www.ubiqx.org/cifs/#Contents.

Christopher R. Hertel, 2011 SNIA SDC Report: Summary (blog post), Samba Team: http://www.samba.org/samba/news/developers/2011-snia-sdc-report.html.

Microsoft Corporation, [MS-CIFS]: Common Internet File System (CIFS) Protocol Specification, 2009: http://msdn.microsoft.com/en-us/library/ee442092.aspx.

Microsoft Corporation, [MS-SMB]: Server Message Block (SMB) Protocol Specification, 2007: http://msdn.microsoft.com/en-us/library/cc246231.aspx.

Microsoft Corporation, [MS-SMB2]: Server Message Block (SMB) Version 2 Protocol Specification, 2007: http://msdn.microsoft.com/en-us/library/cc246482.aspx.

Microsoft Preview Specifications (includes SMB2 updated to include SMB2.2 specifications; SMBD, which specifies SMB2 over RDMA Transport Protocol; MS-SWN: SMB2 Witness Protocol Specification; MS-SWN cluster failure/recovery notification): http://msdn.microsoft.com/en-us/library/ee941641.aspx.

NFSv4

ALEX MCDONALD

Alex joined NetApp in 2005, after more than 30 years in a variety of roles with some of the best-known names in the software industry (Legent, Oracle, BMC, and others). With a background in software development, support, and sales and a period as an independent consultant, Alex is now part of NetApp's Office of the CTO, which supports industry activities and promotes technology and standards-based solutions, and is chair of the SNIA CSI Education Subcommittee and co-chair of the SNIA NFS Special Interest Group. alexmc@netapp.com

NFSv4 has been a standard file-sharing protocol since 2003 but has not been widely adopted. Yet NFSv4 improves on NFSv3 in many important ways. In this article I explain how NFSv4 is better suited to a wide range of datacenter and HPC uses than its predecessor NFSv3, as well as providing resources for migrating from v3 to v4. And, most importantly, I make the argument that users should, at the very least, be evaluating and deploying NFSv4.1 for use in new projects and, ideally, should be using it wholesale in their existing environments.

The Background to NFSv4.1

NFSv2 and its popular successor NFSv3 (specified in RFC-1813 [1], but never an Internet standard) was first released in 1995 by Sun. It has proved to be a popular and robust protocol over the 15 years it has been in use, and with wide adoption it soon eclipsed some of the early competitive UNIX-based filesystem protocols such as DFS and AFS. NFSv3 was extensively adopted by storage vendors and OS implementers beyond Sun's Solaris; it was available on an extensive list of systems, including IBM's AIX, HP's HP-UX, Linux, and FreeBSD. Even non-UNIX systems adopted NFSv3; Mac OS, OpenVMS, Microsoft Windows, Novell NetWare, and IBM's AS/400 systems. In recognition of the advantages of interoperability and standardization, Sun relinquished control of future NFS standards work, and work leading to NFSv4 was by agreement between Sun and the Internet Society (ISOC) and is undertaken under the auspices of the Internet Engineering Task Force (IETF).

In April 2003 the Network File System (NFS) version 4 protocol was ratified as an Internet standard, described in RFC-3530, which superseded NFSv3. This was the first open file system and networking protocol from the IETF. NFSv4 introduced the concept of state to ameliorate some of the less desirable features of NFSv3 and offered other enhancements to improve usability, management, and performance.

But shortly following NFSv4's release, an Internet draft written by Garth Gibson and Peter Corbett outlined several problems with it [2]: specifically, that of limited bandwidth and scalability, since NFSv4, like NFSv3, requires that access be to a single server. NFSv4.1 (as described in RFC-5661, ratified in January 2010) was developed to overcome these limitations, and new features such as parallel NFS (pNFS) were standardized to address these issues.

NFSv4.2 is now moving towards ratification [3]. In a change to the original IETF NFSv4 development work, where each revision took a significant amount of time

to develop and ratify, the work group charter was modified to ensure that there would be no large standards documents that took years to develop, such as RFC-5661, and that additions to the standard would be an ongoing yearly process. With these changes in the processes leading to standardization, features that will be ratified in NFSv4.2 (expected in March 2012) are available from many vendors and suppliers now.

Adoption of NFSv4

While there have been a number of advances and improvements to NFS, many users have elected to continue with NFSv3. NFSv4 is a mature and stable protocol with significant advantages over its predecessors NFSv3 and NFSv2, yet adoption remains slow. Adequate for some purposes, NFSv3 is a familiar and well understood protocol; but with the demands being placed on storage by exponentially increasing data and compute growth, NFSv3 has become increasingly difficult to deploy and manage.

So, What's the Problem with NFSv3?

In essence, NFSv3 suffers from problems associated with statelessness. While some protocols, such as HTTP and other RESTful APIs, see benefit from not associating state with transactions—it considerably simplifies application development if no transaction from client to server depends on another transaction—in the NFS case, statelessness has led, among other downsides, to performance and lock management issues.

NFSv4.1 and parallel NFS (pNFS) address well-known NFSv3 "workarounds" that are used to obtain high bandwidth access; users who employ (usually very complicated) NFSv3 automounter maps and modify them to manage load balancing should find that pNFS provides comparable performance that is significantly easier to manage.

Extending the use of NFS across the WAN is difficult with NFSv3. Firewalls typically filter traffic based on well-known port numbers, but if the NFSv3 client is inside a firewalled network and the server is outside the network, the firewall needs to know what ports the portmapper, mountd, and nfsd servers are listening on. As a result of this promiscuous use of ports, the multiplicity of "moving parts," and a justifiable wariness on the part of network administrators to punch random holes through firewalls, NFSv3 is not practical to use in a WAN environment. By contrast, NFSv4 integrates many of these functions and mandates that all traffic (now exclusively TCP) uses the single well-known port 2049.

One of the most annoying NFSv3 "features" has been its handling of locks. Although NFSv3 is stateless, the essential addition of lock management (NLM) to prevent file corruption by competing clients means that NFSv3 application recovery is slowed considerably. Very often, stale locks have to be manually released, and the lock management is handled external to the protocol. NFSv4's built-in lock leasing, lock timeouts, and client-server negotiation on recovery simplify management considerably.

In a change from NFSv3, these locking and delegation features make NFSv4 stateful, but the simplicity of the original design is retained through well-defined recovery semantics in the face of client and server failures and network partitions. These are just some of the benefits that make NFSv4.1 desirable as a modern datacenter protocol and for use in HPC, database, and highly virtualized applications.

The Advantages of NFSv4.1

The Gibson and Corbett paper [2] identified some issues with NFSv4 that were successfully addressed in NFSv4.1, and NFSv4.1 is where the focus for end-user evaluation and implementation should be. References to features in NFSv4 apply equally to NFSv4.1, since it was a minor version update, unlike the changes from NFSv3 to NFSv4. Many of these base NFSv4 features, such as the pseudo file system, are covered by my SNIA white paper "Migrating from NFSv3 to NFSv4" [4] and will not be covered here.

Internationalization Support: UTF-8

In a welcome recognition that the ASCII character set no longer provides the descriptive capabilities demanded by languages with larger alphabets or those that use an extensive range of non-Roman glyphs, NFSv4 uses UTF-8 for file names, directories, symlinks, and user and group identifiers. As UTF-8 is backwards compatible with 7-bit encoded ASCII, any names that are 7-bit ASCII will continue to work.

Compound RPCs

Latency in a WAN is a perennial issue and is very often measured in tenths of a second to seconds. NFS uses RPC to undertake all its communication with the server, and although the payload is normally small, metadata operations are largely synchronous and serialized. Operations such as file lookup (LOOKUP), the fetching of attributes (GETATTR), and so on, make up the largest percentage by count of the average workload (Table 1).

NFSv3 Operation	SPECsfs2008
GETATTR	26%
LOOKUP	24%
READ	18%
ACCESS	11%
WRITE	10%
SETATTR	4%
READDIRPLUS	2%
READLINK	1%
READDIR	1%
CREATE	1%
REMOVE	1%
FSSTAT	1%

Table 1: SPECsfs2008 %ages for NFSv3 operations [5]

This mix of a typical NFS set of RPC calls in versions prior to NFSv4 requires that each RPC call be a separate transaction over the wire. NFSv4 avoids the expense

of single RPC requests, and the attendant latency issues, and allows these calls to be bundled together. For instance, a lookup, open, read, and close can be sent once over the wire, and the server can execute the entire compound call as a single entity. The effect is to considerably reduce latency for multiple operations.

Delegations

Servers are employing ever more quantities of RAM and flash technologies, and very large caches, on the order of terabytes, are not uncommon. Applications running over NFSv3 can't take advantage of these caches unless they have specific application support. With increasing WAN latencies, doing every I/O over the wire introduces significant delay.

NFSv4 allows the server to delegate certain responsibilities to the client, a feature that allows caching locally where the data is being accessed. Once delegated, the client can act on the file locally with the guarantee that no other client has a conflicting need for the file; it allows the application to have locking, reading, and writing requests serviced on the application server without any further communication with the NFS server. To prevent deadlocking conditions, the server can recall the delegation via an asynchronous callback to the client should there be a conflicting request for access to the file from a different client.

Migration, Replicas, and Referrals

For broader use within a datacenter, and in support of high availability applications such as databases and virtual environments, copying data for backup and disaster recovery purposes and the ability to migrate data to provide VM location independence are essential. NFSv4 provides facilities for transparent replication and migration of data, and the client is responsible for ensuring that the application is unaware of these activities. An NFSv4 referral allows servers to redirect clients from this server's namespace to another server; it allows the building of a global namespace while maintaining the data on discrete and separate servers.

Sessions

Sessions bring the advantages of correctness and simplicity to NFS semantics. In order to improve the correctness of NFSv4, NFSv4.1 sessions introduce "exactly-once" semantics. Servers maintain one or more session states in agreement with the client; a session maintains the server's state relative to the connections belonging to a client. Clients can be assured that their requests to the server have been executed, and that they will never be executed more than once. Sessions extend the idea of NFSv4 delegations, which introduced server-initiated asynchronous callbacks; clients can initiate session requests for connections to the server. For WAN-based systems, this simplifies operations through firewalls.

Security

One area of great confusion is that many believe that NFSv4 *requires* the use of strong security. The NFSv4 specification simply states that *implementation* of strong RPC security by servers and clients is mandatory, not the *use* of strong RPC security. This misunderstanding may explain the reluctance of users to migrate to NFSv4, due to the additional work in implementing or modifying their existing Kerberos security.

Security is increasingly important as NFSv4 makes data more easily available over the WAN. This feature was considered so important by the IETF NFS working group that the security specification using Kerberos v5 [6] was "retrofitted" to NFSv2 and NFSv3 and specified in RFC-2623.

Although access to an NFSv2, 3, or 4 filesystem without strong security such as provided by Kerberos is possible, across a WAN it should really be considered only as a temporary measure. In that spirit, it should be noted that *NFSv4 can be used without implementing Kerberos security* [7]. The fact that it is possible does not make it desirable! A fuller description of the issues and some migration considerations can be found in the SNIA white paper [4].

Many of the practical issues faced in implementing robust Kerberos security in a UNIX environment can be eased by using a Windows Active Directory (AD) system. Windows uses the standard Kerberos protocol as specified in RFC 1510; AD user accounts are represented to Kerberos in the same way as accounts in UNIX realms. This can be a very attractive solution in mixed-mode environments [8].

Parallel NFS (pNFS) and Layouts

Parallel NFS (pNFS) represents a major step forward in the development of NFS. Ratified in January 2010 and described in RFC-5661, pNFS depends on the NFS client understanding how a clustered filesystem stripes and manages data. It's not an attribute of the data, but an arrangement between the server and the client, so data can still be accessed via non-pNFS and other file access protocols. pNFS benefits workloads with many small files, or very large files, and is suitable for a range of HPC-type workloads.

Clients request information about data layout from a metadata server (MDS) and get returned layouts that describe the location of the data. (Although often shown as separate, the MDS is not normally a standalone system but is part of the facilities the cluster provides.) The data may be on many dataservers and is accessed directly by the client over multiple paths. Layouts can be recalled by the server, as is the case for delegations, if there are conflicting multiple client requests.

By allowing the aggregation of bandwidth, pNFS relieves performance issues that are associated with point-to-point connections. The parallel nature of the client connecting directly to multiple dataservers ensures that no single storage node is a bottleneck and that data can be better load-balanced to meet the needs of the client (see Figure 1).

The pNFS specification also accommodates support for many different layouts. Currently, three layouts are specified: files as supported by NFSv4, objects based on the Object-based Storage Device Commands (OSD) standard (INCITS T10) approved in 2004, and block layouts (either FC or iSCSI access).

Many vendors have provided vendor-specific modifications that provide similar functionality to pNFS. So although pNFS is relatively new and there may appear to be a limited amount of best practice advice for employing it, the experience of users with proprietary extensions to NFSv3 systems shows that high bandwidth access to data with pNFS will be of considerable benefit.

Potential performance of pNFS is definitely superior to that of NFSv3 for similar configurations of storage, network, and server. The management is definitely easier, as NFSv3 automounter maps and hand-created load-balancing schemes are

eliminated and, by providing a standardized interface, pNFS ensures fewer issues in supporting multi-vendor NFS server environments.

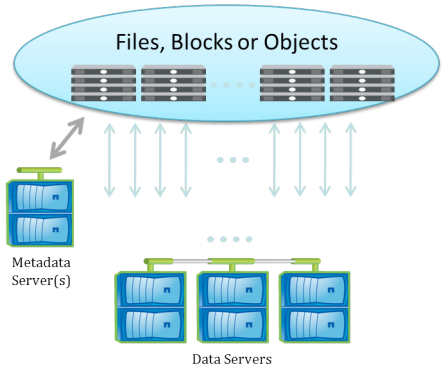


Figure 1: Conceptual pNFS data flow

Some Proposed NFSv4.2 Features

NFSv4.2 promises many features that end users have been requesting, which will make NFS more relevant as not only an "every day" protocol, but one that has application beyond the datacenter.

Server-Side Copy (SSC)

SSC removes one leg of a copy operation. Instead of reading entire files or even directories of files from one server through the client and then writing them out to another, SSC permits the destination server to communicate directly to the source server without client involvement, and it removes the limitations on server-to-client bandwidth and the possible congestion they may cause.

Application Data Blocks (ADB)

ADB allows definition of the format of a file: for example, a VM image or a database. This feature will allow initialization of data stores; a single operation from the client can create a 300 GB database or a VM image on the server.

Guaranteed Space Reservation and Hole Punching

As storage demands continue to increase, various efficiency techniques can be employed to give the appearance of a large virtual pool of storage on a much smaller storage system. Thin provisioning, (where space appears available and reserved, but is not committed) is commonplace but is often problematic to manage in fast-growing environments. The guaranteed space reservation feature in NFSv4.2 will ensure that, regardless of the thin provisioning policies, individual files will always have space available for their maximum extent.

While such guarantees are a reassurance for the end user, they don't help the storage administrator in his or her desire to fully utilize all available storage. In support of better storage efficiencies, NFSv4.2 will introduce support for sparse files, commonly called "hole punching": deleted and unused parts of files are returned to the storage system's free space pool (Figure 2, next page).

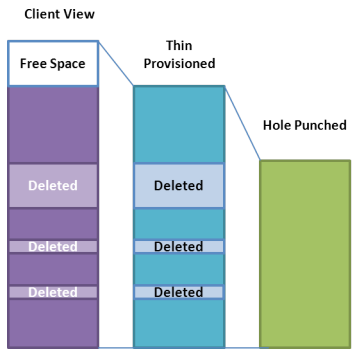


Figure 2: Thin-provisioned and hole-punched data

Availability of Servers and Clients

With this background on the features of NFS, there is considerable interest in the end-user community for NFSv4.1 support from both servers and clients. Many network attached storage (NAS) vendors now support NFSv4, and in recent months there has been a flurry of activity and many developments in server support of NFSv4.1 and pNFS. For NFS server vendors, refer to their Web sites, where you will get up-to-date information.

On the client side, there is Linux Red Hat 6.2, which has an NFSv4.1 Technical Preview [9], and Fedora 15 and 16, available at fedoraproject.org. Both distributions support files-based NFSv4.1 and pNFS. For the adventurous who wish to build their own kernels and want to explore the latest block or objects access, there is full file, block, and object-based pNFS support in the upstream 3.0 and 3.1 Linux kernels.

For Windows, Microsoft has publicly indicated that it will be supporting NFSv4.1 in Windows 8. An open source client implementation preview is available from the Center for Information Technology Integration (CITI) at the University of Michigan [10].

Conclusion

NFSv4.1 includes features intended to enable its use in global wide area networks (WANs). These advantages include:

- Firewall-friendly single port operations
- Advanced and aggressive cache management features
- Internationalization support
- Replication and migration facilities
- Optional cryptography quality security, with access control facilities that are compatible across UNIX and Windows
- Support for parallelism and data striping

The goal for NFSv4.1 and beyond is to define how you get to storage, not what your storage looks like. That has meant inevitable changes. Unlike earlier versions of NFS, the NFSv4 protocol integrates file locking, strong security, operation coalescing, and delegation capabilities to enhance client performance for data-sharing applications on high-bandwidth networks.

NFSv4.1 servers and clients provide even more functionality, such as wide striping of data, to enhance performance. NFSv4.2 and beyond promise further enhancements to the standard that will increase its applicability to today's application requirements. It is due to be ratified in March 2012, and we can expect to see server and client implementations that provide NFSv4.2 features soon after this; in some cases, the features are already being shipped now as vendor-specific enhancements.

With careful planning, migration to NFSv4.1 and NFSv4.2 from prior versions can be accomplished without modification to applications or the supporting operational infrastructure, for a wide range of uses—home directories, HPC storage servers, backups, and so on.

Resources

[1] NFSv3 specification: http://tools.ietf.org/html/rfc1813. Other IETF RFCs mentioned in the text can be found at the same site.

[2] The "pNFS Problem Statement": http://tools.ietf.org/html/draft-gibson-pnfs-problem-statement-01.

[3] NFSv4.2 proposed specification: http://www.ietf.org/id/draft-ietf-nfsv4-minorversion2-06.txt; the draft as of November 2011.

[4] Alex McDonald (SNIA white paper), "Migrating from NFSv3 to NFSv4": http://www.snia.org/sites/default/files/Migrating_NFSv3_to_NFSv4-Final.pdf.

[5] http://www.spec.org/sfs2008/docs/usersguide.html#_Toc191888936 gives a typical mix of RPC calls from NFSv3.

[6] "Kerberos Overview—An Authentication Service for Open Network Systems": http://www.cisco.com/application/pdf/paws/16087/1.pdf.

[7] For examples of NFSv4 without Kerberos, see Ubuntu Linux, https://help.ubuntu.com/community/NFSv4Howto, and SUSE Linux Enterprise, http://www.novell.com/support/dynamickc.do?cmd=show&forward=nonthreadedKC&docType=kc&externalId=7005060&sliceId=1.

[8] Windows Security and Directory Services for UNIX Guide v1.0: http://technet.microsoft.com/en-us/library/bb496504.aspx.

[9] Red Hat 6.2 NFSv4.1 Technical Preview: http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch12s02.html.

[10] NFSv4 Client for Windows at CITI: http://www.citi.umich.edu/projects/nfsv4/windows/.

Tasting Client/Network/Server Pie

STUART KENDRICK

Stuart Kendrick works as a third-tier tech at the Fred Hutchinson Cancer Research Center in Seattle, where he dabbles in trouble-shooting, deep infrastructure design, and developing tools to monitor and manage devices. He started earning money as a geek in 1984, writing in FORTRAN on Cray-1s for Science Applications International Corporation, worked in desktop support, server support, and network support at Cornell University, and reached FHCRC in 1993. He has a BA in English, contributes to BRIITE (http://www.briite.org), and spends free time on yoga and CrossFit.
skendric@fhcrc.org

"The system is slow." How often do we hear those words? Here is one technique I use to start narrowing the fault domain, to better focus my attention on where the cause of the slowness resides. This approach relies on comparing packet traces taken near the client and near the server.

From a high level, I see three players contributing to the performance characteristics of an application: the client, the network, and the server. With this model in mind, I calculate how much time the client spends formulating requests, how much time the server spends formulating responses, and how much time the network spends flinging the packets back and forth. From there, I can represent the relative contribution of each component as a slice in a pie, which naturally focuses my attention on the largest slice (slowest component).

I will sketch three techniques (manual, semi-automated, mostly automated) for constructing the pie and illustrate how this approach can be useful for analyzing application performance issues.

Capture the Pie

The Campus Network (Figure 1) in this article consists of a classic access/distribution/core layer design, where the access layer functions at Layer 2 (k2-esx and s4-esx) and the distribution and core layers function at Layer 3. In this diagram I hide the distribution and core layers inside the cloud. In the pies below, the clients Europa and Deimos copy files across the Campus Network to the server Mars, while the probes Hale, Bopp, and Tempel capture the relevant traffic.

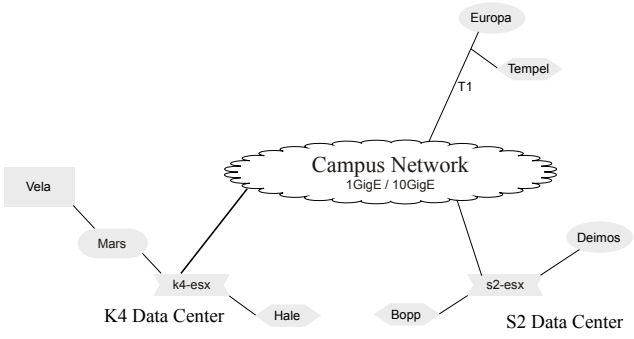


Figure 1: Campus Network

To make pies, I capture [1] two simultaneous packet traces of the experience from separate locations: one near the server (via the probe Hale), the other near the client (via the probe Tempel in the case of Europa, or via the probe Bopp in the case of Deimos). I filter the two traces, such that they include only traffic between client and server and only traffic relevant to the application I'm analyzing:

```
hale# dumpcap –i eth0 –w server.pcap –f "ip host europa and ip host mars"
```

Make the Pie from Scratch

This technique relies heavily on the DeltaT column in a trace. Here are the first handful of frames from a 10 MB SMB file copy from the client Europa to the server Mars across a T1 incurring 250 ms of latency.

Client-side trace:

No.	DeltaT	RelT	Bytes	Src	Dst	Info
1	0.000000	0.000000	70	Client	Server	TCP SYN
2	0.253718	0.253718	70	Server	Client	TCP SYN/ACK
3	0.000386	0.254104	64	Client	Server	TCP ACK
4	0.000007	0.254111	217	Client	Server	Negotiate Protocol Request
5	0.252704	0.506815	64	Server	Client	TCP ACK
6	0.256627	0.763442	226	Server	Client	Negotiate Protocol Response

Server-side trace:

No.	DeltaT	RelT	Bytes	Src	Dst	Info
1	0.000000	0.000000	70	Client	Server	TCP SYN
2	0.000059	0.000059	70	Server	Client	TCP SYN/ACK
3	0.254032	0.254091	64	Client	Server	TCP ACK
4	0.000272	0.254363	217	Client	Server	Negotiate Protocol Request
5	0.000410	0.254773	64	Server	Client	TCP ACK
6	0.257468	0.512241	226	Server	Client	Negotiate Protocol Response

The DeltaT (Delta Time) column records the time that has elapsed since the previous frame. The RelT (relative time, sometimes called cumulative time) column records time elapsed from the beginning of the trace through that frame.

Consider the trace taken close to the client. When the source address is the client, then DeltaT fairly closely represents the amount of time the client spent processing the previous message from the server. When the source address is the server, then DeltaT represents the time the server spent processing the client's request plus the time the network spent transmitting the server's response.

The converse holds for the trace taken close to the server. By using a little arithmetic, we can estimate the time the network contributed to the experience.

1. Filter the client-side trace so that we see only frames sourced from the client:

No.	DeltaT	RelT	Bytes	Src	Dst	Info
1	0.000000	0.000000	70	Client	Server	TCP SYN
3	0.000386	0.254104	64	Client	Server	TCP ACK
4	0.000007	0.254111	217	Client	Server	Negotiate Protocol request

2. Sum the DeltaT column to produce the client time estimate:

```
0.000000 + 0.000386 + 0.000007 = 0.000393s
```

3. Filter the server-side trace so that we see only frames sourced from the server:

```
No. DeltaT     RelT       Bytes  Src     Dst      Info

2   0.000059   0.000059    70    Server  Client   TCP SYN/ACK
5   0.000410   0.254773    64    Server  Client   TCP ACK
6   0.257468   0.512241   226    Server  Client   Negotiate Protocol Response
```

4. Sum the DeltaT column to produce the server time estimate:

```
0.000059 + 0.000410 + 0.257468 = 0.257937s
```

5. Estimate the network's contribution by grabbing the relative time from either trace and subtracting the client and server contributions:

```
Relative – (Client + Server) = Network
0.763442 – (0.000393 + 0.257937) = 0.505112s
```

6. Calculate the size of each slice in the CNS Pie:

```
Client Time   0.000393s
Server Time   0.257937s
Network Time  0.505112s
Total Time    0.763442s

Client%   = Client Time   / Relative Time = 0.000393 / 0.763442 = 0%
Server%   = Server Time   / Relative Time = 0.257937 / 0.763442 = 34%
Network%  = Network Time  / Relative Time = 0.505112 / 0.763442 = 66%
```

The skeptical reader may question why I plucked relative time from the client-side trace rather than from the server-side trace—in this trivial example, I agree that the choice makes a difference (using server-side RelT results in 50% server time and 50% network time, as opposed to the 66% and 34% produced above). However, I claim that the two will be identical, or nearly so, across large traces, and thus we can arbitrarily choose either one.

Naturally, my fingers become tired of punching buttons on a calculator, so I script [2] the process, invoking tshark (part of the Wireshark suite) to produce appropriately filtered text files containing just the summary lines, per above, then crawling through those text files while summing DeltaT. Attentive readers who examine the code will notice that I use a more laborious method for estimating the network contribution than the one sketched here.

As it turns out, once we chew [3] through all 10,776 frames in each of these traces, the results turn out as follows:

```
Client %    2.8s   / 63.5s =  4%
Server %    5.7s   / 63.5s =  9%
Network %  55s     / 63.5s =  87%
```

Use a Food Processor

Alternatively, tshark will perform the calculation for us [4], delivering numbers that are within a few percent of the ones I produce above using my home-grown code.

```
guru> tshark -nlr europa-to-mars-T1-250ms-at-europa.pcap –o
tcp.calculate_timestamps:TRUE -R "(tcp.dstport==445)"
-qz io,stat,600,"SUM(tcp.time_delta)tcp.time_delta"

================================================================
IO Statistics
Interval: 600.000 secs
Column #0: SUM(tcp.time_delta)tcp.time_delta
                  |     Column #0
Time              |     SUM
000.000-600.000         2.6
================================================================
guru>

guru> tshark -nlr europa-to-mars-T1-250ms-at-mars.pcap –o
tcp.calculate_timestamps:TRUE -R "(tcp.srcport==445)"
-qz io,stat,600,"SUM(tcp.time_delta)tcp.time_delta"

================================================================
IO Statistics
Interval: 600.000 secs
Column #0: SUM(tcp.time_delta)tcp.time_delta
                  |     Column #0
Time              |     SUM
000.000-600.000         6.0
================================================================
guru>
```

Capinfos, another Wireshark utility, tells us how long the trace lasted:

```
guru> capinfos europa-to-mars-T1-250ms-at-europa.pcap
File name: europa-to-mars-T1-250ms-at-europa.pcap
[…]
Capture duration: 63 seconds
[…]
```

Knowing that client time is 2.6 seconds, server time is 6.0 seconds, and total time is 63 seconds, we can calculate network time:

```
Network Time  = 63s – 2.6s – 6s = 53.4s
```

Calculate percentages:

```
Client %  2.6s   / 63s = 4%
Server %  6.0s   / 63s = 10%
Network %53.4s   / 63s = 85%
```

Finally, in Figure 2 (next page), we use our favorite charting program to produce the first pie.

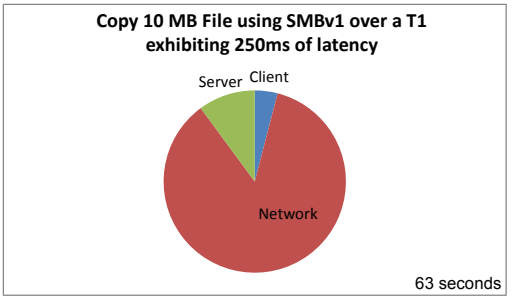


Figure 2: Our first pie

Buy the Pie from a Bakery

For those of us with money to spend, consider purchasing commercial software to provide a more sophisticated estimate of these three components. With these tools, we import the two traces into the analysis software, which then performs the tedious work described above. In Figure 3, I use Fluke Networks' ClearSight Analyzer to produce a stacked chart, functionally equivalent to a pie.

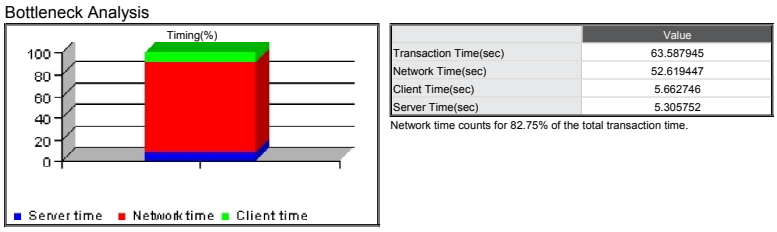


Figure 3: Bottleneck analysis chart from ClearSight Analyzer

More subtly, there are a range of issues which the home-baked or food-processed approaches miss, including TCP window size, TCP congestion window, application block size, packet loss, and parallel threads. As these factors arise in your situation, the manual approaches become increasingly inaccurate, and this is where the introspection baked into the commercial applications shines. Commercial packages also support importing more than two traces, captured at various points along the path between client and server, and are smart enough to track transactions through middleware (e.g., browser to Web server to back-end database).

Try a Slice

In these pies, Deimos copies files to Mars using NFSv3. Figure 4 illustrates situations in which I want to focus attention on the client, as it contributes 80–90% of the total transaction time.

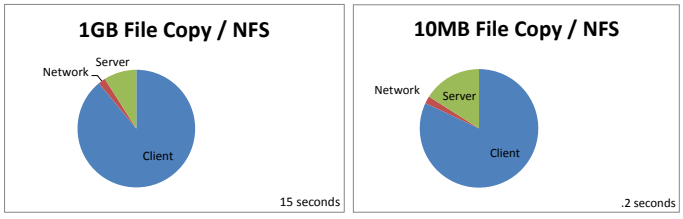


Figure 4: Copy big files across Campus Network

In Figure 5, I want to focus attention on the server, as it contributes 60–70% of the total time.

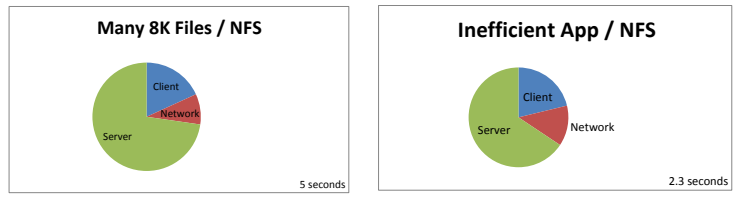


Figure 5: Copy small files across Campus Network

The Many 8K Files test involves copying a thousand 8K files, while the Inefficient App copies a single 1K file one byte at a time, closing and re-opening the file between each byte:

```
#!/usr/bin/perl
# Inefficient application
[…]
$destination = '/mnt/server';
$source = '/var/tmp/test_file';
open $read_fh, '<', $source;  # Open source file
while (read $read_fh, my $tmp, 1) { # Read next byte from the source file
       open $write_fh, '>>', $destination; # Open destination file
       print {$write_fh} $tmp;  # Write this byte to destination file
       close $write_fh;   # Close destination file
}
```

For the one repository I analyzed during this job, Inefficient App turned out to be a dead-ringer for Subversion, making it useful for modeling Subversion behavior (see Figure 6) when testing new client / network / server combinations.

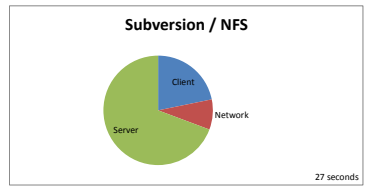


Figure 6: Subversion across Campus Network

Direct Our Attention

The pies suggest that for large file copies—a streaming application—we direct our attention toward Deimos. On investigation, we might find that beefing up its CPU or giving it a faster drive reduces the total transaction time.

For transactional applications, such as copying many small files or Subversion, the pies suggest that we direct our attention toward Mars. On investigation, we find that Mars is backed by the mass storage device Vela, which contains ~500 spindles of 1 TB and 2 TB 10 K SATA drives working in parallel. Vela divides each disk into ~700 MB chunklets, picks at least one chunklet from each disk, and glues them together to produce the LUN which Mars exposes to Deimos.

As a result, Mars performs well for streaming applications and poorly for transactional applications. Why? When we copy a big file to Mars, 500 spindles work together to swallow the datastream: the single spindle inside Deimos cannot keep up, making Deimos the primary contributor to the pie. When we copy many small files one at a time, the two systems are more evenly matched: only a single spindle inside Vela handles each write request, and that request must complete before Deimos can forward the next request.

For Mars, we might experiment with adding a LUN serviced by small, fast spindles—say, a dozen 15 K 250 MB drives—and moving Subversion to a volume hosted on that LUN. These platters are small and they rotate rapidly, reducing seek latency. For transactional applications, we might predict that such a LUN would deliver faster performance.

The CNS Pie provides a visually intuitive tool for narrowing the fault domain and for communicating the contours of the issue to our colleagues [5].

Too Much Sugar

Yes, I've been oversimplifying. Sure, sometimes the CNS Pie accurately directs our attention to the bottleneck. But the real world can be complex, and there are plenty of times when the CNS Pie misdirects our attention.

In Figure 7, both pies suggest that we focus on the network in order to improve performance, but notice how the transaction time drops from 64 seconds to 14 seconds when we upgrade from SMBv1 to v2.

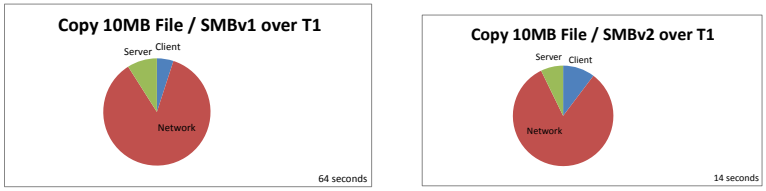


Figure 7: SMBv1 vs. SMBv2 over T1 with 250 ms latency

A simplistic reading of the left-hand CNS Pie would have focused our attention on the network, which might have pushed us to purchase a network pipe with more throughput. This would not have helped: SMBv1's *application block size* is 61 K, and so once latency reaches ~40 ms, no amount of fatter pipe will improve performance [6]. We could have purchased a 10 GigE network service (assuming latency remained the same) without improving total time. On the other hand, by upgrading the client and server to run SMBv2 (available in Windows Vista+ and Samba 3.5+), we improved performance by a factor of ~5. This new version of SMB auto-tunes its application block size and streamlines metadata operations, thus improving performance.

While I find the CNS Pie useful in shedding light on application performance issues, it remains only one tool in the toolkit: there are no silver bullets.

Appreciation

I feel particular gratitude to Mike Pennacchi of Network Protocol Specialists for teaching me to make my first CNS Pie. (As far as I can tell, Mike invented the

CNS Pie back in the 1990s. If you know of prior art, please drop me a note.) I also appreciate the professionals who have given their time to coach me on the topics covered in this article: Glenn Boyle of BT Global Services for opening my eyes to how I could bake at home, for refining my recipes, for teaching me how to use a food processor, and for helping me understand numerous subtleties; Gary Kaiser of Compuware for help understanding yet more subtleties plus the sophistication which commercial products bring to this space; and my colleagues Robert McDermott, for teaching me about the complex world of storage systems, and Wolfe Maykut, for the Inefficient App.

Thank you also to the community active on the LinkedIn Protocol Analysis and Troubleshooting group—I appreciate your contributions to the rich discussions there.

References

[1] I use dumpcap, tcpdump, and tshark interchangeably, depending on mood—their syntax is almost identical. For those interested in high-performance capture, check out Corey Satten's gulp: http://staff.washington.edu/corey/gulp.

[2] Data mangling code is available at http://www.skendric.com/app/code/extract-summary-lines-from-pcap and http://www.skendric.com/app/code/calculate-cns-pie.

[3] For a detailed description of this process, see http://www.skendric.com/app/make-cns-pie/Make-Client-Network-Server-Pie.pdf.

[4] Requires Wireshark 1.7.1 or later.

[5] For more examples of how the CNS Pie can direct our attention, see the "Make Client/Network/Server Pie" article at http://www.skendric.com/app.

[6] Bandwidth Delay Product calculation: 1,544,000 b/s * .04s = 61,760 bits =~ 61K.

Netflix Heads into the Clouds

Interview with Adrian Cockcroft

RIK FARROW

Adrian Cockcroft is the director of architecture for the Cloud Systems team at Netflix. He is focused on availability, resilience, performance, and measurement of the Netflix cloud platform. Adrian is also well known as the author of several books while a Distinguished Engineer at Sun Microsystems: *Sun Performance and Tuning*, *Resource Management*, and *Capacity Planning for Web Services*. From 2004 to 2007 he was a founding member of eBay Research Labs. He graduated with a BSc in Applied Physics from The City University, London.
acockcroft@netflix.com

I first heard about Netflix's remarkable journey into the cloud in Adrian Cockcroft's presentation at HPTS (see the reports in this issue). Adrian explained why Netflix was moving to the cloud after having had their own datacenter: the cloud was both easier to work with and more reliable. Adrian's presentation slides [1] as well as very detailed blog entry [2] do a great job of explaining where Netflix is today, and where they plan to go. I suggest that you read both of these resources; the slides present an overview, and the blog gets into details. What is missing from these resources is a bit of history, which Adrian provides here.

Rik: I am guessing that Netflix started out using a big server in a datacenter, pretty much the way other companies had always done, and you gradually learned about the advantages of not having your own servers. Could you tell us a little about the past that led you to this point?

Adrian: The history of how we got to the cloud is that our original datacenter systems were based on a few large Oracle servers with a Java front end. The load at that time was dominated by the DVD business. We had a storage data corruption bug in the summer of 2008 which took Netflix down for several days, and we also saw that in the future we would need to rebuild our site for higher availability to support the demands of streaming. It was clear that we needed large-scale redundant datacenters, but we couldn't easily predict how much datacenter capacity we would need for the rapidly growing streaming business and where it should be located. We had also had a fairly painful experience moving from a single datacenter to a pair of small ones (leased cages), and needed to decide whether to invest heavily in the staff and skills needed to run a large and high-growth-rate datacenter infrastructure, or outsource and leverage an external cloud supplier. In parallel we ran a large upgrade of datacenter capacity and an investigation into the feasibility of using cloud. We quickly settled on AWS as the largest cloud and established an executive-level relationship as a foundation for the business and technical relationship.

Through 2009, we explored the cloud platform with several pathfinder projects and non-customer-facing workloads such as encoding and Hadoop-based log analysis. In early 2010 we brought up the first customer-facing workloads, starting with the simplest ones with fewest dependencies, and gradually filling in the data sources until almost everything is running in the cloud, but with the data resident in both cloud and datacenter. In 2011 we gradually moved the "source of truth" systems into the cloud, with copies in the datacenter as needed. The final stages of that are currently being completed.

Rik: I understand that you have built many tools for managing and monitoring your servers and software in the cloud, and that you have made them open source. Can you briefly tell us about them?

Adrian: We have built a platform that runs on the AWS cloud but provides abstractions and patterns that are more portable and convenient for the developers at Netflix. I describe this PaaS in a presentation given at QConSF in October; slides are at http://www.slideshare.net/adrianco/global-Netflix-platform.

The platform is primarily based on Java running in Tomcat on Linux. We also support the Groovy/Grails environment, primarily for building internal tools. We use open source components combined with custom code. The Netflix Global PaaS has the following features:

- Global distribution of traffic, processing, and data
- Localized support for multiple languages and jurisdictions
- Support for dynamic and ephemeral cloud resources
- Data migration mechanisms from datacenter to cloud and between regions
- Continuous backup and secure distributed archive of cloud-based data
- Dynamic security key management with multi-level key protection
- Fine-grained least privilege security based on AWS security groups and IAM
- Scalable to many thousands of instances, autoscaled with load

The components that we are open sourcing at github.com include:

- Curator: a distributed coordination framework based on Apache Zookeeper
- Priam: Tomcat-based automation for simple management of Apache Cassandra on AWS
- Astyanax: a Java client library for Cassandra that improves on an earlier client called Hector
- Honu: a high throughput streaming data logging system based on Apache Chukwa

Rik: In your presentations you mention two tools your developers use: Jenkins and Perforce. Could you tell us more about those tools? For example, why did you choose Perforce over an open source solution?

Adrian: Netflix has been using Perforce as its in-house source code control system for many years, and when we moved to the cloud we didn't change it. We use Jenkins to run our build system for the cloud. Carl Quinn presented on this at Devoxx [4]. Carl's team runs Perforce, Jenkins, Ivy, Artifactory, and related tools for our cloud developments.

Rik: You've made it clear that your move to AWS is as much about reliability as it is about flexibility. Yet even AWS can fail, as seen in the April 2011 partitioning event that occurred in Amazon's East Region datacenter [3]. That event was related to a specific network configuration that exacerbated the initial problem, a problem that occurred because the failover hadn't been tested under full load. What do you recommend that other organizations that plan on, or have moved to, cloud operations do to prepare for such events?

Adrian: We have published a Tech Blog that summarizes what we learned from that outage [5]. There is a lot of detail there that answers your question.

Rik: What's coming next?'

Adrian: We are currently working on the backend infrastructure to support our UK and Ireland launch, which leverages the AWS Europe region located in Ireland. Netflix has no employees in Ireland, and the flexibility this gives us in contrast to owning our own datacenters is extremely valuable. From a technical perspective, the Netflix Global Cloud Platform is being polished, hardened, and tuned to run more efficiently, and we have several additional components we are planning to open source during 2012.

Resources

[1] Slides for Cockcroft's presentations: http://www.slideshare.net/adrianco.

[2] Cockcroft's blog entry: http://techblog.Netflix.com/2011/11/benchmarking-cassandra-scalability-on.html.

[3] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region: https://aws.amazon.com/message/65648/.

[4] Carl Quinn at Devoxx: http://www.devoxx.com/display/DV11/Carl+Quinn.

[5] Netflix blog regarding lessons learned from AWS outage: http://techblog.netflix.com/2011/04/lessons-netflix-learned-from-aws-outage.html.

Practical Perl Tools

CSV and the Spreadsheet Go A-Wanderin'

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

I can't predict this for sure, but if I were a betting man I'd lay even money that at some point in your career you are going to be handed data in CSV or Microsoft Excel format and be asked to parse it. You may even be asked to produce data in one of those formats.

CSV, which can either mean "comma-separated" values or "character-separated" values, depending on whether you woke up on the pedantic side of the bed in the morning, is one of the more ubiquitous data formats on the planet. Similarly, there are people in the business world who treat Excel as their mother tongue, so being able to read and write spreadsheets in this format will definitely make you a hit among the suits. By the way, I am aware there are more open and liberty-leaning alternatives to Excel around. Much of what we'll talk about in this column will still be useful if those alternatives are your tools of choice, but I mostly won't be addressing them directly.

C D CSV

The canonical, most basic module for processing CSV in Perl is Text::CSV. If you are going to use this module and there are no pure-Perl restrictions in place, you will definitely want to also install the Text::CSV_XS module at the same time. Text::CSV_XS is a replacement backend for Text::CSV written in C that is much, much faster than the pure-Perl parsing routines found in Text::CSV. These two modules are perfectly intertwined: Text::CSV will automatically use Text::CSV_XS with no change of syntax if it notices it is available.

You might wonder why someone has gone to the trouble of writing an entire module to parse CSV data when it seems as though a few simple split() functions would do the trick. Take a moment to read the documentation for the module. Right around the time your eyes start to glaze over from reading all of the possible options, you'll probably come to the conclusion, "Hey, the CSV format isn't as simple as I thought. There are many different ways it could be implemented." And, indeed, I've talked to Perl programmers who have related horror stories about how different program/systems they had to mesh had slightly different interpretations of how CSV should be parsed/written. Using Text::CSV and its copious options can help to shield you from this unpleasantness.

Let's look at the basics of how Text::CSV is used, check out one advanced feature, and then look at a few other more sophisticated modules that use Text::CSV to do the dirty work. To use Text::CSV, we start with code that looks like this (slightly modified from the example in the doc):

```
use Text::CSV;
use Data::Dumper;

my $c = Text::CSV->new( { binary => 1 } )
       or die Text::CSV->error_diag();

open my $FILE, '<', $ARGV[0] or die "Can't open $ARGV[0]: $!\n";

# row will be an array ref pointing to an array of parsed fields
while ( my $row = $c->getline($FILE) ) {
       print Dumper($row);
}
$c->eof or $c->error_diag();
close $FILE;
```

Here's one section of output showing my local airport when I run it against the airports.csv file provided by OurAirports at http://www.ourairports.com/data/:

```
$VAR1 = [
          '3422',
          'KBOS',
          'large_airport',
          'General Edward Lawrence Logan International Airport',
          '42.36429977',
          '-71.00520325',
          '20',
          'NA',
          'US',
          'US-MA',
          'Boston',
          'yes',
          'KBOS',
          'BOS',
          'BOS',
          'http://www.massport.com/logan/',
          'http://en.wikipedia.org/wiki/Logan_International_Airport',
          'General Edward Lawrence Logan International Airport'
        ];
```

Let's walk through the Text::CSV code above so you can see how this all works. After loading the module (with Text::CSV_XS being loaded implicitly), we create a parser object using new(). There are quite a few options available but in this case I'm setting "binary," the one option that I think makes sense to include in every Text::CSV script you write (I'd be hard-pressed to think of a case where you wouldn't want it). Using the binary option will allow the parser to parse fields that contains non-ASCII characters without choking. You never know when José Feliciano or Björk Guðmundsdóttir might show up in your data, so it is safer to set that option.

The code then opens up a file handle using the preferred three-argument form of open(). From this file handle, we have Text::CSV read and parse each line using getline(). The getline() function reads a line, then parses it into fields, with each field occupying a position in an array. It returns a reference to this array, the contents of which we dump out in the loop. The getline() call will return "undef" if it

can't read any more data or if the parsing process crashes and burns. This explains the line after the loop that says:

```
$c->eof or $c->error_diag();
```

If getline() exited because it hit the end of file, everything is peachy (and the first part of that statement is true so it shortcuts to the next line). If not, we call a function that will print the failure information to STDERR. We close the file and go home, a job well done.

I dumped the contents of the array within the loop, but you can imagine doing all sorts of intricate and productive things by accessing $row->[$someposition]. If we want to get a little more sophisticated with Text:CSV we can use getline_hr() instead of getline() to return a hash reference along the lines of:

```
$VAR1 = {
          'iso_country' => 'US',
          'municipality' => 'Boston',
          'home_link' => 'http://www.massport.com/logan/',
          'local_code' => 'BOS',
          'keywords' => 'General Edward Lawrence Logan International
Airport',
          'iata_code' => 'BOS',
          'latitude_deg' => '42.36429977',
          'iso_region' => 'US-MA',
          'id' => '3422',
          'longitude_deg' => '-71.00520325',
          'name' => 'General Edward Lawrence Logan International Airport',
          'elevation_ft' => '20',
          'ident' => 'KBOS',
          'wikipedia_link' => 'http://en.wikipedia.org/wiki/Logan_
              International_Airport',
          'scheduled_service' => 'yes',
          'continent' => 'NA',
          'type' => 'large_airport',
          'gps_code' => 'KBOS'
        };
```

It is soooo much easier to read code that says:

```
$row->{'longitude_deg'};
```

instead of:

```
$row->[5];
```

In order for getline_hr() to work its magic, it has to know what each of the fields represents. To set this information, we have to call column_names() with an array reference to an array with the list of field names prior to calling getline_hr(). In our case, we were working with a .csv file that had a descriptive header row at the beginning of the file:

```
"id","ident","type","name","latitude_deg","longitude_deg","elevation_ft",
"continent","iso_country","iso_region","municipality","scheduled_service",
"gps_code","iata_code","local_code","home_link","wikipedia_link","keywords"
```

so we could include this right before our parsing loop changed to use getline_hr() instead of getline():

```
my $header = $c->getline($FILE);
$c->column_names($header);
```

or, if we wanted to be terser, just:

```
$c->column_names( $c->getline($FILE) );
```

If you decide neither an array ref nor a hash ref floats your boat, you can take a page from the DBI playbook (truth be told, I don't know which came first) and use bind_columns() instead. With bind_columns() your code looks like this:

```
$c->bind_columns (\$id, \$ident, \$type, ... \$keywords);
while ($csv->getline ($FILE)) { ... }
```

Each time through the loop, the data is parsed and then assigned to each of those scalars in order.

Earlier in this section I suggested that there might be more sophisticated modules for each task. In the case of Text::CSV there are quite a few modules that build upon it. Let's look at two that try to make it even simpler to use for the most common cases:

1. Text::CSV::Simple attempts to collapse down the code we saw before into a very simple set of lines:

```
use Text::CSV::Simple;
my $c = Text::CSV::Simple->new( { binary => 1 } );
my @rows = $c->read_file( $ARGV[0] );
```

At this point `@rows` is an array of arrays. Well, more precisely it is an array containing references to other arrays that represent the rows. The row arrays have one field per position. But all of this is easier if you think of it as an array of arrays, so that `$rows[0][2]` would refer to the third field in the first header row (the string "type").

If we didn't want to capture all of the fields in each row, we could instead use the want_fields() method to specify just the field numbers desired. Text::CSV::Simple can also do the equivalent trick with a hash reference if you use the field_map() method:

```
$c->field_map(
    "id",             "ident",
    "type",            "name",
    "latitude_deg",   "longitude_deg",
    "elevation_ft",   "continent",
    "iso_country",    "iso_region",
    "municipality",   "scheduled_service",
    "gps_code",       "iata_code",
    "local_code",     "home_link",
    "wikipedia_link", "keywords"
    );

my @rows = $c->read_file( $ARGV[0] );
```

Now `@row` contains hash references instead of array references, so you can say:

```
# we could leave out the arrow, but it is harder to read
$row[1]->{ident};
```

To perform the equivalent function of want_fields() in the last example, we could specify "null" in the field_map() statement, as in:

```
$c->field_map('id', null, null, 'name', ... );
```

and those fields will simply be ignored in each row.

2. If you liked Text::CSV::Simple's ability to grab data and place it into a data structure in one fell swoop, but didn't like that it changed data structures based on whether another method had been run before it, you might like Text::xSV::Slurp better. It gets used like this (to mimic functionality we've seen so far):

```
use Text::xSV::Slurp 'xsv_slurp';

open my $FILE, '<', $ARGV[0] or die "Can't open $ARGV[0]:$!\n";

my $aoa = xsv_slurp( $FILE, shape => 'aoa',
                            text_csv => { binary => 1 } );
```

The key magic here is the "shape" parameter. That can be one of these:

```
aoa - array of arrays
aoh - array of hashes
hoa - hash of arrays
hoh - hash of hashes
```

We've seen the first two before; the third allows you to essentially invert the data so that there is a hash that uses the name of the column (from the header information) as a key and a list of all of the values in that column for the value of that hash element. The "hoh" shape lets you pick arbitrary columns to use as keys in an (often multiply nested hash) data structure. It also allows you to provide code to handle "non-unique key combinations." For all of these shapes, the module gives you the opportunity to specify code for col_grep and row_grep parameters that will be used to select certain columns or rows for inclusion in the data structure returned by xsv_slurp().

Are you getting tired of CSV modules yet? Me too, so let me wrap up a few small details and we can move on. The first is that all of the modules we've been discussing have "un-parse" methods that will take a data structure of some sort (e.g., an array) and collapse it into a CSV row for writing. There's nothing very sophisticated in how they work, so I'll pass on providing an example. The second detail I'd be remiss if I didn't mention is that there are other alternatives to Text::CSV-based modules that are worth exploring. For example, Text::xSV makes it easier to cope with CSV data that contains newlines in a field (technically allowed, but usually a pain if you have to deal with it). And finally, there are several modules designed to just slurp CSV files right into databases (e.g., DBIx::TableLoader::CSV) that you should consider if that happens to be your use case.

And Now, the Spreadsheet: Reading

Let's visit our second data format. Despite any misgivings you might have about Microsoft and its business practices, and no matter how much you've tried to avoid sullying your hands by touching only non-proprietary formats, your delicate ocular nerves must have at one time or another lost their innocence and gazed upon an Excel spreadsheet. Or perhaps you think Excel is the coolest thing since Daedalus's

wings and you absolutely adore Excel. No matter where on the continuum you find yourself, at some point you may still be called upon to either read or write an Excel spreadsheet. We're now going to explore how this can be done from Perl.

One quick plot twist before we get there: Excel spreadsheets come in two basic flavors. Prior to Excel 2007, the file format was a relatively intricate binary format (Wikipedia tells me this was called "BIFF," for Binary Interchange File Format, so hey, learn a new thing every day). This was the format of the beloved .xls extension. After that version, Microsoft switched to an XML format (albeit compressed essentially into a zip file) that used an extension of .xlsx by default instead. Although it might be a bit informal or imprecise to refer to the file format by its extension (i.e., .xls-formatted), I'm going to do so because I think it is clearer.

The old .xls format is still opened seamlessly by the newer versions and most anything that processes Excel spreadsheets. The Perl module landscape has far more modules to deal with .xls files than it does with .xlsx files. You can assume that the modules we'll be talking about deal primarily with .xls files unless I mention otherwise.

Just as Text::CSV was a core CSV-parsing module, I think it is safe to say that Spreadsheet::ParseExcel can be considered the equivalent for Excel spreadsheets. The example from the documentation does an excellent job of demonstrating the basic operating principles of the module:

```
use Spreadsheet::ParseExcel;

my $parser = Spreadsheet::ParseExcel->new();
my $workbook = $parser->parse('Book1.xls');

if ( !defined $workbook ) {
    die $parser->error(), ".\n";
}

for my $worksheet ( $workbook->worksheets() ) {

    my ( $row_min, $row_max ) = $worksheet->row_range();
    my ( $col_min, $col_max ) = $worksheet->col_range();

    for my $row ( $row_min .. $row_max ) {
        for my $col ( $col_min .. $col_max ) {

            my $cell = $worksheet->get_cell( $row, $col );
            next unless $cell;

            print "Row, Col = ($row, $col)\n";
            print "Value = ", $cell->value(), "\n";
            print "Unformatted = ", $cell->unformatted(), "\n";
            print "\n";
        }
    }
}
```

Basically, we create a new parse object and point it at an .xls-formatted file. In return, we get a workbook object (if not, we bail). From that workbook we can further home in on a specific worksheet. Using the worksheet object, its methods allow us to determine the row and column ranges present in that worksheet. We then iterate over the rows and columns, retrieving a specific cell as we move

through it. For each cell object, we can retrieve its value and also examine how that cell is formatted. This workbook->worksheet->cell sort of approach to dealing with spreadsheets is also present in Spreadsheet::XLSX, the .xlsx equivalent to Spreadsheet::ParseExcel. The documentation says, "It populates the classes from Spreadsheet::ParseExcel for interoperability; including Workbook, Worksheet, and Cell," but its example code shows it can also support a slightly different syntax.

Based on our experience with CSV-parsing modules, you can probably guess that there exist other modules slightly higher up in the food chain designed to make working with these parsers easier. The first of them has the entirely predictable name of Spreadsheet::ParseExcel::Simple. Working with the ::Simple version consists of the following model, according to the documentation: "You simply loop over the sheets, and fetch rows to arrays." Like so (again, from the docs):

```
use Spreadsheet::ParseExcel::Simple;
my $xls = Spreadsheet::ParseExcel::Simple->read('spreadsheet.xls');
foreach my $sheet ( $xls->sheets ) {
    while ( $sheet->has_data ) {
        my @data = $sheet->next_row;
    }
}
```

A second module that attempts to provide a simpler interface deserves mention, not for the module itself but for a sample script that ships with it that has tremendous utility. Spreadsheet::BasicRead comes with xslgrep.pl, a script that will search all contents of all of the cells in the .xls-formatted spreadsheets in a directory hierarchy for a specified regular expression—tremendously helpful if you know you have the information embedded in a spreadsheet someplace but can't recall which file it is in.

Spreadsheet::ParseExcel::Stream is a module that may come in handy if you are parsing very large spreadsheets. By default, Spreadsheet::ParseExcel will suck a spreadsheet into memory as part of its parsing process. There's a whole section in its doc, called "Reducing the memory usage of Spreadsheet::ParseExcel," which describes a slightly more complicated way of using Spreadsheet::ParseExcel that does not retain this behavior and its drawbacks. Spreadsheet::ParseExcel::Stream implements that recommendation and provides a simpler interface to boot.

And, finally, one last Excel-reading helper module to end this subsection: Spreadsheet::Read. Spreadsheet::Read attempts to be one interface to rule them all. If you point it at an .xls-formatted file, it will call Spreadsheet::ParseExcel, .xlsx: Spreadsheet::XLSX, .csv: Text::CSV_XS, and, yes, open source fans, .ods (OpenOffice format) files will be parsed by a module we haven't discussed, Spreadsheet::ReadSXC. See the documentation for various fiddly options that can be set. If you are a big fan of "single interface" modules, this module may please you.

And Now, the Spreadsheet: Writing

In my experience, it is far more common to be asked to parse Excel spreadsheets created in Excel than it is to be asked to actually produce those documents. Still, that need also arises on occasion. For those requests, there is a similar wolf pack of modules. Unlike our previous problem domains, this is one place where there isn't a clear base module that can be considered the one true central module everything else uses. As far as I can tell, there are two: Spreadsheet::Write (and its fork,

Spreadsheet::Wright—more on that in a minute) and Spreadsheet::WriteExcel. Both of these are used for writing .xls-formatted files. The choice for .xlsx files is a little clearer: Excel::Writer::XLSX.

Here's how you can choose between Spreadsheet::Write/Spreadsheet::Wright and Spreadsheet::WriteExcel. If you care about being able to write not only .xls files but also OpenDocument (i.e., OpenOffice/LibreOffice) files, Spreadsheet::Wright will be your best bet. If you find that having example code will be useful to your development process, you will want to choose Spreadsheet::WriteExcel because it ships with 80+ samples (Excel::Writer::XLSX, by the same author, ships with a measly 64 example scripts).

My inclination is to use the latter module, because I appreciate distributions that have that superior level of documentation, especially when I am pressed for time. Also, Spreadsheet::WriteExcel plays nice with Spreadsheet::ParseExcel (same author—John McNamara clearly rocks). In the Spreadsheet::ParseExcel distribution there is a Spreadsheet::ParseExcel::SaveParser module that allows you to "rewrite an existing Excel file by reading it with Spreadsheet::ParseExcel and rewriting it with Spreadsheet::WriteExcel."

Let's see a teeny Spreadsheet::WriteExcel example from its doc so you can get a quick sense of how one goes about creating an Excel spreadsheet from Perl:

```
use Spreadsheet::WriteExcel;

# Create a new Excel workbook
my $workbook = Spreadsheet::WriteExcel->new('perl.xls');

# Add a worksheet
$worksheet = $workbook->add_worksheet();

# Add and define a format
$format = $workbook->add_format();    # Add a format
$format->set_bold();
$format->set_color('red');
$format->set_align('center');

# Write a formatted and unformatted string, row and column notation.
$col = $row = 0;
$worksheet->write( $row, $col, 'Hi Excel!', $format );
$worksheet->write( 1, $col, 'Hi Excel!' );

# Write a number and a formula using A1 notation
$worksheet->write( 'A3', 1.2345 );
$worksheet->write( 'A4', '=SIN(PI()/4)' );
```

It is basically taking the process we saw in parsing an Excel file and throwing it into reverse. Create a workbook, add a worksheet to it, and then create cells with certain values and formats.

Although we are basically out of time, I will mention that there are a few modules, such as Spreadsheet::SimpleExcel, that attempt to make creating simple Excel spreadsheets easier. There are also single-task modules such as Spreadsheet::WriteExcel::FromDB for converting database tables into spreadsheets. I encourage you to try out any of these Excel creation modules, because being able to create spreadsheets on the fly from Perl is a neat trick that is sure to impress your coworkers. Take care, and I'll see you next time.

iVoyeur

Changing the Game, Part 2

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

Near the end of his poem "The Talking Oak," Tennyson alludes to the oldest of the pagan oracles: Jupiter at Dodona. It was quite different from the oracles that followed it in that no temple, altar, or human contrivance was ever constructed there. It was merely an oak grove on an island in the Aegean Sea. The Selli tribal priests who lived there could decipher the word of Jupiter himself from the sound of the wind rustling the leaves of those sacred oak trees (some stories say wind-chimes were also employed).

I'd read Tennyson's poem in high school but, that being pre-Google, I never understood his reference to "that Thessalian growth" until I recently happened to read about the oracle at Dodona. The resolution of that long-forgotten enigma must have made an impression on my subconscious, because I subsequently dreamt that I visited that ancient oracular forest and heard the whisper of its long-dead deity. His message to me? "Your Web server is down."

I often tell people, when the subject of my occupation arises, that I'm a plumber. This saves me from having to hear about their brother-in-law "the computer guy" with whom I must have so much in common, but really I say this because I often feel like plumbing is what I do for a living. My chimerical jaunt to Dodona, however, has me wondering whether what I do for a living, especially in the context of systems monitoring, has more in common with divination than craftsmanship.

How often, after all, do the systems just tell us what's wrong with them? My troubleshooting technique is invariably a murky blend of experience, data analysis, intuition, and luck. Things will strike me as "wrong," sometimes without my being able to articulate why, and I'll eventually arrive at root cause by following up on the "wrongness." Maybe this works because the systems and their components are both more dependent on each other than we realize and related to each other in ways we don't expect. We can stumble across some wrongness that eventually leads to an answer because everything is ultimately tied together. A deep understanding of the relationship between our systems is what allows us to hear the trouble in the wind, and we're fascinated by the interplay between systems as a result.

Event correlation work like [1] and [2] are related endeavors, but what I'm really thinking about is work like that presented by Adam J. Oliner [3] from Stanford at LISA '10 wherein they attempt to identify and quantify the extent of "influence" between components of complex systems. Once identified and quantified, cause

and effect can be inferred. If you've seen the presentation, you know what I'm talking about; that isn't plumbing, it's divination.

A less mathematically rigorous, more organic version of the same thing goes on between a sysadmin and his graphs. Influence is being explored, cause and effect inferred, answers divined. This, I think, is why we get so excited about new ways to visualize our data, especially if we're provided some means to include data that we weren't visualizing before. Properly understood, every new data point provides an opportunity to understand more deeply the interoperability of a complex system.

In my last article I introduced Graphite, a data storage and visualization system that does an amazing job of giving us the means to visualize data we wouldn't have considered previously, either because it was outside the domain of Ops (e.g., sales data), or because it didn't fit RRDTool's storage paradigm (e.g., temporally asynchronous or inconsistent data). In this article I'd like to explore some of the ways we might integrate Graphite with our existing tools.

Nagios Integration

You'll recall that Graphite's Carbon daemon listens on port 2003 for a string with the form "name value date" and automatically creates a whisker database within which to store the data. It's trivial to push data from any sort of classical monitoring system into Graphite using netcat as a client. Nagios, for example, will export "performance data" for host or service checks. Performance data is officially defined as anything that follows a pipe character ("|") in the output of a check result, but the performance data command can easily be modified to include the entire text of the check result.

For example, my Nagios "process_performance_data" command looks like this:

```
command_line    /usr/bin/printf "%b\n" \
      "$LASTSERVICECHECK$::$HOSTNAME$::$HOSTNOTES$::$SERVICEDESC$::\
      $SERVICEOUTPUT$::$SERVICEPERFDATA$" \
      >> /var/log/nagios/service-perfdata.out
```

This captures the entire output of the check result (including any performance data) and logs it to /var/log/nagios/service-perfdata.out. I then use logsurfer [4] to parse data out of the log and ship it to a Graphite sender. Here's the logsurfer definition to parse the output of the Nagios check_ping command:

```
'^([^:]+)::([^:]+)::([^:]+)::([^:]+)::PING [A-Z]+ - Packet loss = ([0-9]+)%, RTA
= ([^ ]+) ms::.*$' - - - 0 continue
      exec "/usr/local/bin/gclient.sh $4.$3.$5-loss $6 $2 graphitebox 2003"
```

gclient.sh is a simple shell script that uses netcat to push data to Graphite like so:

```
#!/bin/sh
#send data to carbon
SERVER=$4
PORT=$5
[ "${SERVER}" ] || SERVER=graphitebox
[ "${PORT}" ] || PORT=2003
echo "$1 $2 $3" | /usr/bin/nc -c $SERVER $PORT
```

Ganglia Integration

When I went to look for Graphite/Ganglia [5] integration I came up a bit bare. Current versions of Ganglia support using Graphite instead of RRDTool as a rendering and storage engine for GWeb, but this isn't really what I was looking for. In my mind, Ganglia fits the bill for Ops guys looking for an easy way to get metrics across all of their hosts, and Graphite is a more corporate-wide, general-purpose data visualization solution. I wanted an easy way to export my Ganglia metrics on the fly to Graphite while still keeping the former in its native format. The options were not what I'd consider viable [6].

It seemed to me that the shortest path would be to modify gmetad with an option to export the metrics directly to Graphite as it writes them to RRDTool locally, so I took a look at the gmetad source and submitted some patches to the mailing list a few hours later [7]. These still need work, but they're functional, and have been merged into Ganglia monitor-core [8], so you can get them by checking monitor-core out from git, or taking the patches from the mailing list in the usual way. I've added the following four configuration options to gmetad.conf, two of them being optional:

1. carbon_server should be set to the remote hostname or IP address of the graphite server.
2. graphite_prefix should be whatever you want to prefix your graphite path with (ganglia.<gridname> or something like that).
3. You can optionally specify a "carbon_port." This defaults to 2003 if you don't specify it.
4. You can optionally specify a "carbon_timeout" to timeout connection attempts if/when the Graphite server is down. This defaults to 500 ms if you don't specify it.

Statsd

Graphite makes it feasible for developers to instrument their code to send metrics to Graphite relating to the inner workings of their applications. If the foo() function gets called every time someone makes a $5 purchase on a Web site, it might be interesting to maintain a counter of the number of times foo() gets called. If bar() might cause performance problems, it might be interesting to keep a gauge for how long bar() takes to execute. A problem with these scenarios is what happens when the application gets distributed to hundreds of servers. Suddenly the foo() counters need to somehow be aggregated and the bar() gauges need to be averaged. The people at Etsy [9] wrote a very popular NodeJS-based metrics aggregation daemon called StatsD [10] to deal with this problem, and Jeff Buchbinder ported it to C [11].

StatsD libraries now exist for most popular programming languages (Perl, PHP, Python, Java, Ruby, Lua, etc.). These make it easy for developers to create and maintain counters and gauges (called "timers" in StatsD) in their application. The metrics are sent to the StatsD server, where they are aggregated on normal intervals and sent on to Graphite. StatsD has the additional advantage of using UDP, so the application servers can "fire and forget."

StatsD and developer interaction makes it possible to collect some interesting business metrics. Questions like "How many users did we register?" or "How many SKU4242s did we sell?" can now be easily visualized on the same graph with

system metrics (e.g., network utilization) or other dev metrics (e.g., release cycles) imported from integration systems like Hudson.

Logster

The guys at Etsy also created a dedicated log-parser for Graphite called logster [12]. Logster is a forked and simplified version of ganglia-logtailer which uses logcheck [13] along with external definitions for parsing metrics out of log files and sending them to Carbon. It comes with parsers for the Apache Web server and is intended to be run every minute from cron.

Collectd Integration

Joe Miller wrote a Graphite plugin [14] for collectd [15] that bears mentioning. I haven't personally used collectd, so I can't really provide any details, but it's there.

Reverse Integration

I'm referring here to our ability to take graphs from Graphite and re-purpose them back into your monitoring tools. Once your metrics are in Graphite, the easiest way to get graphs back out is the excellent URL interface. Every feature and function available in the Graphite CLI or Web interface is exposed as a CGI attribute in the URL interface, making it possible to graph any combination of metrics, apply functions like "average" or "derive," and control the look and feel aspects of the graph such as image size, fonts, colors, etc., all with URL parameters.

These graphs can be referenced by any external dashboard or monitoring system that will take a URL. For example, I use the "action_url" attribute in the Nagios service description for this purpose. Taking our ping example from above, we can reverse-integrate our ping data back into the Nagios UI by adding an action_url attribute in the ping service description that looks like this:

```
http://graphitebox/render?from=-6h&target=dc2.linux.$HOSTNAME$.PING
-rta&width=1024&height=768&hideGrid=true
```

The $HOSTNAME$ parameter is a Nagios macro that will be replaced at runtime with the hostname of the host to which it refers. The rest should be self-explanatory.

I doubt I'll begin introducing myself as an oracle anytime soon. Not because it'll make me sound like a crazy person (I'm sure I sound like a crazy person anyway) but, rather, because if I think too hard about the metaphor I find it a bit depressing. I realize the Selli priests had it way better than we do, because their oracles would sometimes deliver them a propitious message. Ours, on the other hand, rarely have anything but bad news.

Take it easy.

References

[1] Paul Krizak, "Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)": http://www.usenix.org/events/lisa10/tech/full_papers/Krizak.pdf.

[2] Ariel Rabkin and Randy Katz, "Chukwa: A System for Reliable Large-Scale Log Collection": http://www.usenix.org/events/lisa10/tech/full_papers/Rabkin.pdf.

[3] Adam Oliner and Alex Aiken, "Using Influence to Understand Complex Systems": http://www.usenix.org/multimedia/lisa10oliner/.

[4] Logsurfer: http://www.crypt.gen.nz/logsurfer/.

[5] Ganglia: http://ganglia.sourceforge.net/.

[6] Vladimir Vuksan, "Integrating Ganglia with Graphite" (blog post): http://blog.vuksan.com/2010/09/29/integrating-graphite-with-ganglia/.

[7] Dave Josephsen, Graphite support for gmetad (mailing list thread): http://www.mail-archive.com/ganglia-general@lists.sourceforge.net/msg06964.html.

[8] Github Ganglia Monitor Core: https://github.com/ganglia/monitor-core/pull/1.

[9] Etsy: http://www.etsy.com/.

[10] StatsD: https://github.com/etsy/statsd.

[11] StatsD-c: https://github.com/jbuchbinder/statsd-c.

[12] Logster: https://github.com/etsy/logster#readme.

[13] Logcheck: http://logcheck.org/.

[14] Collectd-graphite: http://joemiller.me/2011/04/14/collectd-graphite-plugin/.

[15] Collectd: http://collectd.org/.

Three Years of Python 3

DAVID BEAZLEY

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

First, I'd like to offer a general welcome. This is the first of what I hope will be an ongoing series of articles about all things Python. This year marks the 21st anniversary of Python's first release. Although Python has been around for some time, its popularity has been growing by leaps and bounds—especially if one looks at the soaring attendance at Python conferences worldwide.

I first discovered Python in 1996 when I was still a graduate student. Since then, it's become my tool of choice for, well, just about everything (although I still do a fair bit of C programming for occasional projects involving embedded systems or high performance computing).

In this column I hope to explore a variety of topics related to modern Python software development. This includes advances in the language itself, interesting new add-on libraries, useful programming techniques, and more. No topic is off-limits, although I'll admit that I do have a certain preference for problems involving data analysis and systems programming. In this issue I start by sharing some of my experiences working with Python 3. In future issues, I hope to look at a variety of other topics, including the PyPy project, the state of Python concurrency, and hidden secrets of the standard library.

Python 3

It's hard to believe, but last December marked the three-year anniversary of the first release of Python 3. Judging by the glacial rate of adoption, you might hardly notice. Honestly, most Python programmers (including most of Python's core developers) are still using various versions of Python 2 for "real work." Nevertheless, Python 3 development continues to move along as the language is improved and more and more libraries start to support it. Although you might look at the adoption rate with some dismay, it has always been known by Python insiders that it could take five years or more for a significant number of users to make the switch. Thus, we're still in the middle of that transition.

A few years ago, I presented a tour of Python 3 (see *;login:* April 2009). At that time, it wasn't all rosy. I noted some major problems, such as horrible I/O performance and complications in the bytes/Unicode interface. For the most part, Python 3 was simply a curiosity and something far too experimental to rely on for "real" work. Since then, a lot of problems have been addressed and the implementation improved. However, if you've been sitting on the sidelines, it's still hard to know exactly what Python 3 offers.

About a year ago, I made a conscious decision to write all new projects in Python 3 to get a better sense of working with it in practice and to explore issues involved in porting existing code. So, in this article, I hope to revisit the topic of Python 3, but with a slightly different spin. Rather than simply rehashing a long list of new language features, I thought I would simply focus on the specific parts of Python 3 that have, through experience, actually proven to be rather useful, surprising, or problematic.

Useful Features

There are certain language features unique to Python 3 that I now find myself using with some regularity. Few of these features have been backported, and they are unlikely to appear in any future version of Python 2.

Sequence Unpacking Wildcards

If you have a Python tuple (or other sequence), you can easily unpack it into variables. For example:

```
>>> row = ('ACME',50,91.15)
>>> name, shares, price = row
>>>
```

In Python 2, such unpacking only works if the number of items in the sequence on the right exactly matches the number of storage locations on the left. However, in Python 3, you can introduce a wildcard that will match any number of items and collect them into a list. For example:

```
>>> name, *last = row
>>> last
[50, 91.15]
>>> *first, price = row
>>> first
['ACME', 50]
>>>
```

Although this feature might seem minor, unpacking sequence data comes up quite a bit in the context of working with tabular data: for example, reading data out of databases, reading lines from CSV files, and so forth. Using the wildcard can be a convenient means to write code that only wants to work with some of the fields or with data that has a varying number of columns.

I have also found wildcard unpacking to be a useful technique for treating Python tuples as a kind of prefixed data structure akin to a Lisp S-expression. Here the first element might be some kind of operator, tag, or identifier, while the remaining elements are data. Thus you can write code that processes the data like this:

```
>>> s = ('+',3,4,5)
>>> op, *data = s
>>> op
'+'
>>> data
[3, 4, 5]
>>>
```

In Python 2 you could achieve the same effect by writing `op, data = s[0], s[1:]`. However, the new version is just a bit more elegant, so why not use it?

Keyword-only Function Arguments

Consider the following Python function:

```
def recv(block=True, timeout=None):
        # Receive a message
        ...
```

One issue with using the above function is that subsequent calls can potentially lead to cryptic code where the meaning of the arguments is not entirely clear to someone reading it. For example:

```
msg = recv(False)  # What does False mean?
msg = recv(1,5)  # 1? 5?
Huh? ...
```

To avoid this, you can force keyword arguments by inserting a * into the argument signature:

```
def recv(*,block=True, timeout=None):
        # Receive a message
        ...
```

For all arguments after the *, users are forced to use keywords when calling:

```
msg = recv(block=False)  # Ok
msg = recv(block=1,timeout=5)  # Ok
msg = recv(1,5)  # Error
```

Although you could achieve a similar effect in Python 2, it was always rather clumsy. For example, this old code implements the same functionality, but with much less elegance:

```
dec recv(**kwargs):
        block = kwargs.pop('block',True)
        timeout = kwargs.pop('timeout',None)
        if kwargs:
           raise TypeError("Unknown argument(s): %s" % list(kwargs))
        ...
```

One of the reasons I like this feature is that it allows you to write functions that have more precise calling signature and less underlying magic related to fiddling around with various forms of `*args` and `**kwargs`. It even plays nice with documentation and IDEs. For instance, if you ask for `help(recv)`, you'll get more descriptive output showing the argument names as opposed to a vague description of `**kwargs`. Believe it or not, this is one of my most-used Python 3–specific language features. It's nice.

Dictionaries, Sets, and Views

One feature of Python 3 that has taken some time to fully appreciate is the better unification of dictionaries, sets, and the newly introduced dictionary "view" objects. Under the covers, sets and dictionaries are implemented in an almost identical manner. Essentially, a set is just a dictionary, but with keys only.

In Python 2, sets always felt kind of bolted on to the rest of the language (somewhat true, as sets weren't actually introduced until Python 2.3). In Python 3, they are much more integrated with all of the other data types. First, there is new syntax for writing a set:

```
>>> fruits = { 'pear', 'apple', 'banana', 'peach' }
>>>
```

Dictionaries now support a different mechanism for working with the keys, values, or key/value pairs. Consider a simple dictionary:

```
>>> a = {
        'x' : 1,
        'y' : 2,
        'z' : 3
      }
>>>
```

If you ask for the dictionary keys, Python 3 gives you a "key-view" object. What's unusual about a view is that it's not a distinct container. Instead, it gives you a window on the current set of keys defined on the associated dictionary. If the dictionary changes, so does the view. For example:

```
>>> k = a.keys()
>>> k
dict_keys(['y', 'x', 'z'])
>>> a['t'] = 4
>>> k
dict_keys(['y', 'x', 'z', 't'])# Notice the change
>>>
```

Key-view objects also support a core group of set operations, including unions, intersections, and differences. This means that you can start to mix dictionary data with sets and easily perform more complex operations (e.g., identifying common keys, finding missing values, etc.). Here are some examples of such operations:

```
>>> b = { 'w' : 10,
        'x' : 20,
        'y' : 30 }
>>> a.keys() & b # Find keys in common
{'x', 'y'}
>>> a.keys() - b # Find keys in 'a', not in 'b'
{'t', 'z'}
>>> assert a.keys() == {'x','y','z','t'}
True
>>>
```

Such operations are actually highly relaxed in their type checking. In fact, they work if the other operand is any kind of sequence.

```
>>> a.keys() - ['x','y']
{'t', 'z'}
>>> a.keys() - "xy"  # "xy" is a sequence of chars 'x', 'y'
{'t', 'z'}
>>>
```

Tied into this whole picture are set and dictionary comprehensions. These mimic similar functionality found in list comprehensions. For example, here is a set comprehension:

```
>>> fruits = { 'pear', 'apple', 'banana', 'peach' }
>>> { f.upper() for f in fruits }
{'PEAR', 'BANANA', 'PEACH', 'APPLE'}
>>>
```

The above code runs about 30–40% faster than equivalent versions using list comprehensions such as `set([f.upper() for f in fruits])`. This is mainly due to not having to build the intermediate list first.

Dictionary comprehensions can be used similarly to construct new dictionaries. Here is an example that creates a dictionary consisting of keys not found in another dictionary:

```
>>> {key:a[key] for key in a.keys() - b }
{'z': 3, 't': 4}
>>>
```

I should note that general use of dictionary comprehensions seems to be a little tricky. Coming up with good use cases seems to be something that requires a bit more thought.

The New super()

For object-oriented programming, Python 3 features a new shorthand super() function that takes no arguments. For example:

```
class A:
       def bar(self):
          ...

class B(A):
       def bar(self):
          r = super().bar() # Call bar in parents
```

In past versions, you had to type `super(B,self).bar()`. Although this might seem like a minor feature, I find myself using it a lot—mainly taking advantage of the reduced typing.

Surprising Features

Certain subtle features have caught me by surprise. In most cases, I didn't discover these until I started porting libraries.

Changes to Exception Handling

Python 3 makes numerous changes to how exceptions get handled. Some changes are more subtle than others. For instance, the scope of exception variables has changed so that such variables do not exist beyond the associated `except` block:

```
>>> try:
      int("N/A")
    except ValueError as e:
      print("Error!")
```

```
Error!

>>> e  # Exists in Python 2, not in Python 3

Traceback (most recent call last):

      File "<stdin>", line 1, in <module>

NameError: name 'e' is not defined

>>>
```

Similarly, exceptions can no longer be indexed as tuples:

```
>>> try:
      int("N/A")
    except ValueError as e:
      print(e[0]) # Works in Python 2, not in Python 3

Traceback (most recent call last):
      File "<stdin>", line 2, in <module>
ValueError: invalid literal for int() with base 10: 'N/A'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
      File "<stdin>", line 4, in <module>
TypeError: 'ValueError' object is not subscriptable
>>>
```

Both of these changes have the potential to break old code in a very subtle way—especially in unit tests involving exception handling.

The error message in the last example demonstrates yet another new feature of exception handling: chained exception messages. If an exception occurs while handling another exception, you will get a traceback that includes information about both exceptions (in this case, the original `ValueError` exception and the `TypeError` that occurred while trying to subscript the exception value). This is actually a welcome feature that should help people debug exceptions in complicated applications and libraries.

Bizarre Scoping Behavior of exec()

In Python 2, the exec() function executes a string of code as if it had been typed in place. For example, you could write the following code and it will produce exactly the output you expect:

```
def foo():
      y = 10
      exec("x = y + 42")
      print(x)  # Outputs 52 (in Python 2)
        # NameError exception in Python 3
```

In Python 3, exec() no longer executes in quite the same way—in fact, you'll get an exception if you try the above example. This is because it now executes the associated code in a dictionary that is a `copy` of the actual local variables (the result of the locals() function). If changes are made to any of the values in this dictionary, they are simply discarded instead of being written back to the original local vari-

able. Thus, to execute the above code, you actually have to manage the variables yourself, as shown here:

```
def foo():
        y = 10
        lvars = locals()
        exec("x = y + 42",globals(),lvars)
        x = lvars['x'] # Get the changed value of x
        print(x)
```

Admittedly, use of exec() doesn't come up that often in most code, but if you do use it, you'll need to be aware that it doesn't work in quite the same way.

Subtle Differences Between Bytes and Strings

In Python 3, all strings are Unicode by default. However, there is a byte-string object for use with binary data. You might think that the Python 3 byte string is the same as the Python 2 string (str) object. This would be wrong. It actually behaves in some surprising ways, as shown below:

```
>>> s = b"Hello World"
>>> s[0] == b'H'
False
>>> s[0]
72
>>> s[:1]
b'H'
>>> t = bytes(13)
>>> t
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
>>>
```

In this example, you'll find that byte strings return integers when indexed but return new byte strings when sliced. In addition, converting values to bytes might not do what you expect (e.g., supplying an integer value to `bytes` makes a string of that size filled with zeroes).

Changes to bytes/Unicode handling have been, by far, the biggest headache in porting existing libraries to Python 3, especially libraries related to any kind of network programming or I/O. Not only do you have to make sure you're dealing with Unicode correctly, but you also need to account for the changed semantics of bytes to make sure your code works correctly.

The State of Third-Party Libraries

Perhaps the most major issue standing in the way of Python 3 adoption has been its lack of support for third-party libraries. A lot of Python programmers rely on packages such as numpy, Twisted, matplotlib, not to mention the plethora of Web frameworks. In this department, Python 3 is still a bit of a mixed bag and probably not for everyone.

Coming from the sciences, I have an interest in data analysis packages. For this, you're starting to see a lot more Python 3 support. For example, the popular numpy extension has supported Python 3 for at least the past year, and a recent coding

sprint has been working to produce a Python 3–compatible version of matplotlib (https://github.com/matplotlib/matplotlib-py3).

For many modules, you can find experimental Python 3 support hidden away in a project fork or patch set. For example, if you're browsing around a project on a site such as GitHub, go look at the different project forks and pull requests. Sometimes you'll find an experimental Python 3 version just sitting there.

The one big caution with third-party packages is that much of the ported code is unproven or experimental. At this time, there just isn't a critical mass of Python 3 programmers to really iron out bugs. Thus you might find that you have to fix a lot of things on your own. For this, it helps to be comfortable with Python 3 itself (especially its I/O handling), makefiles, setup.py files, Python code, and even C programming. Frankly, it's probably best to keep your expectations low so that you aren't disappointed when something doesn't work as expected.

On the subject of low expectations, Web developers should probably avoid Python 3 for now. None of the large Web frameworks seems to support it or even provides a timeline of when Python 3 support might be added. If you're using a small HTTP framework or library, you might have more luck. For example, you can find Python 3 support in CherryPy (http://www.cherrypy.org/).

Final Words

Three years ago I recommended that it might be best to sit back and watch Python 3 development for a bit to see what happens. Today, my recommendation is not much different. If you're working with a lot of existing Python code involving various library dependencies, working in Python 3 certainly won't be easy. On the other hand, if you're starting something new or don't mind tinkering, you'll find a lot of neat stuff waiting for you. As for Python 3's future, the next few years should be interesting to watch as Python's grand experiment continues to unfold.

If you're interested in working in Python 3, there are a few books and resources of interest—for example, Lennart Regebro's *Porting to Python 3* (http://regebro.wordpress.com/porting-to-python-3) and the official "What's New in Python 3" documentation (http://docs.python.org/dev/whatsnew/). I have also given some PyCon tutorials on Python 3 I/O handling (http://www.dabeaz.com/python3io/) that may be useful for understanding some important issues that will arise in porting.

Finally, it's probably worth noting that much of the excitement in the Python world is currently focused on PyPy (http://pypy.org). PyPy is an implementation of Python 2.7 that features a just-in-time compiler and offers a substantial performance boost over CPython for many kinds of problems. Will PyPy be the future of Python? Only time will tell. However, I hope to take a closer look at PyPy in a future issue.

/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

I think it's high time I gave my loyal reader(s) a tantalizing peek at the little cobweb-draped corner of *;login:* in which I lurk. Approved eye protection is advised. Three months before every issue hits the mail room/Web site, Rik sends out to his columnists the overall theme and expected content of that upcoming issue, in the overly optimistic hope that we will tailor our columns accordingly. Every author has her or his own way of dealing with deadlines; of coming up with devastatingly clever and technically on-target copy; in short, of doing that voodoo that we do so... um...well. Sharing my own process with you, rather than, say, writing an actual column, might enable me to discharge this issue's obligation without thinking too much: a welcome break from my usual routine. Oh, who am I kidding? "Without thinking too much" nails my modus operandi with uncanny accuracy.

Proceeding apace, let us examine the soft underbelly of yon columnist. Lint there is and in abundance, but we shall hold our breath and step over that damp and unsightly goo, for what we really hope to discover here are the arcane mental gymnastics that somehow lead to words being put to paper. Usually the theme or principal topic Rik gives us (us as in we columnists, not you the reader and I, unless you're another columnist, in which case you may as well stop reading this now and play Angry Birds) to work from is a term I've either never heard of or heretofore vaguely believed to be connected with a rare tropical skin condition.

This leads to a flurry of Internet-facilitated self-education, at the conclusion of which I have decided that I can't possibly learn enough about whatever it is to squeeze out a thousand-word column in time. (Not that cutting and pasting from Web sites hasn't crossed my mind: a lot of what passes for factual material in cyberspace is pretty funny in and of itself. But the next link up in the plagiarism chain might get all cross.) So I fall back on a literary technique that has served me well for over 20 years: making stuff up.

I occasionally consider writing an actual technical article, rather than this travesty, but I did a fair amount of technical authoring eight or ten years ago and managed hopelessly to confuse an entire generation of potential IT professionals who now have little recourse but to appear on reality shows or attend law school. I'm not too clear on the details, but I believe as a result several nations have passed statutes limiting the distribution of technically oriented publications with my name anywhere in the table of contents. Fortunately, these same countries have mysteriously reverted to an abacus-based computational infrastructure and thus have no real use for *;login:* except, perhaps, as kindling. No harm, no foul.

There is most likely not an old Norse aphorism that translates roughly (a little sandpaper might well do wonders here) to "word fame is word fame," apropos of the proposition that there is no such thing as bad publicity. I have friends who are inordinately fond of repeating this. Sometimes they repeat it so often I have to leave the room and get chicken enchiladas mole from a nearby (or, if they're really getting on my nerves, not so nearby) Mexican restaurant. You can't have chicken enchiladas mole without a margarita, and you can't have just one of those. At this juncture the day is effectively shot and there is nothing for it but to go home and crawl into bed.

Let's just presume, for the sake of getting on with it, that I have somehow achieved a meager understanding of the esoteric topic du jour. The next step in my column-generation protocol is doing research to fill in the gaps in my knowledge, some of which are so large they can be seen from the International Space Station. In the olden days this involved heading down to the library, cajoling my way past the staff who remember what happened the last time I was there, and spending an afternoon locating and reading books that have nothing whatever to do with the subject at hand. Now, thanks to our society's maniacal obsession with providing access to Wikipedia to every last person on the planet (except those with abacuses...abaci? abacera? whatever), I can just open up a half-dozen browser windows and let the irrelevant, unverified information flow over me like bad movie dialogue. I often lose track of both time and purpose during these forays, a condition I call "cyber-spatial disorientation."

It usually requires about three hours (six, if I have the Xbox fired up as well) of clicking on links and reading about Android exploits, dark matter, Islay scotch, Mayan ruins, unexplained phenomena, and video game previews before I've collected enough background information to feel well prepared to write. At this point I take a nap.

Awake and refreshed, I wander back over and sit in front of the keyboard for a while, trying to remember what I was doing earlier. Then it's time to invoke the browser windows and hit the Interwebs again, avoiding anything that contains the words "Republican," "Democrat," "Deficit," "Congress," "Bailout," "Sanctions," "Occupy," or "Licorice" (I hate licorice). Most of the remaining Web sites are devoted to dead celebrities, but I've forgotten what the topic was anyway, so this has almost no impact on operations.

Along about now the cat will saunter in and rub against my legs, pretending to be affectionate in an attempt to con me into getting up and making the long trek into the kitchen to produce another sacrificial offering of the same indeterminate slurry of grain and meat by-products she haughtily refused to ingest yesterday. When I fail to take the demanded action, she leaves in a feline huff, having during her brief tenure deposited fine, dander-laden hairs in each and every interstitial space on my keyboard.

By this point the deadline is looming. By looming, I mean tracking it no longer requires a calendar, but, rather, an egg-timer. I stare at the ceiling for inspiration. This gives me a nosebleed, which wastes another good half-hour while I staunch it and clean up the attendant mess. Finally, I can conjure no further distractions and reluctantly put fingers to keys. I start typing, aiming in the general direction of this issue's theme. Sometimes I manage to get in a few sentences that have a modicum of relevancy, but more often I miss that mark by a wide margin.

Eventually, by dint of sheer perseverance (or is it perversion? I always get those two confused), I cobble together a sufficient number of words arranged in such a way that they can be mistaken by the incautious observer for a column, if one ignores the total lack of cohesion. The result looks a lot like...

This.

Book Reviews

ELIZABETH ZWICKY, WITH SAM STOVER

12 Essential Skills for Software Architects

Dave Hendricksen

Addison-Wesley, 2011. 242 pp.
ISBN 978-0-321-71729-0

This is a book about non-technical skills for technical types who want to get into the higher reaches of designing software. It's not a book about managing, and it's not a book about architecting software, either. It's about what are often called "soft skills" (as if programming were inexorably scientific and talking to people nicely was easy). It lays out, carefully and clearly, the non-technical skills you need to succeed in getting things done in groups of people you are probably not in charge of: how to get along with people, how to talk to management, how to think about and talk about risk and failure.

I agree with the author almost all the time, and I think the book is a useful guide for people who may find themselves blocked for mysterious reasons. If you know you're right but you're not winning, this book will tell you why and will give you a good idea of what the winning strategy would be. It is as non-political as it is possible to be; that is, it advises that you think about strategies, personalities, and implications, but it does not advise you to be devious or manipulative.

What it won't do is get you all the way to implementing these skills. It's one thing to know that being right is not the most important thing; it's another thing entirely to manage to implement that in a meeting. If these skills don't come naturally to you, you're going to need to do some follow-up reading to pick up implementation strategies, and then you're going to need practice and probably assistance to get good at using them. It's not impossible to do, but it's not easy, either.

Privacy and Big Data

Terence Craig and Mary E. Ludloff

O'Reilly Media, 2011. 79 pp.
ISBN 978-1-449-30500-0

Designing Data Visualizations

Noah Iliinsky and Julie Steele

O'Reilly Media, 2011. 93 pp.
ISBN 978-1-449-31228-2

Big Data Glossary

Pete Warden

O'Reilly Media, 2011. 43 pp.
ISBN 978-1-449-31459-0

Building Data Science Teams: The Skills, Tools, and Perspectives Behind Great Data Science Groups

DJ Patil

O'Reilly Media, 2011. 25 pp.
ISBN 978-1-449-31623-5

Big Data Now: Current Perspectives from O'Reilly Radar

O'Reilly Media, 2011. 125 pp.
ISBN 987-1-449-31518-4

Here's a whole heap of short books, published by O'Reilly, about "big data." I will follow the example of the authors and leave the definition of "big data" vague; it's definitely more data than will fit on your laptop, probably more data than you have at home, and maybe more data than you feel comfortable having in one place for somebody to poke through. Aside from the common theme, the books vary greatly.

Privacy and Big Data tells you (most of) why you should be worried about big data, what the rules around it are, and where those rules come from. Its primary audience is not big data processors but other people who are thinking about how these giant data collections affect them. I found it competent, but as somebody whose employer's business is based on big data, I felt that there were some important omissions. The authors reliably fail to distinguish between

intentional actions by companies and accidents, presenting (for instance) Google's collection of wireless data while doing Street View photos as if it were an intentional policy. This is important not only because it paints big data holders as worse than they actually are, but also because it implies that you only have to worry about companies that intend to invade your privacy. In fact, as in most things, well-meaning ineptitude, accidents, and oversights do as much damage to consumers as intentional violations. In addition, there is little discussion of the loopholes and inconsistencies of data privacy law; almost all data privacy rules have an exception for data collection to provide security measures and detect fraud. Those exceptions are very important and useful, but they also lead to piles of tempting data that would otherwise be uncollectable left lying around.

Designing Data Visualizations is a nice overview of the issues. It isn't the only data visualization book you'll ever need, but if you're looking for a good, short lead-in to the processes and issues, it's a nice place to start, with a practical bent and a helpful bibliography to take you to the next level. If you're staring at your Excel charts with a sinking feeling and no idea what to do to fix them, this will show you where to go.

Big Data Glossary is another introduction. It is not, as you might have expected, a glossary. Instead, it's an introduction to big data tools and languages; sort of a tour guide to the land of big data, detailing the main hotels and attractions, important phrases in the language, and how to get there. Again, this is a nice starting point; it's not going to get your data ready for human consumption, but it will at least enable you to understand the pieces of the problem and contemplate which ones you want more information on.

Building Data Science Teams is free, which is good; it's short; and it details one very specific approach. When it says "data science teams," it means it; silly of me, I guess, but I was hoping for something about teams that deal with big data in general, and the author is really only interested in teams where you put "people who think about big data" on one team and people who're related to particular subjects elsewhere. And he's very biased towards advanced degrees in the people who think about big data. It's a valid approach, and he has some interesting ideas about running teams and selecting people, but as a detailed description of a single approach, it's only going to be useful as a whole if you happen to have the right problem and the right environment.

Big Data Now is also free, and it's an anthology of short pieces already published. They contain their original advertising, which is a slightly odd effect, since they're often talking about conferences which are now over. As you can see from this collection of reviews, "big data" as a topic doesn't make for a terribly cohesive document. But there were several pieces here I found particularly useful or illuminating: Mike Loukides on "Data hand tools" (yes, I work at the home of Hadoop, but sometimes knowing how to run grep correctly over a terabyte of data is still faster than moving that data onto a Hadoop cluster), Pete Warden on "Why you can't really anonymize your data," and Alistair Croll on "There's no such thing as big data."

Programming Pig

Alan Gates

O'Reilly Media, 2011. 193 pp.
ISBN 978-1-449-30264-1

(Disclaimer: I am employed by Yahoo!, the primary location of Pig development and the copyright holders for this book. My corporate overlords have not, however, expressed an opinion on this book to me.) If you need to program in Pig, you are going to want a copy of this book around. If you're not sure whether you need to program in Pig, it's a scripting language used with Hadoop; and if that doesn't help, either you don't deal with big data or you need a copy of the *Big Data Glossary*.

Pig is an odd little language (and it is definitely a little language; 193 pages is enough to explore the entire language, the common extensions to it, and the ways to write your own extensions). It has a glancing resemblance to SQL, from which it borrows some syntax, but in spirit it's more like R: mind-bendingly fitted to its problem domain, to the extent of introducing new data types. When I utter sentences like, "Remember, that's not a bag, it's a tuple," I begin to wonder what non-obscene nouns have not yet been used to describe data structures, and whether I will some day in all technical seriousness borrow a cup of data from my neighbor.

Pig's strong point is its ability to express a lot of data transformations simply and in such a way that the software can do a reasonable job of optimizing them, sometimes with your assistance. Its corresponding weak point is its specialization. It's very easy to learn enough Pig to express simple queries, but then there tends to be a wall of non-comprehension where it seems like things must be expressible but you're not quite sure how. This is a good book for getting you past that, to the point where you can tell a bag from a tuple, write nested filters, and use Pig to manage complex data flows (the ability to handle multiple inputs and outputs is one of the great advantages of Pig over straightforward Hadoop streaming).

Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains

Edgar G. Daylight

Lonely Scholar Scientific Books, 2011. 119 pp.
ISBN 978-94-9138-600-8

This is a transcription of an interview with Peter Naur, best known to most of us as the Naur in Backus-Naur notation. The interview covers the rest of his career, never mentioning the notation. It breaks into roughly three topics: a section on the early history of computing, one on issues about computing and formalism, and one on Naur's theories of neurology.

I expect that not very many people will find all three equally gripping. I understand that there are people who are fascinated by the early history of computing, but I am not one of them; sadly, I would rather hear the interpersonal gossip that Naur carefully (and very appropriately) avoids discussing. And while his theories about neurology are interesting, I have a well-earned distrust of very smart people who hypothesize outside the fields they are expert in.

From that you can deduce that I liked the section on formalism, particularly Naur's assertions that as far as he can tell, it's not terribly useful for programming—programming is an idiosyncratic process—and most of the people who describe neat, beautiful ways they arrive at solutions to problems are, if not lying, creatively rearranging the truth to make better reading.

If this sounds gripping to you, I recommend searching it out; if you're on the fence, it's probably not going to win you over. The interview format is always a little clunky, and it is not perfectly implemented here.

MongoDB and Python

Niall O'Higgins

O'Reilly Media, 2011, 66 pp.
ISBN 978-1-449-31037-0

Since I started messing around with Hadoop, I've been exposed to a couple of other similar technologies, and one of them was MongoDB. I'll admit (more than) half the reason I picked this book up is the Python part, and I figured I'd see what MongoDB can do. Now that I see how easy it is to use MongoDB with Python, I really need to learn it from the ground up. That aside, this book packs a lot into its 66 pages and certainly wasn't a waste of my time.

There are just four chapters, and the first gets you started with a brief intro to MongoDB and some comparisons with both traditional and NoSQL databases. Installing, running, and setting up a Python environment round out Chapter 1 in typical O'Reilly fashion. Chapter 2 gives you the basics of interacting with a MongoDB: connecting, getting a database handle, insertion, queries, etc. MongoDB is a document-oriented database, and this chapter does a decent job of explaining how that is different from a relational database. Two things I really liked about this chapter were (1) comparisons to SQL concepts and (2) consistently putting things into Pythonic terms. I might not know what a JSON document is (well, I didn't then), but I know what a Python dictionary is. This is a database book written for people who know databases—which I am not. Keeping things in Python style made it much easier for me to follow what was going on.

Chapter 3 goes a little deeper into different ways to use MongoDB more efficiently: The concept of embedding documents, while mentioned previously, is explained in depth, and there are some suggestions for making your queries more efficient with proper indexing, among others. The really cool feature though, which I was totally not expecting, was the section on geospatial indexing, which is supported out of the box. MongoDB uses a public domain algorithm (geohashing), which "translates geographic proximity into lexical proximity." This is a pretty cool capability and it's not hard to use—examples given in Python (grin).

Chapter 4 discusses integrating MongoDB with three Web frameworks: Pylons 1.x, Pyramid, and Django. Pylons and Pyramid are somewhat similar, and it seems that Pyramid is the more active of the two. Django differs from them in that it is "one well-integrated package" with its own set of templates, interfaces, etc. Regardless of what your needs are, chances are good that one of these will fit.

Overall, a solid book with lots of examples and code. Installing and setting up MongoDB, Pylons, Pyramid, and Django are all covered well and should make getting started pretty simple and fast for anyone. As a Python guy with a little database experience, I still found the book very accessible and am already coming up with ideas on what I can use MongoDB for. IMHO, MongoDB does not replace or compete with Hadoop but, as a document-oriented database, provides answers to different questions.

—Sam Stover

Privacy and Big Data

Terence Craig and Mary E. Ludloff

O'Reilly and Associates, 2011, 79 pp.
ISBN 978-1-449-30500-0

Right out of the gate, the authors give the disclaimer that they are executives from a "growing startup in the big data and analytics industry," which caught me off guard. I was expect-

ing this to be a Top Technical Tips for preserving online privacy, and what I got was something different—very interesting and educational, but different. It's a short book—five chapters and fewer than 80 pages, but chock full of interesting facts and tons, I mean *tons*, of references. Not that a book should be judged by end/foot notes, but there are a lot: 218 by my count. This makes for a lot of research potential, should you want to go chasing down the facts they present.

Chapter 1 starts with ARPANET and takes you to today; from zero personal data online to the mess we all live in now. It's an interesting chapter. While at first glance you might think it doesn't bring anything that we aren't all aware of, one point that really jumped out at me was that it's not just about advertising. There are lots of other ways to use our personal information out there, and they're probably scarier than presenting you with a targeted ad (which no one likes anyway).

Chapter 2 deals with privacy in the digital age, with an emphasis on US vs. EU stances. It's probably the driest of the chapters but, again, was just full of stuff that I didn't know. The authors are of the opinion that in the US, privacy is a commodity that we can use to barter for stuff/conveniences we want for "free." In the EU, privacy is a basic human right "that transcends commoditization." Pretty heady stuff, but it does make you think. Well, it made me think.

Chapters 3 and 4 deal with The Regulators and The Players, respectively: lots of discussion on who is doing the regulating, how they are doing it, pros and cons of the different approaches, as well as how the Players deal with (or ignore) the Regulators. It gives plenty of examples, some that you might not be aware of, some you probably have seen on the news.

Chapter 5 wraps everything up and lays out the most important point of all. Whether you view your privacy as a right or a commodity, the fact still stands that once you release information into the Internet, it will (probably) never go away. No matter what legislation comes about, no matter what rights you think you may have, "you can't unring the bell." It might not be scary, or it might be, depending on how paranoid you are, but either way, this book is a good read. It's not going to walk you through protecting yourself, although there is some discussion on ways to make it harder for your privacy to be invaded. But I think after you read this, you might just start looking at how much of you exists on the Internet, and what you can do about it. Plus, it's pretty non-technical, so if you're already paranoid but know people who aren't, you may have just found a great present for them.

—Sam Stover

Conference Reports

In this issue:

Every two years, 75–100 systems, database, and application developers and researchers gather at Asilomar for the Workshop on High Performance Transaction Processing Systems (HPTS). The name, something of a misnomer, stems from its origin in 1985, when Jim Gray, Dieter Gawlick, Andreas Reuter, and other luminaries invited practitioners and academics to discuss the challenges and successes in the area of large, scalable, high-throughput systems. Today, I think of HPTS as the place where people with large-scale problems come to talk with people who like to solve large-scale problems. The crowd is a seamless blend of researchers and practitioners and infrastructure suppliers and consumers.

Each HPTS seems to have a dinstinctive flavor. A few years back, it seemed that all the large online service providers were talking about how they used Lucene to solve large-scale problems. This year, there was a lot of talk about integrating NoSQL solutions into large-scale services. HPTS feels a lot like HotOS, but with more emphasis on data and less emphasis on operating systems. This year Rik Farrow went to HPTS to soak in the ambience, learn a bit about the community, and coordinate student scribes so that we could bring a taste of HPTS to the USENIX community. Unlike many USENIX workshops, HPTS is not based on paper submissions; the written record mostly consists of personal blog postings, a collection of presentations, and some less-than-one-page submissions. These reports are the closest thing you'll find to an HPTS proceedings, although Web surfing will reveal several personal blog reports.

—Margo Seltzer, USENIX Acting Executive Director

14th International Workshop on High Performance Transaction Systems (HPTS)

Pacific Grove, CA
October 23–26, 2011

Datacenter Trends 101

Summarized by Eugenia Gabrielova (eugenia.g@uci.edu)

Internet-Scale Datacenter Economics: Where the Costs & Opportunities Lie

James Hamilton, Amazon

James kicked off HPTS by saying it is his favorite conference, primarily because of the people in the room. He then claimed there has been more innovation in the past five years than in the previous fifteen, primarily due to advances in cloud computing and the accessibility it provides to application developers. Datacenters are expensive and don't really help innovation—when you are spending millions or billions of dollars, you do things the way you know it will work.

At Amazon, there are always multiple datacenters under construction. In the past four years, AWS has evolved into a phenomenal business generating tons of revenue and passing on savings to customers. Amazon was approximately a $2.7 billion annual revenue enterprise in 2000. Now, every day Amazon Web Services adds enough capacity to have support ed all of Amazon.com's infrastructure in the company's first five years. There is a competitive advantage in having better infrastructure.

The talk shifted to everything below the OS, because that is generally where the money goes. Charts often show people costs, but at a really large scale these costs are very minor relative to the costs of servers and power distribution. As a rule of thumb, "If you want to show people your infrastructure, you're probably spending too much." In the monthly costs of a datacenter, servers (not power distribution) dominate. However, server costs are decreasing, while networking costs are creeping up. Networking is a problem precisely because it is "trending up," so it is broken—this is a huge opportunity for innovation.

Another area with great potential for innovation is cooling systems, which have remained the same for about 30 years. Fans moving air is expensive, and moving water is also fairly expensive. Datacenters of the future could be designed beautifully with eco-cooling, no AC required. In the meantime, modular and pre-fabricated datacenters are regaining popularity, because of how quickly they can be deployed. Making datacenters better isn't just a technical advantage, it is an enormous business advantage.

Bruce Lindsay (Independent, ex-IBM) commented on the declining cost of network ports. Someone asked about OpenFlow, and James said that Google supports Quagga for routing, and OpenFlow comes from Stanford. Both open up the infrastructure by allowing the control plane to run centrally, with cheap hardware for running the data plane.

Someone noted that standard practice in the computer industry is to "prepare for the worst." James replied that there are test sites running with high-voltage direct current and many high-profile datacenters have very robust strategies for ensuring uptime (such as fully dedicated power generators). However, due to high demand, it can be hard to know which workloads will be running in a datacenter at a given time.

Slides from this talk can be found at http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_HPTS2011.pdf. James can be reached at James@amazon.com.

The Rise of Dark Silicon

Nikos Hardavellas, Northwestern University

Dr. Hardavellas was unable to make it to HPTS this year but has made the slides for his talk available at www.hpts.ws/sessions/Hardavellas.pdf.

The Hitchhiker's Guide to Precision Time Synchronization

Krishna Sankar, Egnyte

Before he become Lead Architect at Egnyte, Krishna was a Distinguished Engineer at Cisco Systems. In his free time he enjoys working as a technical judge for FIRST LEGO League Robotics. He began his talk by emphasizing that time synchronization is different from time distribution. There is incredible value in offering time precision in an application. Ocean observatory networks, industrial automation, cloud computing, and many other fields would benefit. Time synchronization is also slowly finding its way into routers and blade server fabrics.

Krishna gave an overview of IEEE 1588 v2 PTP (Precision Time Protocol), which concerns the sub-microsecond synchronization of real-time clocks in components of a network distributed measure and control system. This capability is intended for relatively localized systems, like those often found in finance, automation, and measurement. The purpose of IEEE 1588 is simple installation, support for heterogeneous clock systems, and minimal resource requirements on networks and host components.

PTP uses a master/slave model to synchronize clocks through packets over unicast and/or multicast transport. It follows a simple protocol: master and slave devices enabled with PTP send messages through logical ports to synchronize their time. Of the five basic PTP devices, four are clocks. Each clock determines the best master clock in its domain, including itself. It is very difficult to achieve high precision, so some hardware-assisted time stamping can be used to help accuracy (which is more complex than it sounds). A few key lessons in working with PTP are that shallow, separate networks are preferable; anything too hierarchical will prove difficult to manage and synchronize. Accuracy depends largely on hardware and software abilities and interaction. Additionally, GPS satellite visibility is needed for the GMC (Grand Master Clocks, the most accurate).

Krishna closed by encouraging audience members to submit to ISPCS 2012, which will take place in San Francisco. Learn more at http://www.ispcs.org. A central theme of the Q&A was whether these time precision techniques are accessible to the average application developer. How can an average application, subject to layers of virtualization and delays, take advantage of precision timing? The main takeaway was that, with some planning, developers can certainly take advantage of advances in time synchronization. The slides for this talk can be found at http://www.hpts.ws/sessions/Synchronization.pdf.

Not Your Traditional Data Management Session

Summarized by Andy Pavlo (pavlo@cs.brown.edu)

Enterprise Supercomputing

Ike Nassi, SAP

Ike began with a harsh denunciation and lamentation about current enterprise computing hardware, which supports only a single TB of DRAM on a single motherboard. That limitation makes it difficult for servers to be used for enterprise computing systems, because they often have a much greater working set size. Ike strongly believes it is time to re-examine our current predilection for shared-nothing architectures and that the database research community should take advantage of developments in high-performance computing research from the past 25 years, which has favored a shared-everything architecture. Large memory systems on the scale required by SAP are simply not being built; thus Ike sought to create one himself.

Ike presented a new DBMS server architecture, currently under development at SAP, which uses a virtual shared-

everything paradigm built on a single rack cluster. In SAP's new system, the database executes on a single instance of Linux, while underneath the hood the ScaleMP hypervisor routes operations and data access requests over networking links (i.e., no shared buses) to multiple, shared-nothing machines. By masking the location of resources through a coherent shared-memory model, Ike argues that they are able to minimize the amount of custom work individual application developers have to do in order their database platforms.

The early morning audience was languid, but several skeptics, such as Margo Seltzer, were concerned that the data links between machines would not match the speed of DRAM. Ike assured these doubters that high-performance communication links such as InfiniBand would be sufficient for this system. He also remarked that the system currently does not support distributed transactions; thus there is no message passing needed between nodes. Roger Bamford (Oracle) asked, why divide the system into so many cores, and Ike replied that they need the RAM. Adrian Cockcroft asked how common failures are. Ike said that this is a lab test so far, and in thirty days there were no failures. Margo Seltzer said she loved this project, which reminded her of late '80s shared memory multiprocessor systems such as Encore. Ike said that unlike the early systems, which used busses, their system is using fast serial connections, and he suggested that people not be blinded by what happened in the past. Both Margo and James Hamilton wondered about the problem of having a NUMA architecture, especially when the ratio of "near" memory to "far" memory reaches 10 to 1. Ike said that he lied, that all memory is used as if it were L4 cache. Roger pointed out the cost of going to the cache coordinator, and Ike replied that identifying the location of memory has a constant cost.

Forget Locality

Randal Burns, John Hopkins University

Randal Burns, a systems research professor at Johns Hopkins, raised the issue that the canonical optimizations used in DBMS systems were insufficient to achieve high-performance data processing (i.e., > 1 million IOPS) on large and complex graph data sets. This is because any algorithm that must perform a scan of the entire data set or a random walk in the graph cannot take advantage of locality in the data. Thus, optimizations such as partitioning, caching, and stream processing are rendered impotent.

Randal then discussed ongoing work at Hopkins that seeks to understand the main bottlenecks that prevent modern systems from scaling to larger I/O operation thresholds. His work shows that low-level optimizations to remove lock contention and interference can improve throughput by 40% over file access through the operating system.

Margo Seltzer asked whether making certain assumptions about the physical layout of the graphs could be exploited. That is, could performance be improved if the system stored the data in a way that optimized for a particular processing algorithm? Randal responded that such techniques would be unlikely to work for attribute-rich graphs, since there is no optimal ordering. Roger suggested that he put the answer in their database and be done with it, eliciting laughter. Mike Ubell asked whether the cache was throttling IOPS, and Randal said yes, that there is lots of bookkeeping and page structures to manage. James Hamilton asked why not have the database use memory directly, and Randal said that is where they are going. They want to get away from local and global data structures. James pointed out that databases had already done this. Mohan asked about latches, and Randal replied that they want only locks that matter, such as a read lock on dentry and on mapping.

Someone suggested proper indexing, declaration of graph processing, having the database make decisions in advance. Randal replied that that is ground that has been trod before. Someone else pointed out that it seemed they were looking for storage memory that had DRAM-like characteristics. Randal agreed, saying that without a memory hierarchy his talk would be a no-op.

Flexible Hardware for Flexible Data Intensive Software

Arun Jagatheesan, Samsung

Arun Jagatheesan from Samsung shared his perspective on new hardware trends and configurations for big-data systems and supercomputing platforms. He was specifically focused on the flexibility of both the hardware systems (i.e., allowing administrators to configure the hardware) and the software platforms that they support (i.e., allowing users to execute variegated workloads). Arun began with an overview of the flash-based Gordon system that he helped to develop while at the San Diego Supercomputer Center in 2009. Arun said that the three main lessons that he learned from this project were (1) not all the configuration options that one needs are available in hardware, (2) there is a nebulous tradeoff between flexibility and performance, and (3) manufacturers, applications, users, and administrators are unprepared for new hardware.

From this, Arun then introduced his more recent work on Mem-ASI at Samsung. Mem-ASI is a memory-based storage platform for multi-tenant systems that is designed to learn the access patterns and priorities of applications and react to them accordingly in order to improve throughput. Such priorities could be either service-level hints from applications, service-level requests from the computing platform's infrastructure, or simply how the individual application accesses data. This additional information could be used by the

system for more intelligent scheduling and resource management. Arun believes that such a model could both improve performance and possibly reduce energy consumption.

James Hamilton said he could understand the power savings, but not the factor of 4 for performance gains. Arun said that the idea is that you can change something on the memory controller to change what is happening at the transport layer. James asked if this had to do with the number of lanes coming off the core, and Arun replied that it is not about lanes but about what you can do behind those lanes.

Mapping the NoSQL Space

Summarized by Aaron Elmore (aelmore@cs.ucsb.edu)

The NoSQL Ecosystem

Adam Marcus, MIT

Adam Marcus provided a brief history of the origins of NoSQL, beginning in the late 1990s with Web applications developed using open source database systems. Applications that saw increased load began wrapping stand-alone DBMSes to allow for sharding to achieve scale. Additionally, relational operations were removed and joins were moved to the application layer to reduce costly database operations. These modifications led to the creation of databases that went beyond traditional SQL stores and came to be referred to as Not Only SQL (NoSQL). With a plethora of recent NoSQL options, Adam lightheartedly introduced Marcus's Law, which tells us that the number of persistence options doubles every 1.5 years.

The majority of NoSQL stores rely on eventual consistency and are built using a key-based data model, sloppy schemas, single-key transactions, and application-based joins. However, exceptions to these properties were highlighted, including alternatives to data models, query languages, transactional models, and consistency. For example, while many NoSQL databases utilize eventual consistency, many alternatives exist, such as PNUTS's timeline consistency or Dynamo's configurable consistency based on quorum size. With a basic understanding of NoSQL properties, real-world usage scenarios were outlined.

Recently, Netflix has undergone a transition from Oracle to Cassandra, to store customer profiles, movie watching logs, and detailed customer usage statistics. Key advantages that motivated the migration include asynchronous datacenter replication, online schema changes, and hooks for live backups. More information about this migration is detailed in Adrian Cockcroft's paper at http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra. Contrasting Cassandra, Facebook chose HBase for the new FB Messages storage tier, primarily due to difficulties in programming against eventual consistency. HBase also provides a simple consistency model, flexible data models, and simplified distributed data node management. MongoDB usage for Craigslist archival and Foursquare check-ins were briefly highlighted.

After detailing NoSQL databases and use cases, Adam presented takeaways for the database community. First, and most contentiously, is developer accessibility. Adam said that the ability of a programmer to set up and start using a NoSQL db really mattered. Bruce Lindsay (ex-IBM) strongly objected to the question on "whether first impressions made within five minutes of database setup and use matter." Margo Seltzer (Harvard) countered that a new generation of developers, who use frameworks such as Ruby on Rails, do make decisions on accessibility and that these developers should matter. Adam furthered the argument by claiming accessibility will matter beyond minutes in schema evolution, scaling pains, and topology modifications. Database development should also examine the ecosystem of reuse found in some NoSQL projects. This is exemplified in Zookeeper, LevelDB, and Riak core becoming reusable components for systems beyond their initial development. Lastly, the NoSQL movement espouses the idea of *polyglot persistence*, where a specific tool is selected for a task. Selecting various data solutions can create painful data consistency issues, as an enterprise's data becomes spread among disjoint systems.

In closing, Adam presented several open questions. These focused on data consistency, datacenter operational tradeoffs, assistance for scaling up, the ability to compare NoSQL data stores, and next-generation databases. A question by C. Mohan (IBM) about the need for standardization of a query language drew mixed reactions.

The Present and Future of Apache Cassandra

Jonathan Ellis, DataStax

Jonathan Ellis, of DataStax and a major contributor to the Apache Cassandra project, outlined developments in the recent version 1.0 release and goals for future Cassandra releases. Inspired by Google's BigTable and Amazon's Dynamo, Cassandra began as a project at Facebook before becoming an Apache incubator project. Cassandra's popularity is partly due to the ability for multi-master (and thus multi-datacenter) operation, linear scalability, tunable consistency, and performance for large data sets. Cassandra's user base today includes large companies such as Netflix, Rackspace, Twitter, and Gamefly.

For release 1.0 of Cassandra, leveled compaction was introduced to improve the reconciliation of multiversion data files. Advantages over the previous size-tiered compaction include improved performance due to lower space overhead and fewer average files required for read operations. Addition-

ally, individual nodes can construct local secondary indexes on columns; however, denormalization and materialized views are necessary to avoid join operations. Improvements mentioned but not discussed were compression, expiring columns, and bounded worst-case reads.

An interesting application that was developed for Cassandra was the ability for eventually consistent counters. Every node in the system maintains a list of counter values associated with each node. Any local modification for the counter performed only modifies the replica's value of the counter. To ascertain the value of the counter, all replica values are summed. This allows for concurrent modifications to the counter without needing synchronization between nodes.

Heavy optimizations were undertaken to improve read and write performance for Cassandra, including JVM tuning and garbage collection. Future advances in Cassandra will involve easing administration and use, improving the query language, support for range queries, and introducing entity groups. Pat Helland (ex-Microsoft) asked how to improve the performance of random reads for large data sets. Jonathan said a reliance on SSD would be needed to make significant gains. Someone wondered why Facebook had moved from Cassandra to HBase; Jonathan answered that it was mostly a personnel issue within Facebook. Mehul Shah (Nou Data) asked about the advantages of developing in Java. Jonathan said they included core consistency, memory management, immutable collections, and a rich ecosystem. The last question was about the largest install of Cassandra. Jonathan thought that it was around 400 nodes and 300 TB of data.

Oracle's NoSQL Database

Charles Lamb, Oracle

Charles Lamb began the presentation on Oracle's latest data store with what NoSQL means to Oracle. A NoSQL database encompasses large data, distributed components, separation of OLTP from "business intelligence," and simplified data models, such as key-value, document stores, and column families. Lamb said that Berkeley DB alone does not meet all of these requirements and that the focus of the Oracle NoSQL DB is a key-value OLTP engine. Requirements for the database include support for TB to PB scale data sets, up to one million operations per second, no single point of failure, predictably fast queries, flexible ACID transactions, support for unstructured or semi-structured data, and the ability to have a single point of support for the entire stack, from hardware up to the application.

The system has multiple storage nodes, potentially residing in multiple datacenters, and is accessed by a jar deployed within the application. This jar, or driver, maintains information about the state of each storage node. Data is accessed using major and minor keys, and all records with the same major key are clustered on the same replication group of storage nodes. Operations are simple CRUD (create, read, update, and delete), read-modify-write (or compare-and-set style), and iteration. CRUD may operate on one or more records with the same major key. ACID transactions are provided but may not span multiple API calls. Iteration is unordered across major keys and ordered within major keys. Management and monitoring of the system are available through a command-line interface and Web-based console. Oracle's NoSQL database is built upon the battle-tested, high-throughput, large-capacity, and easy-to-administer Berkeley DB Java Edition/High Availability. Since Berkeley DB JE/HA was built for a single replication group, features such as data distribution, sharding, load balancing, multi-node backups, and predictable latency (which was highlighted as a difficult goal) were required to achieve better scaling.

Hashing a major key, modulo the number of partitions, identifies the group of nodes responsible for storing replicas of a data record; this group provides high availability and read scalability. The Rep[lication] Node State Table (RNST) identifies the best node to interface within a replication group. The RNST is stored at the driver and is updated by responses sent to the client. From the RNST the driver can determine a group's master, staleness of replicas, last update time, number of outstanding requests, and average trailing time for a given request. Replication is single-master, multi-replica, with dynamic group membership provided by election via the Paxos protocol. Durability can be configured at the driver or request level, and there are options for disk sync on both the master and replicas and replica acknowledgment policies. Consistency can be specified on a per-operation basis as well, with options to read from (1) the master, (2) any replica that lags no more than a specified time-delta from the master, (3) any replica that is at least as up-to-date as a specific version, or (4) any replica (i.e., with no consistency guarantees). The presentation concluded with an evaluation of the database's performance and scale-out capabilities.

Mohan asked about multi-node backup. They can do that, but it will not be consistent. As with Cassandra, they can take a snapshot for a consistent backup. Roger asked how they are supporting read-modify-write. Charlie said that the application does a get, does operations, then a put-if-version, and, conditionally, updates. Mohan wondered if reads are guaranteed to see the final versions, and Charlie answered that he would cover that later, but there are no guarantees.

There was vigorous discussion after Charlie finished. Mohan asked if the data and operations log were stored on the same disk. Margo Seltzer, who is also involved with Oracle NoSQL, said that they use a log-structured data store and that data and log are stored the same way. James Hamilton wondered if they could migrate off a node if it gets hot, and Charlie

replied, Not in this version. They do use hashing for even data distribution. Shel Finkelstein (SAP) asked about time-based consistency. Charlie explained that data is tagged with a Java-based timestamp. Mehul Shah wondered if they can continue operations after a partition, and Charlie said they could do reads but not writes without access to the master for that major key. Mehul then wondered if they can move partitions around and Charlie replied, Not in this release.

Someone asked if the drivers knew about all partitions. They get initialized on the first request and can connect to any replication nodes. Roger asked Charlie to describe their access control model. Charlie said that the assumption is that the system is in a DC, producing an "OMG" response.

Big Data Experiences & Scars

Netflix Goes Global

Adrian Cockcroft, Netflix

Summarized by Hatem Mahmoud (hatem@cs.ucsb.edu)

Adrian Cockcroft described the process of migrating Netflix to a public cloud in order to provide highly available and globally distributed data with high performance. The migration focuses on the control plane (e.g., users' profiles, logs), not the actual movie streaming, which is done using CDN. Amazon AWS was chosen as the public cloud to host Netflix's services because it is big enough to allocate thousands of instances per hour as needed. Adrian mentioned a remarkable idea in his presentation: the notion of design anti-patterns, that is, that design is better defined by undesirable properties than by desirable ones.

The Netflix migration involved a bi-directional replication phase in which data was replicated between Oracle and Simple DB, while backups remained in the datacenter via Oracle. Later on, replication of new account information to the datacenter was eliminated. Each data item is replicated to three different zones (i.e., different buildings with different power supplies within the same datacenter). This keeps all the copies close for fast synchronization. There is a tradeoff between recoverability and latency; to achieve the lowest latency a write operation must acknowledge once it is done on at least one replica, while to achieve the highest recoverability a write operation has to wait for all three replicas to be updated before acknowledging the user. The middle path is to use a quorum of two replicas. Overall, Netflix's data are distributed across four Amazon regions, plus a backup region. Remote replication can be also achieved through log shipping.

Backup is done by: (1) taking a snapshot (full backup) periodically by compressing SSTable and storing it to S3; (2) doing incremental compressed copying to S3 triggered by SSTable writes; or (3) scraping the commit log and writing it to EBS every 30 seconds. Also, there are multiple restore modes, multiple ways to do analytics, and multiple methods for archiving. Backups are PGP encrypted and compressed, with the lawyers keeping the keys for encryption. If S3 gets broken, they also make an additional copy to another cloud vendor.

Adrian pointed out that they find cloud-based testing to be frictionless. As an example, he asked a Netflix engineer to spin up enough Amazon instances to perform one million client writes per second. It took a couple of experiments to come up with the correct number of nodes, 288, to do this, and a total of two hours and about $500 of Amazon charges.

Margo Seltzer asked the size of their biggest database. It is currently 266 GB. Adam Marcus (MIT) asked if engineers had their own machines, to which Adrian replied that they used Jenkins for build testing,and had a special Eclipse plugin for working with EC2.

Towards Improved MySQL Scalability and Reliability

Ryan Huddleston, RightNow

Summarized by Rik Farrow (rik@usenix.org)

Ryan described RightNow as a company that provides MySQL as a service. Located in Bozeman, Montana, the one-thousand-person company provides database services, on the company's servers, for over two thousand customers worldwide. The US military is one of their larger customers. RightNow uses the Percona Server MySQL port and has paid companies like Percona to add features to MySQL. In 2001 they paid to have the Innodb file-per-table feature added. They found they needed to switch from ext3, the default Linux filesystem, to XFS, because file deletion time was scaling with file size. Someone asked if this is still an issue with ext4, and Ryan said it was. James Hamilton asked if `create table` was an issue, and Ryan said it never had been an issue.

Ryan discussed their technique for migrating customers between shared servers when a customer's load becomes too great. James Hamilton wondered how they prevent a single customer from dominating a server. Ryan said they had a system that keeps track of load and can migrate a customer to another node. It keeps track of queries and can queue queries that will take a long time and move the queries from real-time to batch. Ryan said that their goal is to remain an open source company and that they plan to push all patches up to MariaDB (a branch of MySQL).

Bruce Lindsay asked whether adding a column requires them to delete a table. Ryan said it does. They add tables/columns on a slave server, move data in batches, cut over columns and tables, then drop tables and columns. Then they snap the customer to the slave and alter the master while doing updates. The entire process appears to occur with no delay for queries.

Someone exclaimed, "This was all fixed 25 years ago!" Ryan calmly replied that if they were doing this on Oracle, it would cost them $25 million a year. Instead it costs them $100,000 for support of MySQL.

Storage Infrastructure Behind Facebook Messages

Kannan Muthukkaruppan, Facebook

Summarized by Hatem Mahmoud (hatem@cs.ucsb.edu)

Kannan Muthukkaruppan explained why Facebook has moved from Cassandra to HBase as a storage system for Facebook messages, the architecture used, and the lessons learned from that experience.

HBase is used to store small messages, message metadata (thread/message indices), and the search index, while large messages and message attachments are stored in Haystack. HBase was chosen for its high write throughput, good random read performance, horizontal scalability, automatic failover, and strong consistency. Besides, by running HBase on top of HDFS the system takes advantage of the fault tolerance and scalability of HDFS, as well as the ability to use MapReduce to do analytics.

Each of the datacenters that host Facebook's data is considered a cell that is managed by a single HBase instance. A cell contains multiple clusters, and a cluster spans multiple racks. Each user is assigned initially to a random datacenter, although the user may later be migrated to another datacenter via a directory service. Typically, a datacenter consists of several buildings. Thus each data item stored in HBase is replicated three times, in three different buildings.

The migration to HBase took more than a year. Shadow testing was used before and after rollout. To account for potential bugs, Scribe was used to write offline backups to HDFS, both locally and at remote datacenters. The developers had to introduce several modifications to HDFS to improve reliability, including sync support for durability, multi-column-family atomicity, several bug fixes in log recovery, and a new block replacement policy to reduce the probability of data loss. Also, to improve availability, the developers introduced rolling upgrades to account for software upgrades, online `alter table` to account for schema evolution, and interruptible HFile compaction to account for cluster restarts and load balancing. The developers also added several modifications to improve performance and solicit fine-grained metrics.

Someone asked whether they have an additional sharding layer on top of HBase. Kanna said yes, but that HBase only works within a single DC. Margo Seltzer asked if users are mapped to cells randomly. Kanna said yes and that they can migrate users later. Overall, they average 75+ billion read-write IOPS per day, with a peak of 1.5 million operations/second. Their load is 55% read and 45% write, over 6 PB of data (2 PB with three replicates), all compressed using LZH. Margo asked if they lose all the users within a DC if it goes down, and Kanna replied that they do offline backups to other DCs. Cris Pedregal-Martin asked if they had non-peak hours. Kanna answered that Monday between 12 and 2PM is their peak, so in a sense, yes. Adam asked if they planned on upstreaming their patches to HBase. Kanna said that they do, as most of what he talked about is open source.

Someone asked about network speed. Kanna said they use 1G at hosts and 10G at the top of racks. Mike Caruso asked what type of changes they made to the schema. Kanna said that making threads longer meant writing metadata back to HBase, so they fixed that as an example. Then he said there is lots more work to be done, such as fixing the problem of a single HDFS Name Node and having fast hot backups. Mohan asked if all users are mapped to US DCs, and Kanna said yes. Cris asked if they ever lose messages, and Kanna said that they don't know, but they do sample, and sampling looks good.

Big Analytics

Summarized by Michael Armbrust (marmbrus@cs.berkeley.edu)

Big Data at eBay

Tom Fastner, eBay

There are a number of important use cases for analytics over big data at eBay, spanning daily decisions such as A/B testing for experiences or treatments on ebay.com all the way to supporting long-term and multi-step programs such as the buyer protection plan. Tom Fastner described the architecture of their system and some of the challenges they have experienced operating at such a large scale (50+ TB/day of new data and 100+ PB/day processed by 7,500+ users and analysts).

Analytics at eBay is supported by three separate platforms, each with its own strengths but with some common capabilities. At the high end they run EDW (Enterprise Data Warehouse) systems based on Teradata for all transactional data, sharing it with a wide user base and supporting >500 concurrent requests per minute. For the application logs and other structured or semi-structured data, they use a low-end enterprise-class Teradata system. The world's largest Teradata installation (256 nodes, 36 PB of spinning disks able to hold 84 PB of raw data with compression) is supporting use cases on very large, but still structured, data. This platform is called Singularity. The dominating data use today is user behavior information. It also serves as a DR for the EDW data, as most of that data is required to be joined to the user behavior data for analytics.

The ability to easily work with semi-structured data is important for several reasons. First, the use of semi-struc-

tured data greatly simplifies the process of modeling the data and results in a system that is less vulnerable to changes. Additionally, the resulting de-normalization of the data can result in improved performance, as the data is already joined. Singularity enables processing over this semi-structured data by providing developers with SQL functions that extract individual items and sequences from the key/value pairs stored in a given row.

The final platform of their data analysis system is a Hadoop cluster running on 500 commodity nodes, used primarily for structuring unstructured data and for finding patterns that are difficult to express in SQL.

There is no silver bullet to cover all forms of analytics on a single platform. Integration across the three platforms is key. eBay deployed a self-service data shipping tool and is working on a transparent bridge between Teradata and Hadoop.

Mike Caruso asked if they had ever compared performance. It is not worth the effort; Hadoop is cheaper but less efficient than Teradata or EDW. Stephen Revilak (University of Massachusetts) asked how big their DBA team was. Tom said they had four DBAs and an offshore support contract.

Cosmos: Big Data and Big Challenges

Ed Harris, Microsoft

Ed Harris presented Cosmos, a multi-petabyte storage and query execution system. Used in Microsoft's Online Service Division, Cosmos is designed for large-scale back-end computation, such as parsing data from Web crawls, processing search logs, and analyzing clickstream data. Cosmos is run as a service within Microsoft; users provide the data and queries to be run, without having to worry about the underlying infrastructure. At a high level, Cosmos is broken into three major layers: storage, execution, and the SCOPE language.

The storage layer is organized around the concept of *extents*. An extent is an immutable block of data, up to 2 GB in size. The storage layer automatically handles compression and ensures that each extent is replicated to three different extent nodes for fault tolerance. Multiple extents are concatenated to form a *stream*, and the storage layer is also responsible for maintaining the namespaces of available streams.

On top of the storage layer, the execution engine is responsible for taking a parallel execution plan and finding computers to perform the work. For better performance, the system ensures that computation is co-located with data when possible. The execution model is based on Dryad, which is similar to MapReduce but more flexible, since it allows the expression of arbitrary DAGs. The execution engine, by managing failures and restarting computation as needed, also shields the developer from some of the flakiness inherent in running jobs on large clusters of commodity machines.

Finally, at the top of the stack is the SCOPE language. Influenced heavily by SQL and relational algebra, SCOPE provides developers with a declarative language for manipulating data using a SQL-like language extended with C# expressions. The SCOPE optimizer, which is based on the optimizer found in Microsoft SQL Server, decides the best way to parallelize the computation while minimizing data movement.

Since Cosmos is a hosted service, it's important to allocate resources fairly among the system's many users. This is accomplished by defining the notion of a virtual cluster (VC). Each VC has a guaranteed capacity, but can also take advantage of idle capacity in other VCs. Within any given VC the cost model is captured in a queue of work (with priority).

Harumi Kuno from HP Labs asked Ed to elaborate on how they divide cluster resources. Ed responded that each VC is provided with tokens that represent some amount of processing cores, I/O bandwidth, and memory. Mike Caruso asked if they do any migration of data, and Ed said that they have, because they bring up or shut down clusters.

Scal(a)ing up Big Graph Analytics

Tyson Condie, Yahoo! Research

Tyson Condie presented ScalOps, an embedded domain-specific language in the Scala programming language designed for running machine-learning algorithms over big data. ScalOps expands on current systems such as MapReduce and Spark by providing a higher-level language based on relational algebra that natively supports successive iteration over the same data.

A motivating example for their system is performing spam classification for Yahoo! Mail. Their first prototype used Pig to extract labels and generate a training set which was then used to train a model using sequential code. This code was executed repeatedly until a satisfactory model was found. Unfortunately, this process was suboptimal, for several reasons. First, since their tools did not natively support the iteration, they needed to construct a fractured workflow using Oozie. Second, the sub-sampling required to fit the data on a single machine hurt the accuracy of the classifier. Finally, copying data to a single machine can be very slow.

An obvious improvement is to parallelize the training algorithm. This, however, does not fit nicely into the MapReduce model. Thus, real-world implementations often involve using fake mappers that cross-communicate, eliminating many of the fault-tolerance benefits of the MapReduce model. While systems like Spark provide an improvement over such practices, by allowing users to explicitly cache data for subsequent iterations, explicit caching is a point-solution that limits opportunities for optimization. In contrast, ScalOps is a Scala DSL that is capable of capturing the entire analytic pipeline. It supports Pig Latin and has a looping construct

that can efficiently capture iteration. It runs on top of the HyracksML + Algebricks runtime, which provides the system with a relational optimizer and a data-parallel runtime.

Ed Harris (Microsoft) asked how they knew when iteration for a given algorithm was complete. Tyson replied that for global models the UDF would specify completion, and for local models computation terminates when all messages stop. Mike Stonebraker asked why they didn't use R, given its popularity with analysts. Nothing in their system precludes the use of R UDFs. Mike Caruso asked how opaque UDFs are and if this is a problem for optimization. There is no visibility into the UDFs, but the looping construct can look at the underlying AST and perform algebraic optimizations.

Consistency Revisited

Summarized by Aaron Elmore (aelmore@cs.ucsb.edu)

Eventually Consistent Is Eventually Not Enough

Mehul Shah, HP

In an analysis of eventual consistency, Mehul Shah shared experiences and insights with building a large distributed key-value store at HP. Some applications need scalable solutions to support high availability and globally distributed data. Traditional DBMSes have limited scalability, due to their consistency requirements, resulting in the creation of NoSQL databases that dropped ACID and traditional models to achieve scale. This equates to traditional databases providing CP and NoSQL databases providing AP. Eventual consistency then becomes the standard tool to enable availability in a distributed environment. There are two myths about NoSQL today: eventual consistency is enough, and adding stronger consistency later is easy.

Shah described a database built at HP as a geo-distributed, highly available, large object store that supports a large user base and versioned keys by use of timestamps. Conceptually the database is similar to S3, with unique accounts owning multiple buckets, and each bucket having a unique name and containing many objects. Buckets have unique ownership and can be shared with other users via bucket permissions. With this context, two partitioned users attempting to concurrently create the same bucket is a conflict simplified by strong consistency, whereas eventual consistency for metadata operations, such as bucket creation and deletion, could result in a user viewing unowned objects.

To facilitate strongly consistent operations and prevent undesirable situations, the Atomic Conditional Update (ACU) was introduced as a primitive for achieving consistency. Multi-key and atomic get/test/put are operations that ACU needed to satisfy with a single RPC call. Pat Helland asked whether this was effectively concurrency control, or a strongly consistent tool that can be used for optimistic concurrency control. Shah answered that larger transactions could be built out of ACU primitives if needed.

Not all operations need the strong consistency of ACU, so applications can mix strong and weak consistency operations for the same data store. This introduces subtle interactions, such as weakly consistent operations that are not serialized against strongly consistent operations. Developers are not used to thinking about these interactions, and that typically results in workarounds in higher layers. Armando Fox (UC Berkeley) asked if the operations are not serializable, due to core operations checking different targets. Shah answered that they are serializable, because a read-write dependency exists between the operations. If they were strongly consistent operations, they would be serialized by the system.

With the assumption that partitions will occur, CAP presents a choice between consistency and availability. However, the terms in CAP are not crystallized. Consistency could include notions of recency, isolation, or integrity. Availability could encompass uptime, latency, or performance. Partition toleration could be supporting a single node or a minority partition. Shahed claim that CAP is not a theorem to be applied, but more of a principle. With the many semantics that exists for consistency and availability, an ideal single system should support various consistency options that span a spectrum from consistency (transactions) to availability (eventual consistency). This was likened to isolation levels, which are easy to understand, configurable, and compatible.

Several options exist when adding consistency to a weakly consistent system. Layering services, coordinated components, and an integrated approach are techniques to provide a consistency primitive, such as ACU, to a weakly consistent database. The integrated approach still requires insight into mixed consistency operations, but these complexities are abstracted from application developers. Shah said that if you are starting over, your system design would benefit from relaxing a strongly consistent core rather than strengthening a weakly consistent core. Eventual consistency is required and, at the same time, is not enough, and now is the time to rigorously examine our understanding of consistency.

Flexible OLTP Data Models in the Future

Jags Ramnarayan, VMware

Jags Ramnarayan presented his view of the future of OLTP databases. He noted the high demand that exists currently for databases that can support low latency, predictable performance, graceful handling of large spikes in load, big data support, and in-memory operations on commodity hardware. Data input is increasingly trending toward streaming and bi-temporal behavior. Additionally, rapid application development requires a more flexible schema to support frequent changes. Most significantly, while a single database

instance is ACID, rarely does ACID hold for enterprise-wide operations. This results in data silos and duplication across databases, so enterprises must live with cleaning and de-duplication of data. Jags concluded that people actually do not want ACID but, rather, deterministic outcomes.

Having outlined trends, Jags provided a brief overview of VMware's GemFire and the similar SQL-interfaced SQLFire. GemFire is a highly concurrent, low-latency, in-memory and distributed key-value data store. Keys and indexes are stored in memory, with persistence of the data handled by compressed rolling logs. Tables can be partitioned or replicated, with replicas acting as active-active for reads, but using serialized writes by a single master for any given key. Distributed transactions are supported, but effort is undertaken to prune queries to a single partition for co-located transactions. Jags mentioned that GemFire supports asynchronous WAN replication and a framework for read-through, write-through, and write-behind operations; however, no details were given.

While hash partitioning typically provides uniform load balancing, databases should exploit OLTP characteristics to go beyond key-based partitioning. Jags made a key observation: the number of entities typically grows, not the size of each entity. Additionally, access is typically restricted to only a few entities. If related entities can be grouped, they can be co-located and thus minimize the number of distributed transactions. To build entity groups, compounded primary keys should be constructed, using foreign keys to capture relationships between entities. Grouping will largely prune operations to single-entity groups that are co-located, allowing for scalable cluster sizes, transactional write sets on entity groups, serializabilty for entity groups, and joins within a group. This does not eliminate distributed transactions across entity groups, since access pattern complexity invariably goes beyond grouping semantics. Despite the promise of hashed keys and grouping, hotspots and complex queries create difficult scenarios for "partition aware" designs. Smart replication of reference data, which is frequently joined with partitioned data, can help with this.

Looking forward and beyond the traditional SQL models, Jags described a *polyglot* OLTP database. This multi-purpose data store should support (1) continuously changing, complex object graphs, (2) structured transactional data, and (3) flexible data models, such as JSON.

Inconsistency and Outconsistency

Shel Finkelstein, SAP, and Pat Helland

Shel Finkelstein's presentation focused on approaches to handle views based on inconsistent data sources. He began with a quote by F. Scott Fitzgerald, "The test of a first-rate intelligence is the ability to hold two opposed ideas in the mind at the same time, and still retain the ability to function." Similarly, the goal for database systems should be to maintain functionality despite having potentially inconsistent data sources. To understand whether data is inconsistent, a complete view of the data's context may be required. A set of weather measurements without location or time may seem like inconsistent data but is simply lacking event details and provenance. After challenging notions of consistency, Shel provided Jim Gray's definition of consistency: "A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state."

Data inconsistencies can occur for a variety of reasons. First, inconsistencies can derive from integrity constraint violations, such as impossible address information, corrupted-entity foreign relations, violated business rules, or domain constraints. Second, logical impossibilities can occur. This can include unanticipated data unknowingly being transformed, incomplete data, or real-world data contradictions, such as a location changing that clearly could not relocate. Third, replication issues include asynchronous data feed corruption and relaxed consistency models for replication protocols. Fourth, many databases run with read committed as the default isolation level, which does not guarantee serializability and can result in data inconsistencies. Normann and Ostby's *A Theoretical Study of Snapshot Isolation* in EDBT 2010 was given as a reference for inconsistencies that can arise even when using snapshot isolation.

In an ideal world inconsistencies in data would not exist, but databases reside in the real world and need to handle inconsistent data sources. Every application operates with assumptions about the consistency of data, and disjoint applications can make different assumptions about the same inconsistent data source. Shel introduced the concept of *outconsistency*, which involves providing an "outwardly consistent view of the data" and guidelines for how applications should operate on inconsistent sources. This provides a regimen that enables different applications to operate on the same data source with some understanding of methods for dealing with inconsistencies.

Approaches for addressing outconsistency were defined as the following techniques, which are likely to be used in combination. *Preventing inconsistent data* provides an identical view to the data, relying on approaches such as strong integrity constraints, checking business rules, and utilizing "transactional intent" (CIDR 2011) to prevent inconsistencies at the source. *Tolerating inconsistent data* utilizes expected inconsistencies to transform data for the outward view. *Ignoring inconsistent data* filters the outward view to include only data consistent for the purposes of the application. *Fixing inconsistent data* requires the application to take an active role in correcting the inconsistencies in the source

data. For each approach a set of challenges were discussed. A final claim was that all data, and subsequently transaction processing, consists only of events, reports, and decisions. Transactions and consistency should be discussed in this context. The work presented is just the beginning of examining the fluid relationship between applications and data.

Graefe Goetz wondered whether "kicking the can down the road" really means "eventually consistent"? Shel countered that "kicking the can down the road" means that something else deals with it. It can be data cleansing applications, services with alerts, interpolation and extrapolation, or renewal processes, such as SAP's APO. Roger Bamford (Oracle) asked if Shel would consider compensating transactions, and Shel said that this fits into this category. Shah said that he had experience with fixing these problems, and that it comes down to cost, earlier versus later. Can you comment a little on costs? Shel replied that the tradeoff has to do with coping with inconsistencies or fixing them. Mike Caruso mused that you could have an outconsistency in one system that would be an inconsistency in a second. Shel concluded by asking the audience to consider whether his factoring is correct, and if it is whether we should write applications based on it.

It May Be Fast, But Is It Right?

Summarized by Yingyi Bu (yingyib@ics.uci.edu)

Debugging Designs

Chris Newcombe, Amazon

Chris Newcombe presented a model checking–like approach for finding bugs at the design stage. He used Stoica et al.'s 2001 Chord paper as an example. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications" is one of the most cited computer science papers (8,966 cites as of Nov. 11, 2011). However, Pamela Zave from AT&T Research recently found eight major defects in the Chord ring membership protocol, using exhaustively testable pseudo-code (written using Alloy). The testable pseudo-code is remarkably simple. The example reveals that even top work done by the best people and reviewed by very smart peers can still have bugs! In particular, those systems bundling concurrency control, recoverability, failure handling, and business logic together are very hard to debug. However, test tools can help.

Chris proposed that pseudo-code should be written in a support tool rather than only in design documents, and people should use the tool to do exhaustive testing on the pseudo-code so that bugs could be found at the design phase. He proposed that TLA+ and PlusCal should be the pseudo-code language, and he used Michael J. Cahill's SIGMOD 2008 paper "Serializable Isolation for Snapshot Databases" as a running example to show how to get testable pseudo-code from the pseudo-code in the paper. Finally, Chris showed that the TLA tool can automatically check Cahill's algorithm. Chris also suggested that the audience read Lamport's book *Specifying Systems* as well as *TLA+ Hyperbook* (http://research.microsoft.com/en-us/um/people/lamport/tla/hyperbook.html).

Margo Seltzer said that the testable pseudo-code actually is a specification, and TLA+ could be thought of as a specification language. Chris replied that the word "formal" is like death for many people, and he steers away from using that word. He claimed to be an escaped video game programmer who has never proven anything in his life. Mike Caruso asked if Lamport's TLA+ can generate code to the state machine level, and Chris replied that Lamport designed his tool to be very expressive declaratively, but it cannot be used to compute. Adrian Cockcroft wondered why he couldn't find Lamport's book at Amazon, and Ernie Cohen replied that the PDF is available for free.

Verifying Real-World Transaction-Processing Code with Microsoft VCC

Ernie Cohen, Microsoft

Ernie Cohen argued that testing sucks, and, instead, deductive verification should be widely used in production software development. He proposed that programmers should be able to write contracts such as pre-conditions, post-conditions, and invariants in their code. Therefore, verification could be done in a program-centric way. Ernie clarified that the cost of deductive verification should be comparable to complete functional testing.

After giving the high-level vision, Ernie briefly introduced VCC (Verified Concurrent C), which was developed by his group. VCC allows programmers to annotate their original C code with contracts. Then he did a live demo to show how VCC can find bugs in a C binary search program. Ernie added several preconditions and invariants as annotations to the code, and then bugs such as race conditions, buffer overflow, and value overflow were quickly caught. After the live demo, Ernie illustrated several useful constructs in VCC, such as data invariants, ownership, and ghost data and code. Data invariants are invariants on objects and can be defined as part of type declarations. Ownership is mostly used for specifying the contracts of concurrent reads/writes. Ghost data usually represent abstract states, while ghost code is actually executable contract and only run at verification time. Ernie also showed how to add annotations such as invariants and ghosts to make a piece of lock-free, optimistic, multiversioned transaction processing code verifiable. Finally, Ernie said that they should add prophecy support in VCC in order to verify properties such as "whether a timestamp obtained from the DB will be its final timestamp."

Armando Fox (UCB) asked if VCC can handle runtime polymorphism. Ernie pointed out the C includes runtime polymorphism, such as function pointers. Armando asked if VCC works for languages other than C. They may port it to C++.

Data Without Provenance Is Like a Day Without Sunshine

Margo Seltzer, Harvard University

Margo Seltzer argued that provenance is playing an increasingly important role in computer systems. It is the "how, when, why" metadata about the data. She used Wikipedia revision history to illustrate provenance: by looking at the editors historically, one can gain someconfidence about the Wikipedia content. Provenance can come from instruments, application software, system software, or software tools. Provenance reminds people what has happened and gives people a way to understand why something happened. Margo pointed out that nowadays provenance is usually managed manually, implied, embedded, or part of a workflow system.

Margo emphasized that provenance is everywhere! Every day, people ask questions such as "Why does Facebook recommends this ad to me?" "Where does this file come from?" "What did the customer do before she hit this bug?" Margo advocates that provenance be built into every system in a layered way. The key concept there is that each layer collects provenance and each layer associates its provenance objects with both upper- and lower-layer provenance objects. The example systems Margo's group has built include a provenance-aware storage system, simple provenance in PostgreSQL, and a provenance-aware Python workflow engine.

Rusty wondered why the person who wrote an algorithm couldn't supply provenance, and Margo said she wants the algorithm to include the generation of the provenance, so that the information generated can be used to improve the algorithm. Pat Helland said that machine learning is like Mulligan stew, it's "ginormous," and Margo agreed. But Margo said she still wants everything, which is why disk vendors love her. Jim Waldo (Harvard) said that for non-disk vendors, transporting all the provenance data will not be wonderful (or cheap). Margo pointed out that this is HPTS, so with a provenance handle you can make distributed queries on replicated stores wherever you want. Margo's group has worked on how much provenance you are likely to want. Jim asked if this was a ratio of provenance to data. Margo answered that it depends. Someone asked if provenance was like using CVS, and Margo replied that CVS is "the poor man's provenance."

Trusting the Cloud

Summarized by Steve Revilak (srevilak@cs.umb.edu)

Clouds & Condos

Pat Helland

Pat asked, "What can condos teach us about cloud computing?" Quite a few things! Condos place constraints on living environments, but they also provide benefits: for example, most repair and maintenance work is taken care of for you. One can take advantage of these benefits, as long as there's a willingness to live within the constraints. Similar trends can be seen with other types of buildings. Retail space and office parks are built with a notion of how the space will be used but without knowledge of who will be using them. This allows building developers to support a wide variety of tenants; they build to a common set of usage patterns and impose a few constraints on what the building tenants can do.

Cloud computing can develop along similar lines. Cloud computing can provide basic services, such as stateless request processing, session management, load balancing, provisioning, and scalability. These services may not fit the needs of every conceivable cloud user, but they will fit the needs of *most* cloud users. We can design cloud computing systems according to common patterns of use, just as we do for buildings, even if we don't know who the cloud's users will be.

Laws and norms governing landlord-tenant relationships have evolved over time and work to the advantage of both parties. Pat believes we could benefit from a set of common rules that govern cloud providers and cloud users. Such rules would provide fair treatment to users and offer protection to service providers.

In summary, our relationship with buildings has changed over time. As we've done with buildings, we need to develop usage models and constraints, as well as rights and responsibilities, governing the cloud computing environment.

Mike Caruso pointed out that customers will need to tell the cloud providers what they want/need. Pat agreed, but said that we already know some patterns. Someone pointed out that it took many years for landlord-tenant law to evolve. Armando Fox said that he already uses Heroku, and it provides many of the things he wants.

Go Fast and Don't Break Things: Ensuring Quality in the Cloud

Scott Hansma, Salesforce.com

Salesforce began life as a CRM application. It has evolved into a full-blown development platform that conducts 575 million transactions per day. All Salesforce customers run in a hosted environment, and all customers use the same

version of Salesforce's software. This scenario makes quality control extremely important; upgrades must work for all users, and upgrades cannot break functionality users have come to depend upon.

Salesforce is intensely focused on software quality, and this commitment to quality manifests itself in several ways. Salesforce uses a continuous integration (CI) system to test changes as they are committed to their source code repository. This CI system runs 150,000+ tests in parallel across many machines, and it will do binary searches across revision history to pinpoint the precise check-in that caused a test to fail. Developers do not get off easy—once the CI system has identified an offending check-in, it will open a bug for the developer to address the problem.

Salesforce allows customers to customize their applications with a programming language called APEX. As a best practice, Salesforce requires customers to test their APEX code prior to deployment. These customer-written tests provide an excellent way to regress new releases; Salesforce can run customer-written tests against new releases to identify problems prior to deployment. (Salesforce developers are given access to information about failing customer-written tests, but they are not given access to the underlying customer data.)

Finally, Salesforce maintains a Web site (http://trust.salesforce.com) where they publish availability metrics and service announcements. The company believes that this transparency—publishing their uptime metrics—helps to promote user confidence in the platform and keep the company focused on quality.

A Non-Proprietary Social Internet

Monica Lam, Stanford University

Cloud computing offers a long list of benefits, but that list does not always include privacy. Take Facebook as an example: Facebook provides a great user experience, and a great platform for application development. But Facebook is also a social intranet—all interactions pass through Facebook's servers, and Facebook controls access to user data. Monica believes that social networking should be more like email. Two people can exchange email without having to use the same email provider, so, why should two users need to use the same social network provider in order to have social interactions?

Monica presented an application called Musubi, to demonstrate how open social networking could work. Musubi is a mobile application that runs on the Android platform and permits peer-to-peer social networking. Social networks are created through the users' address book and do not require a central service provider. This platform preserves privacy by eliminating the need for a central provider (all of your data lives on your mobile phone) and by encrypting communications. During the talk, Monica set up a social networking group for HPTS attendees; several people joined and began exchanging messages.

Monica also demonstrated how smartphone applications could be turned into collaborative social applications. She presented an application called We Paint, which is a collaborative drawing application.

Stanford has conducted several usability studies with Musubi. The reactions have varied by age group. Some adults believe that this is the future of social networking. College students were indifferent; they preferred to use Facebook and found nothing new and attractive in Musubi. Elementary school students were the most receptive; they thought Musubi was "awesome."

An attendee who worked at Facebook was very upset with Monica for suggesting that Facebook might sell user data. Monica said that people should be free to use Facebook if they want to, but she also believes that users should have the freedom to use different social networking platforms if they choose to do so.

Debate Panel: Scale Up vs. Scale Out

Summarized by Pinar Tozun (pinar.tozun@epfl.ch)

Panelists: Michael Stonebraker, MIT; Mark Callaghan, Facebook; Michael Cahill, Wired Tiger; Andy Gross, Basho

The debate panel of this year's HPTS was about whether to focus on scaling up, utilizing a single node in the system with useful work as much as possible, or scaling out, increasing the performance of the system by adding more machines. A node was initially defined as a single processor by the panel but during the panel it was sometimes also referred to as a single multiprocessor machine. The panel chairs, Margo Seltzer (Harvard) and Natassa Ailamaki (EPFL), had a slider where 0% indicated total focus on scaling up and 100% indicated focusing only on scaling out. The chairs asked the panelists where on this slider they stood while building their systems.

Michael Stonebraker chose 15% on the slider. Focus on using all the cores available in your processor as efficiently as possible first, but also think about how to scale out: unless you have both in your database today, you are not going to be successful. One size does not fit all. Different markets should optimize their systems for their needs. In OLAP (online analytical processing), for example, a column-store beats a row-store, and you need clever overhead cleanup and, in scientific databases, array-based designs. He emphasized getting rid of the shared-data structures (buffer pool, B-trees, etc.),

locking, and latching bottlenecks in database management systems in order to scale up to many cores in a node, as they do in their VoltDB-related work. He claimed that Facebook could have done what they are doing with a 4000-machine cluster with only 40 machines if they had been using VoltDB. He believes that in the next 20 years: there will be around five gigantic public cloud vendors, so we should trust the cloud and try to adapt our systems to its environment.

Mark Callaghan picked 90%. He leads the MySQL engineering team in Facebook; they run the MySQL at Web scale across many machines. He tried to answer why they were using that many machines to handle Facebook's workload. They know they can never be working on a single node with Facebook's enormous scale, so as long as the software is efficient enough, they focus on how to scale out rather than how to scale up. He believes their market requires a focus on scaling out. They are mostly I/O bound and not CPU bound, and he does not think using database designs focused on in-memory databases, such as VoltDB, will help them. To have better IOPS and provide lower latency to their customers they need to buy more machines and think about how to scale out.

Andy Gross put his choice at 70% , arguing for focusing on both but favoring scaling-out. His background is in distributed systems; he likes Dynamo-like systems. Naturally, he said, he is interested in more than one machine. He also pointed out different ways of scaling out and scaling up: not just thinking how to use one node or more machines better but also dealing with how to exploit different technologies, such as SSDs, GPUs, and FPGAs. He also said that some people who do not have the choice of using specialized solutions need to use general-purpose products. Public cloud environments such as EC2 are good spaces as general-purpose solutions, and they also try to address problems related to power and energy consumption for general-purpose systems.

Michael Cahill chose 0%. Working inside the storage engine, he argued that we need to revisit the assumptions we make in storage engines to ensure serializable isolation among transactions. We have to find non-blocking algorithms to get the best out of a single processor. People mostly focus on scaling out across multiple machines, but he wants to make contributions within a single node. People should design their software in a way that it will work well on new hardware. He said that big companies such as Facebook have the luxury to focus on scaling out, but smaller companies should focus on the storage engine first and do their best to scale up there.

Natassa Ailamaki asked for an example of a technique that is good for scaling up but not good at all for scaling out. Michael Cahill answered that optimistic concurrency control is definitely a good technique for scaling up in a single node, but since it is hard to coordinate across nodes, it is not good for scaling out. Natassa wondered whether buying more machines to have more IOPS is a waste of machines and power. Mark said if they tried to get rid of the disk and be in-memory they would need 10 times as many machines as they have now. Michael Stonebraker said that if they are I/O bound, they should use what is optimal for the data-warehouse market, have column-stores with better compression, and reduce their I/O load. Mark argued that having compression does not reduce the number of IOPS you need linearly, and it does not solve the random I/O problem. On the other hand, Margo Seltzer opted for focusing on scaling up first: if you have a system that cannot saturate the memory and CPUs you have, then you have a badly built system.

All the panelists thought open source products were great. However, Michael Cahill said that his team cannot maintain an open source product for their own needs. Open source products such as MySQL also might end up having so many versions around that it will not be clear which one should be used. However, Mark Callaghan argued for MySQL, since there is one main MySQL version and only two or three other versions to choose from.

Someone asked, If you have a 160-core machine, how is scaling up within that machine different from scaling out? Michael Cahill said it is the same, but there are more failure cases when you have more machines. Andy Gross said that such a machine will make you use ideas from distributed systems in a single machine. Another person asked what academic researchers should focus on. Andy Gross argued that the interesting papers are the applied ones and they are mostly about scaling out. Michael Stonebraker advised that academics should talk to real customers, understand their problems first, and then try to solve them in their research. What should we do if we had non-volatile main memory? Andy Gross said that SSDs can be thought of in that way.

There was discussion about the NoSQL databases and database knobs that require DBAs. Mark Callaghan argued that NoSQL systems are good, because even though they do not focus on performance they are easier to manage, and that is what matters for some users. Andy Gross also supported NoSQL systems. On the other hand, C. Mohan (IBM) argued that NoSQL systems made users optimizers of databases, and Stonebraker argued that SQL provides an abstract layer on top of a database for their customers. Stonebraker also supported the need to get rid of as many knobs as possible so as not to be dependent on the database vendor and the DBAs. However, Armando Fox (UC Berkeley) argued that as a customer he would prefer not knowing about the database design details and that people who can tune are good if they know how to do it well.

New Age OLTP

Summarized by Pinar Tozun (pinar.tozun@epfl.ch)

All the Rules Have Changed

Michael Stonebraker, VoltDB

Michael Stonebraker started his talk by categorizing the OLTP market: OldSQL, NoSQL, and NewSQL. OldSQL represents the major RDBMS vendors, which Stonebraker called the "elephants." They are disk-based and use ARIES (Algorithms for Recovery and Isolation Exploiting Semantics), dynamic record-level locking, and latching mainly to ensure ACID properties. On the other hand, NoSQL is supported by around 75 companies today and favors leaving SQL and ACID. Finally, NewSQL suggests not leaving SQL and ACID but changing the architecture of the traditional DBMS used by OldSQL.

He pointed to an earlier study that shows that in OldSQL-like databases only 4% of the total CPU cycles go to useful work, with the remainder almost evenly shared by latching, locking, logging, and buffer pool management. He argued we cannot benefit much from trying to optimize those architectures, so a redesign of the DBMS architecture without compromising SQL and ACID is needed to increase the percentage of the useful work. This is what NewSQL databases are trying to achieve. Some examples of the NewSQL movement are VoltDB, NuoDB, Clustrix, and Akiban.

He focused on his own work on VoltDB in this field. VoltDB is a shared-nothing database which has hash or range partitioning on the data. It gets rid of the buffer pool by deploying an in-memory database design, because most of the OLTP databases today can fit in the aggregate main memory of few commodity machines. Main memory is statically divided per core, and there is a single partition for each core in a machine. Only a single worker thread executes transactions within a partition. There are no shared data structures, hence, no need for locking and latching. K-safety measures are used by replication of the data to ensure durability. VoltDB uses stored procedures for transactions, but on-the-fly compilation of ad hoc queries is possible with minor overhead. Speculative execution can be used for distributed transactions to reduce their overhead, although VoltDB does not have such a technique yet. Overall, in terms of performance, the NewSQL database VoltDB achieves a 60x improvement compared to an OldSQL vendor, offers an 8x improvement over Cassandra, which belongs to the NoSQL movement, and has the same performance as Memcached. Stonebraker argues that VoltDB is good for both scaling up and scaling out.

Stonebraker also discussed some of the recent techniques they introduced in VoltDB and some future work. To deal with cluster-wide failures, they have continuous and asynchronous checkpointing and a log that keeps the stored procedures executed with their inputs so that they can re-execute those after a failure, starting from a checkpoint. Moreover, for the databases which are too big to fit in memory, they are investigating how to do semantic caching for the current working set of the workload. In addition, they work on WAN replication, compression, and on-the-fly reprovisioning in VoltDB.

Bruce Lindsay exclaimed that the CPU cycles breakdown on the presentation was not true for the big OldSQL vendors. Stonebraker replied that they cannot do that measurement with the products of major vendors, since the source code is not available, and Oracle does not permit benchmarking of its products. C. Mohan (IBM) asked whether they support partial rollbacks in VoltDB. Stonebraker said no. Bruce also wondered how VoltDB will do semantic caching if indexes are not subsetted.

HyPer-sonic: Combined Transaction AND Query Processing

Thomas Neuman, T.U. Munich

Neuman described a main memory database system that combines the execution of OLTP and OLAP workloads, called HyPer, which they built at T.U. Munich. OLTP and OLAP workloads have different characteristics. OLTP has frequent short-running transactions that observe high data locality in accesses and require high performance with updates; OLAP has a few long-running transactions that touch a lot of database records and need to see a recent consistent state for the database. Most of today's systems are optimized to be good in either. Two separate systems are kept for each workload, and data is transferred from the OLTP side to the OLAP side with an ETL (extract, transform, load) process. However, this causes both high resource consumption and the OLAP copy to see an older version of the data.

HyPer tries to bring the query processing from OLAP workloads to an efficient OLTP system. It has an in-memory database design, wherein the data is partitioned and only a single thread operates on a partition at any time. This way they can run OLTP transactions lockless and latchless, very similar to VoltDB design. Whenever an OLAP query needs to be executed, the OLTP process is forked to create a virtual memory snapshot of the database and the query processing is done on that snapshot, which provides a fresh version of the database for query processing. Updates from the OLTP side to the database while a query is running are not reflected in the database snapshot seen by the OLAP query. This is handled efficiently because the initially forked OLAP process shares physical memory with the OLTP process. Only when there are updates from the OLTP process does a copy-on-update take place and only on the updated memory locations,

to separate the update done by the OLTP (parent) process from the OLAP (child) process.

They use a datacentric query execution model rather than the iterator model. There is a pipeline of operators for a query. They have a producer and consumer interface: the former pushes the data to the current operator and the latter accepts the data and pushes it further up to the next operator. The functions are generated in assembly code at compile time using LLVM, which is fast when you want on-demand compilation and creates portable code. Moreover, it achieves better data and instruction locality than the iterator model.

To evaluate HyPer they designed the CH-benchmark. This mainly keeps TPC-C database schema and its transactions with some additional tables from TPC-H schema, and it converts TPC-H queries to be compatible with this schema. The performance numbers are good, although the memory price might be high unless the working set is fairly small.

Bruce Lindsey wondered how they could run serialized transactions. It's easy if the transactions are not touching the same data. Mohan asked how they knew this statically at runtime; Neuman said, Through partitioning. Mike Ubell asked how they could get copying without copying all the data. They used the MMU to detect when a write would occur and created a copy only at that time. Russell Sears (Yahoo!) commented on the use of LLVM to create queries, saying that code generators blow away the instruction cache.

Scaling Out with Meld

Phil Bernstein, Microsoft

Phil Bernstein presented a database design where a shared binary-tree log is the database. The log supports multiversion concurrency control, and the most recently used parts of it are cached in each server's memory. In this design there is no need for cross-talk between the servers, so the design is very suitable for scaling out without dealing with the burdens of partitioning, which is the more common technique for scaling-out today.

Each server has its local (partial) copy of the log in-memory, and it has the last committed database state. Transactions executing on these servers append their intention log records to these local copies of the log. At the same time, the meld operation takes place at each server that processes the log in log order to see whether there are any transactions that conflict with each other. Depending on this process, the meld operation decides which transactions to commit or abort. If a transaction is to be committed, then the meld merges its updates (intention log records) to the database state. The meld operation basically performs optimistic concurrency control for the transactions.

Even though such a design is good for scaling out, the simple way of doing it has bottlenecks. Optimistic concurrency control can hinder performance severely if the conflict rates are high. However, this is dependent on the application. Reading from and writing to the log might be bottlenecks, but these can improve in the future with technology trends. On the other hand, the meld operation, as a single-threaded operation that reads many entries in the log, might become a huge bottleneck, because the speed of a single processor no longer improves, so the meld operation should be optimized first.

The main idea was for the meld process to have a lot fewer log records to check for conflicts for a transaction, by storing more information about the transaction in each transaction's intention. This is mainly done by keeping version numbers for each node of the binary tree.

Mohan asked if they broadcast the log. Everybody has to read it. Margo Seltzer noticed a potential conflict on a slide, and Bernstein responded that she had found a bug. Someone asked how they handle scan operations. They currently do not support scans.

Mobility Trends & Implications

Summarized by Yingyi Bu (yingyib@ics.uci.edu)

Data Management Challenges in Location Based Services

Srinivas Narayanan, Facebook

Srinivas Narayanan talked about the status and vision of location-based services in Facebook. Mobile users usually add location check-ins, upload photos, and create events. Srinivas emphasized that location is not only latitude and longitude, but also people, activities, and places. In the future, social events will be built on top of locations, and interesting applications such as social events/friends discovery on top of places will become very popular.

Srinivas listed several challenges. First, a good location search needs to address queries with either strong location bias or weak location bias, and considers rankings. Second, social queries need to scale up and scale out; currently Facebook uses a MySQL back-end but does not use complex queries. In the future, Facebook wants to support queries more complex than NoSQL queries. Third, everybody can have privacy policies for every data point, and to apply rules to each data point would be expensive. Fourth, social recommendations add a new dimension (location) to personal data, which advanced machine learning and data mining algorithms should leverage. Fifth, data quality will be challenging;the true real-world location for many data sources (which get tagged by user annotations), and crowd-sourcing or machine learning might be a solution.

Harumi Kuno (HP Labs) asked if Srinivas could say more about the index used for queries. Srinivas said you can use lots of predicates. Adam Marcus (MIT) pointed out that the information is both sensitive and has many compelling use cases, and he wondered about privacy. Srinivas said Facebook provides you with complete control over who sees your data. Mike Caruso wondered if they could use the location data to reveal where someone lives. Srinivas said that he didn't have a specific answer for Mike, and he pointed out that Facebook is not just an urban product.

Urban Data Analysis—Projects, Methods and Tools Used to Describe 21st Century Cities

Oliver Senn, MIT SENSEable City Group

Oliver Senn described several interesting projects built by their group. The Copenhagen Wheel includes sensor devices on bikes that continuously collect data from the bikes. With gathered sensor information from all bikes, the back-end system can do real-time data analysis, considering both traffic and social information. Co2go uses accelerometer traces in smartphones to estimate CO2 emissions and enables aggregated CO2 emission analysis among users. LIVE Singapore! is an enabling platform for applications that collect, combine, distribute, analyze, and visualize either historical or real-time urban data. Applications such as realtime call locations, formula one city, and raining taxis have been built on top of LIVE Singapore! (http://senseable.mit.edu/livesingapore/).

After describing those interesting urban data analysis applications, Oliver mentioned software tools they used in the project, including Matlab, R, OpenMP, MPI, Boost Graph Library, GNU Scientific Library, C++, Java, Python, awk, sed, Oracle, MySQL, and PostgreSQL. One challenge is that the project group has only a few computer science people, and everyone is sticking to some tools; thus, to integrate different components requires a lot of work. The other challenge is to understand the data set in terms of what system produced the data, what parts of the data should be included/excluded, and how inconsistency should be resolved.

Someone asked about future problems/challenges. Oliver said that one challenge is that they don't have access to huge clusters, so for certain tasks (e.g., graph analysis) they are currently restricted. Also, certain data sets (such as the LIVE Singapore data) belong to the owners (are under NDA) and cannot be exported offshore.

Mobile Data Management in the CarTel System

Sam Madden, MIT CSAIL

Sam Madden talked about problems in mobile data management and their solutions in the CarTel system. Applications of mobile data management include smart tolling insurance, urban activity monitoring, and personal medical monitoring. The volume of road sensing data is huge, and personal trajectory data is sometimes sensitive. Madden's group developed software to efficiently store and access such data while providing users control over privacy.

One subproject of CarTel is CTrack, which transforms sensed raw data to meaningful trajectories. CTrack handles cell phone signal location points with incorrect data, using probability-based estimation techniques, and pre-processes this data to visualizable road traces. The other subproject is TrajStore, a storage system for indexing and querying trajectories. TrajStore recursively divides a region into grid cells and dynamically co-locates/compresses spatially and temporally adjacent segments on disk in order to minimize disk I/Os.

Sam pointed out that there are more and more location-aware mobile devices, from a database researcher's view: cleaning, matching, filtering, visualizing, and animating mobile data at large scale is challenging.

Mehul Shah wondered why they didn't put all the data into a database. Sam replied that some of the group are machine intelligence experts trained in tools that don't look like SQL. If you push the data into SQL, they won't use it. Adam Marcus asked to what extent this is an interface problem. Sam said that some of the algorithmic issues haven't been figured out yet.

Scalability Under the Hood at Foursquare

Jorge Ortiz, Foursquare

Jorge Ortiz introduced Foursquare, a location-based social networking service provider which has general social networking utilities, games, and city guides At the launch in March 2009, Foursquare used a single-node PostgreSQL database, which served 12,000 check-ins/day and 17,000 users. At the beginning of 2010, carrying a workload of 138,000 check-ins/day and 270,000 users, the system broke trying to serve all the reads/writes on a single database. From then on Foursquare used MongoDB to serve reads but still used one PostgreSQL node to serve check-in writes. In 2011, with 2.8 million check-ins/day plus 9.4 million users, Foursquare began to use MongoDB for both reads and writes. By October 2011, there were over 13 million users and more than 4 million check-ins per day. Drawbacks of PostgreSQL include connection limits, VACUUM (free space recovery), and lack of monitoring tools; advantages of MongoDB include auto-balancing, shard routing, and synchronization.

Jorge explained that MongoDB does not shard geography queries. Therefore they use Google's s2-geometry library, which turns polygons into sets of covering tiles and turns the geography index problem into a search problem.

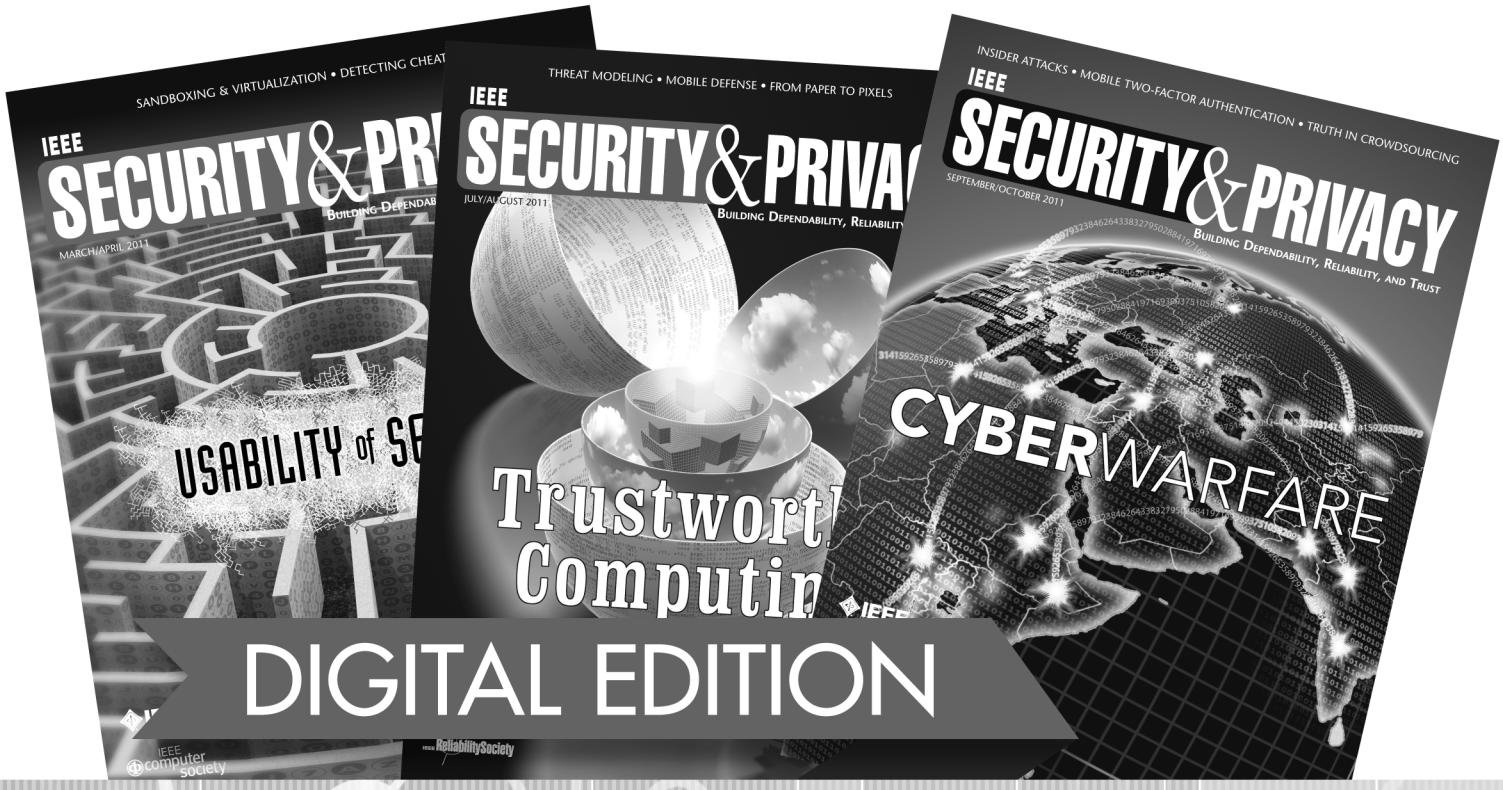
GO WHERE UNIQUE TALENTS CONVERGE

At EMC, innovative thinking is sustained through diverse perspectives. The EMC workforce is a world of more than 52,000 thought leaders working together to drive the future of cloud computing and information management solutions.

Through innovative products and services, EMC accelerates the journey to cloud computing, helping IT departments to store, manage, protect and analyze their most valuable asset – information – in a more agile, trusted and cost-efficient way.

To learn more about EMC, visit www.emc.com. To learn about working at EMC, visit www.emc.com/careers.

EMC²



DIGITAL EDITION

IEEE

SECURITY & PRIVACY

SUBSCRIBE FOR $19.95

- **Protect Your Network**
- **Further your knowledge with in-depth interviews with thought leaders**
- **Access the latest trends and peer-reviewed research anywhere, anytime**

"*IEEE Security & Privacy* is the publication of choice for great security ideas that you can put into practice immediately. No vendor nonsense, just real science made practical."

—Gary McGraw,
CTO, Cigital, and author of *Software Security* and *Exploiting Software*

www.qmags.com/SNP

usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

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

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

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES