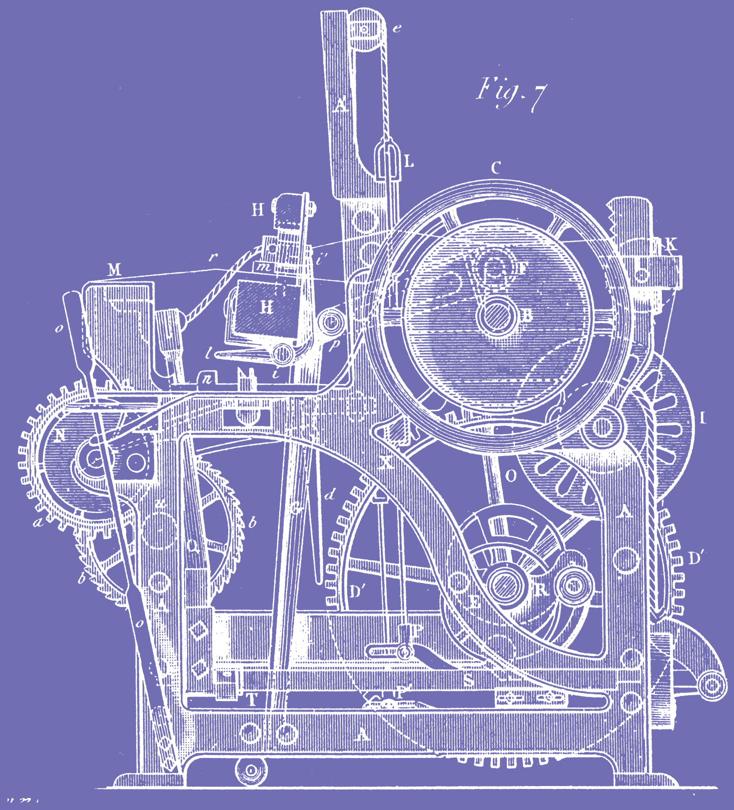
APRIL 2010 VOLUME 35 NUMBER 2

;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '10)

Co-located with NSDI '10

April 27, 2010, San Jose, CA, USA
http://www.usenix.org/leet10

2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN '10)

Co-located with NSDI '10

April 27, 2010, San Jose, CA, USA
http://www.usenix.org/inmwren10

9th International Workshop on Peer-to-Peer Systems (IPTPS '10)

Co-located with NSDI '10

April 27, 2010, San Jose, CA, USA
http://www.usenix.org/iptps10

7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

April 28–30, 2010, San Jose, CA, USA
http://www.usenix.org/nsdi10

2nd USENIX Workshop on Hot Topics in Parallelism (HotPar '10)

Sponsored by USENIX in cooperation with ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, and ACM SIGARCH, and ACM SIGPLAN

June 14–15, 2010, Berkeley, CA, USA
http://www.usenix.org/hotpar10

USENIX Federated Conferences Week

June 22–25, 2010, Boston, MA, USA

2010 USENIX Annual Technical Conference (USENIX ATC '10)

http://www.usenix.org/atc10

USENIX Conference on Web Application Development (WebApps '10)

http://www.usenix.org/webapps10

3rd Workshop on Online Social Networks (WOSN 2010)

http://www.usenix.org/wosn10

2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)

http://www.usenix.org/hotcloud10

2nd Workshop on Hot Topics in Storage and File Systems (HotStorage '10)

http://www.usenix.org/hotstorage10

2010 Electronic Voting Technology Workshop/ Workshop on Trustworthy Elections (EVT/WOTE '10)

Co-located with USENIX Security '10

August 9–10, 2010, Washington, DC, USA
http://www.usenix.org/evtwote10

3rd Workshop on Cyber Security Experimentation and Test (CSET '10)

Co-located with USENIX Security '10

August 9, 2010, Washington, DC, USA
http://www.usenix.org/cset10
Submissions due: May 24, 2010

2010 Workshop on Collaborative Methods for Security and Privacy (CollSec '10)

Co-located with USENIX Security '10 and sponsored by USENIX and Deutsche Telekom

August 10, 2010, Washington, DC, USA
http://www.usenix.org/collsec10

1st USENIX Workshop on Health Security and Privacy (HealthSec '10)

Co-located with USENIX Security '10

August 10, 2010, Washington, DC, USA
http://www.usenix.org/healthsec10

5th USENIX Workshop on Hot Topics in Security (HotSec '10)

Co-located with USENIX Security '10

August 10, 2010, Washington, DC, USA
http://www.usenix.org/hotsec10
Submissions due: May 3, 2010

19th USENIX Security Symposium (USENIX Security '10)

August 11–13, 2010, Washington, DC, USA
http://www.usenix.org/sec10

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

contents

;login:

VOL. 35, #2, APRIL 2010

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Casey Henderson
Jane-Ellen Long
Jennifer Peterson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

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

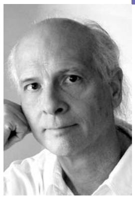
$90 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $125 per year.

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.

RIK FARROW

musings



Rik is the Editor of *;login:*.

rik@usenix.org

I'VE BEEN HAVING A DIFFICULT TIME keeping my head out of the clouds. Not that I've been flying, or even daydreaming much. It's just that some interesting clouds popped into the foreground recently, and I am finding it hard not to pay attention.

Intel announced its Single-chip Cloud Computer [1] back on December 2, right about the time I was working on my previous column. Unlike Intel's earlier Teraflops project [2], the SCC seemed like something I had once dreamed about, as well as a practical experiment that researchers might actually want to work with.

The Teraflops project was a proof-of-concept: 80 floating-point processors tiled on a chip. While this was cool, it wasn't particularly useful and seemed more like a publicity stunt. But the Teraflops Chip did prove to Intel that it was possible to put many cores on a single chip and have them work.

The SCC also sounded like some PR at first, but that is probably because it has the word "cloud" in its name. It seems as though everything must include "cloud" for marketing purposes, even AV software [3], so ignoring yet another cloud announcement makes perfect sense. One of the OS researchers I contacted about the SCC just blew it off at first for that reason. Yet the SCC represents a likely future design for manycore CPUs.

Distributed Systems

Using a network of processors goes back to the dawn of computing. Even the tube-based IBM 709s had channel I/O processors [4], programmable processors subordinate to the main CPU that handled I/O tasks. Using channel I/O makes a lot of sense, as I/O is slow, and potentially a lot of work could be done if the main CPU wasn't waiting on I/O or, worse, copying data between I/O and memory.

Channel I/O even appeared, briefly, in microprocessor-based systems in the early '80s. Morrow Designs had a hard-disk controller that worked just like a channel controller, complete with the ability to execute programs stored in main memory and copy data between memory and hard drives. At the time, I thought that distributed processing would take off (1984), but Morrow was far ahead of the curve.

Distributed systems got popular with the development of various clusters, starting as early as 1970, and really taking off with the Parallel Virtual

Machine [5] software in the late ’80s and Beowulf clusters in the ’90s. The ability to use groups of heterogeneous systems as if they were a single supercomputer changed how scientific computing was done. These days, MapReduce and Hadoop are the most used systems for building large-scale clusters, sometimes composed of thousands of systems networked together.

Not Quite a Cloud

Although Intel PR conflates the SCC with cloud computing, that’s just abusing the current hot buzzword with their distributed computing design. The SCC consists of 24 dual-core x86 CPUs, each core having its own level 2 cache. The 24 dual-core CPUs each has both memory and hardware dedicated to message passing, with all the CPUs connected in a mesh network. Memory controllers sit at the edges of this network, implying the ability to have four independent memory transfers simultaneously.

Each dual-core CPU, or “tile” in Intel-speak, can run at a different frequency, and groups of four tiles can be run at reduced voltage levels, giving the chip a thermal envelope from 25 to 125 watts. Perhaps this is why Intel styles this chip a “cloud,” since, like a cloud, its computing resources can be varied on demand.

The SCC only vaguely resembles today’s clusters/clouds, which are composed of networked but complete systems. So each member of a Hadoop cluster, for example, has its own disk, memory, and network. In the SCC, memory, disk, and network get shared among all the cores on the chip.

Even with four memory controllers, the use of the mesh network implies that reading or writing to memory will involve the routers along the path to the proper memory controller. And that suggests to me that a lot of the issues with memory bandwidth contention will still exist in the SCC. OS developers will have to take the latency, based on position in the mesh network and the physical address of memory, into account when they design or modify their operating systems to use the SCC.

But it is the message passing that most intrigues me. Details are vague, but the message itself is not. Current multicore chips have cache-coherent memory, meaning that they also include hardware that keeps track of cache lines that are shared between cores. If data in one core’s copy of a cache line changes, then all other copies of that cache line must be invalidated and eventually updated with the current data. The cache-coherency mechanisms share the memory bus, as well as interfering with memory accesses, and this in itself is a limiting factor to how many cores can be used effectively in one chip.

Intel has announced a six-core chip (Gulftown) that still uses cache-coherent hardware, and the SCC has only been released in very limited quantities to researchers. Although the size of these chips is similar, as is the total transistor count—about 1.3 billion—the number of processors and how they maintain memory consistency are very different. I believe the issue here is software, as current AMD and Intel multicore chips are supported by a variety of operating systems.

Intel has demonstrated real systems running Linux on the SCC, so software capable of using these systems does exist. But the SCC takes the concept of multicore into the realm of manycore made with standard cores (Pentium-light CPUs with no out-of-order execution capability) into reality. What are lacking are operating systems and software that can take advantage of the amount of potential parallelism in the SCC.

Multikernels appear best posed as a new model for manycore operating systems. Barrelfish [6] is the best example around today. It not only already relies on message passing instead of cache coherency, but can also run on heterogeneous hardware. Not that the SCC provides this, as it is all x86, but if you imagine a system with intelligent NICs or cores dedicated to simple instruction pipelines (like GPUs), then Barrelfish is well suited to do this.

There are other forms of mildly distributed systems popping up. The Apple iPad uses its own CPU design. Based on an ARM processor, the A4 is a System-on-Chip (SoC), which means it incorporates many of the functions found in separate chips on motherboards in a single chip: the GPU, NIC, I/O bus, and memory controller. SoC designs using the ARM have been around for years, but it will be interesting to see just how well Apple's A4, running at 1GHz, will work in practice. That is, will the A4 be able to render Web pages quickly enough for impatient users, while not sucking dry its battery in a matter of a few hours?

Again, details about the A4 and its host, the iPad, are vague at this time, but iPads should be in the hands of users by the time you read this. Then we will see if the A4 is just another way Apple can lock in control, or if it is really an innovative processor design that saves energy while appearing as zippy as its more energy-intensive relatives, such as the Atom.

Lineup

We lead off this issue with another article about Hadoop. Konstantin Shvachko, one of the developers of the Hadoop File System (HFS), discusses the implications of having a single namespace server and how that might limit performance in very large Hadoop installations. Along the way, you will learn more about how the open source, distributed, cluster, but not cloud, HFS works and what it is capable of in terms of performance.

Next, I had the opportunity of exchanging email with Timothy Roscoe. Roscoe is one of the participants in the development of Barrelfish, the world's first multikernel OS. Mothy was kind enough to correct the many mistaken impressions I retained after reading the SOSP paper several times, and I found myself more enthused than ever about the direction taken by the Barrelfish researchers.

We also have several sysadmin researchers sharing their views about the future of sysadmin. Mark Burgess and Carolyn Rowland discuss the results of past LISA workshops on the Business Directions of IT Management (BDIM). The authors offer advice for sysadmins on how they might better align themselves with business goals and thus become a more integral part of their organization.

Alva Couch takes issue with describing system administrators in terms of the tasks they perform. Instead of tasks, Alva suggests using the notion of social contracts, as sysadmins do more than manage a mail server, for example. Sysadmins have tacitly agreed to provide a reliable mail service to their customers, which is an agreement that goes beyond the mere task of a configuring and maintaining a mail server.

We have two articles on file systems. The first, by William Josephson and his co-authors, is based on their FAST '10 paper about the Direct File System for virtualized flash devices. Josephson explains that key features of current file systems, the buffer cache and block allocation strategies, can actually hinder performance when used with a flash device that handles these features at the device-driver level. This technique places intimate knowledge about the flash device at a point in the stack where much more is known about the way the device operates. You will also learn more about how current flash drives (solid state drives) work and how flash devices that have interfaces like hard drives compare with the product used in this research.

Jake Wires and Andrew Warfield give us their perspective on file systems. Both Jake and Andy work with the Xen VMM, and this gives them a much different way of looking at how file systems should ideally work. Current VMMs hook into file systems at the block layer, and that obscures a lot of information that would make storage for VMs much more efficient or allow better methods of system updating.

Elizabeth Zwicky provides us with some advice about passwords. Using yet another massive exposure of passwords as her starting point, Elizabeth points out several strategies for the use of passwords, an old technology that just won't go away.

David Blank-Edelman expresses his admiration for regular expressions in Perl. As is usual for David, he provides useful modules that make regular expressions easier to use, something I would not have considered possible until I read his column.

Peter Galvin exposes us to deduplication in ZFS. Peter explains that deduplication is currently not supported by Sun/Oracle, but you can start using it now with the latest OpenSolaris build. Peter also provides examples of what deduplication does and does not do.

Dave Josephsen takes a look at how to get Nagios to scale further. The DNX event broker distributes events to worker nodes, so they can execute plugins and share load with the Nagios server. His second topic is the op5 Merlin module, an event broker that can synchronize events in the database of your choice, as well as perform load balancing and failover of Nagios.

Robert Ferrell examines network protocols that, while interesting to consider, failed for various reasons.

We conclude with book reviews by Elizabeth Zwicky and Brandon Ching.

There are no summaries in this issue, as there were no conferences or workshops over the Christmas holidays, for some reason.

The cloud is more than marketing talk, but also much more specific than marketers would have us believe. What I find much more interesting is how distributed systems, from smart phones and tablets, through manycore chips, right up to massive clusters, appear to be the future of computing.

REFERENCES

[1] Single Chip Cloud (SCC): http://www.theregister.co.uk/2009/12/02/intel_scc/.

[2] Teraflop chip: http://techresearch.intel.com/articles/Tera-Scale/1449.htm.

[3] McAfee Cloud (really SaaS): http://www.mcafee.com/us/enterprise/products/hosted_security/index.html.

[4] Channel I/O: http://en.wikipedia.org/wiki/Channel_I/O.

[5] Parallel Virtual Machine: http://www.csm.ornl.gov/pvm/pvm_home.html.

[6] Barrelfish: http://www.barrelfish.org/.

[7] Apple's A4: http://www.pcworld.com/businesscenter/article/188146/apple_inside_the_significance_of_the_ipads_a4_chip.html.

KONSTANTIN V. SHVACHKO

HDFS scalability: the limits to growth

Konstantin V. Shvachko is a principal software engineer at Yahoo!, where he develops HDFS. He specializes in efficient data structures and algorithms for large-scale distributed storage systems. He discovered a new type of balanced trees, S-trees, for optimal indexing of unstructured data, and he was a primary developer of an S-tree-based Linux file system, treeFS, a prototype of reiserFS. Konstantin holds a Ph.D. in computer science from Moscow State University, Russia. He is also a member of the Project Management Committee for Apache Hadoop.

shv@yahoo-inc.com

THE HADOOP DISTRIBUTED FILE SYS-tem (HDFS) is an open source system currently being used in situations where massive amounts of data need to be processed. Based on experience with the largest deployment of HDFS, I provide an analysis of how the amount of RAM of a single namespace server correlates with the storage capacity of Hadoop clusters, outline the advantages of the single-node namespace server architecture for linear performance scaling, and establish practical limits of growth for this architecture. This study may be applicable to issues with other distributed file systems.

By software evolution standards Hadoop is a young project. In 2005, inspired by two Google papers, Doug Cutting and Mike Cafarella implemented the core of Hadoop. Its wide acceptance and growth started in 2006 when Yahoo! began investing in its development and committed to use Hadoop as its internal distributed platform. During the past several years Hadoop installations have grown from a handful of nodes to thousands. It is now used in many organizations around the world.

In 2006, when the buzzword for storage was Exabyte, the Hadoop group at Yahoo! formulated long-term target requirements [7] for the Hadoop Distributed File System and outlined a list of projects intended to bring the requirements to life. What was clear then has now become a reality: the need for large distributed storage systems backed by distributed computational frameworks like Hadoop MapReduce is imminent.

Today, when we are on the verge of the Zettabyte Era, it is time to take a retrospective view of the targets and analyze what has been achieved, how aggressive our views on the evolution and needs of the storage world have been, how the achievements compare to competing systems, and what our limits to growth may be.

The main four-dimensional *scale requirement targets for HDFS* were formulated [7] as follows:

> 10PB capacity x 10,000 nodes x
> 100,000,000 files x 100,000 clients

The biggest Hadoop clusters [8, 5], such as the one recently used at Yahoo! to set sorting records, consist of 4000 nodes and have a total space capacity of 14PB each. Many production clusters run on 3000 nodes with 9PB storage capacities.

Hadoop clusters have been observed handling more than 100 million objects maintained by a single namespace server with a total capacity of 100 million files.

Four thousand node clusters successfully ran jobs with a total of more than 14,000 tasks reading from or writing to HDFS simultaneously.

Table 1 compares the targets with the current achievements:

	Target	**Deployed**
Capacity	10PB	14PB
Nodes	10,000	4000
Clients	100,000	15,000
Files	100,000,000	60,000,000

TABLE 1: TARGETS FOR HDFS VS. ACTUALLY DEPLOYED VALUES AS OF 2009

The bottom line is that we achieved the target in petabytes and got close to the target in the number of files, but this is done with a smaller number of nodes, and the need to support a workload close to 100,000 clients has not yet materialized.

The question is now whether the goals are feasible with the current system architecture. And the main concern is the *single namespace server architecture*. This article studies scalability and performance limitations imposed on HDFS by this architecture.

The methods developed in this work could be useful or applicable to other distributed systems with similar architecture.

The study is based on experience with today's largest deployments of Hadoop. The performance benchmarks were run on real clusters, and the storage capacity estimates were verified by extrapolating measurements taken from production systems.

HDFS at a Glance

Being a part of Hadoop core and serving as a storage layer for the Hadoop MapReduce framework, HDFS is also a stand-alone distributed file system like Lustre, GFS, PVFS, Panasas, GPFS, Ceph, and others. HDFS is optimized for batch processing focusing on the overall system throughput rather than individual operation latency.

As with most contemporary distributed file systems, HDFS is based on an architecture with the namespace decoupled from the data. The namespace forms the file system metadata, which is maintained by a dedicated server called the *name-node*. The data itself resides on other servers called *data-nodes*.

The file system data is accessed via *HDFS clients*, which first contact the name-node for data location and then transfer data to (write) or from (read) the specified data-nodes (see Figure 1).

The main motivation for decoupling the namespace from the data is the scalability of the system. Metadata operations are usually fast, whereas data transfers can last a long time. If a combined operation is passed through a single server (as in NFS), the data transfer component dominates the response time of the server, making it a bottleneck in a highly distributed environment.

In the decoupled architecture, fast metadata operations from multiple clients are addressed to the (usually single) namespace server, and the data transfers are distributed among the data servers utilizing the throughput of the whole cluster.

The namespace consists of files and directories. Directories define the hierarchical structure of the namespace. Files—the data containers—are divided into large (128MB each) blocks.

The name-node's metadata consist of the hierarchical namespace and a block to data-node mapping, which determines physical block locations.

In order to keep the rate of metadata operations high, HDFS keeps the whole namespace in RAM. The name-node persistently stores the namespace *image* and its modification log (the *journal*) in external memory such as a local or a remote hard drive.

The namespace image and the journal contain the HDFS file and directory names and their attributes (modification and access times, permissions, quotas), including block IDs for files, but not the locations of the blocks. The locations are reported by the data-nodes via block reports during startup and then periodically updated once an hour by default.

If the name-node fails, its latest state can be restored by reading the namespace image and replaying the journal.

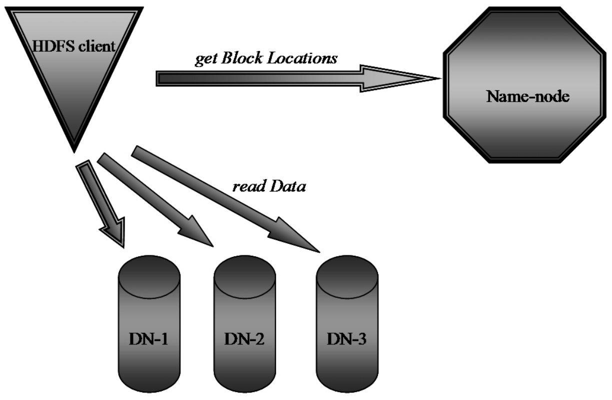


FIGURE 1: AN HDFS READ REQUEST STARTS WITH THE CLIENT MAKING A REQUEST TO THE NAME-NODE USING A FILE PATH, GETTING PHYSICAL BLOCK LOCATIONS, AND THEN ACCESSING DATA-NODES FOR THOSE BLOCKS.

Namespace Limitations

HDFS is built upon the single-node namespace server architecture.

Since the name-node is a single container of the file system metadata, it naturally becomes a limiting factor for file system growth. In order to make metadata operations fast, the name-node loads the whole namespace into its memory, and therefore the size of the namespace is limited by the amount of RAM available to the name-node.

Estimates show [12] that the name-node uses fewer than 200 bytes to store a single metadata object (a file inode or a block). According to statistics on our clusters, a file on average consists of 1.5 blocks, which means that it takes 600 bytes (1 file object + 2 block objects) to store an average file in name-node's RAM. This estimate does not include transient data structures, which the name-node creates for replicating or deleting blocks, etc., removing them when finished.

CONCLUSION 1

If

- $objSize$ is the size of a metadata object,
- λ is the average file to block ratio, and
- F is the total number of files,

then the memory footprint of the namespace server will be at least

$$RAM \geq F \cdot (1 + \lambda) \cdot objSize$$

Particularly, in order to store 100 million files (referencing 200 million blocks) a name-node should have at least 60GB ($10^8 \cdot 600$) of RAM. This matches observations on deployed clusters.

Replication

HDFS is designed to run on highly unreliable hardware. On Yahoo's long-running clusters we observe a node failure rate of 2–3 per 1000 nodes a day. On new (recently out of the factory) nodes, the rate is three times higher.

In order to provide data reliability HDFS uses block replication. Initially, each block is replicated by the client to three data-nodes. The block copies are called *replicas*. A replication factor of three is the default system parameter, which can either be configured or specified per file at creation time.

Once the block is created, its replication is maintained by the system automatically. The name-node detects failed data-nodes, or missing or corrupted individual replicas, and restores their replication by directing the copying of the remaining replicas to other nodes.

Replication is the simplest of known data-recovery techniques. Other techniques, such as redundant block striping or erasure codes, are applicable and have been used in other distributed file systems such as GPFS, PVFS, Lustre, and Panasas [1, 3, 6, 10]. These approaches, although more space efficient, also involve performance tradeoffs for data recovery. With striping, depending on the redundancy requirements, the system may need to read two or more of the remaining data segments from the nodes it has been striped to in order to reconstruct the missing one. Replication always needs only one copy.

For HDFS, the most important advantage of the replication technique is that it provides high availability of data in high demand. This is actively exploited by the MapReduce framework, as it increases replications of configuration and job library files to avoid contention during the job startup, when multiple tasks access the same files simultaneously.

Each block replica on a data-node is represented by a local (native file system) file. The size of this file equals the actual length of the block and does not require extra space to round it up to the maximum block size, as traditional file systems do. Thus, if a block is half full it needs only half of the space of the full block on the local drive. A slight overhead is added, since HDFS also stores a second, smaller metadata file for each block replica, which contains the checksums for the block data.

Replication is important both from reliability and availability points of view, and the default replication value of 3 seem to be reasonable in most cases for large, busy clusters.

STORAGE CAPACITY VS. NAMESPACE SIZE

With 100 million files each having an average of 1.5 blocks, we will have 200 million blocks in the file system. If the maximal block size is 128MB and every block is replicated three times, then the total disk space required to store these blocks is close to 60PB.

CONCLUSION 2

If

- *blockSize* is the maximal block size,
- r is the average block replication,
- λ is the average file-to-block ratio, and
- F is the total number of files,

then the storage capacity (SC) referenced by the namespace will not exceed

$$SC \le F \cdot \lambda \cdot r \cdot blockSize$$

Comparison of Conclusions 1 and 2 leads us to the following rule.

RULE 1

As a rule of thumb, the correlation between the representation of the metadata in RAM and physical storage space required to store data referenced by this namespace is:

1GB metadata ≈ 1PB physical storage

The rule should not be treated the same as, say, the Pythagorean Theorem, because the correlation depends on cluster parameters, the block-to-file ratio, and the block size, but it can be used as a practical estimate for configuring cluster resources.

CLUSTER SIZE AND NODE RESOURCES

Using Conclusion 2, we can estimate the number of data-nodes the cluster should have in order to accommodate namespace of a certain size.

On Yahoo's clusters, data-nodes are usually equipped with four disk drives of size 0.75–1TB, and configured to use 2.5–3.5TB of that space per node. The remaining space is allocated for MapReduce transient data, system logs, and the OS.

If we assume that an average data-node capacity is 3TB, then we will need on the order of 20,000 nodes to store 60PB of data. To be consistent with the target requirement of 10,000 nodes, each data-node should be configured with eight hard drives.

CONCLUSION 3

In order to accommodate data referenced by a 100 million file namespace, an HDFS cluster needs 10,000 nodes equipped with eight 1TB hard drives. The total storage capacity of such a cluster is 60PB.

Note that these estimates are true under the assumption that the block-per-file ratio of 1.5 and the block size remain the same. If the ratio or the block size increases, a gigabyte of RAM will support more petabytes of physical storage, and vice versa.

Sadly, based on practical observations, the *block-to-file ratio tends to decrease* during the lifetime of a file system, meaning that the object count (and therefore the memory footprint) of a single namespace server grows faster than the physical data storage. That makes the *object-count problem*, which becomes a *file-count problem* when $\lambda \rightarrow 1$, the real bottleneck for cluster scalability.

BLOCK REPORTS, HEARTBEATS

The name-node maintains a list of registered data-nodes and blocks belonging to each data-node.

A data-node identifies block replicas in its possession to the name-node by sending a *block report*. A block report contains *block ID*, *length,* and the *generation stamp* for each block replica.

The first block report is sent immediately after the data-node registration. It reveals block locations, which are not maintained in the namespace image or in the journal on the name-node. Subsequently, block reports are sent periodically every hour by default and serve as a sanity check, providing that the name-node has an up-to-date view of block replica distribution on the cluster.

During normal operation, data-nodes periodically send *heartbeats* to the name-node to indicate that the data-node is alive. The default heartbeat interval is three seconds. If the name-node does not receive a heartbeat from a data-node in 10 minutes, it pronounces the data-node dead and schedules its blocks for replication on other nodes.

Heartbeats also carry information about total and used disk capacity and the number of data transfers currently performed by the node, which plays an important role in the name-node's space and load-balancing decisions.

The communication on HDFS clusters is organized in such a way that the name-node does not call data-nodes directly. It uses heartbeats to reply to the data-nodes with important instructions. The instructions include commands to:

- Replicate blocks to other nodes
- Remove local block replicas
- Re-register or shut down the node
- Send an urgent block report

These commands are important for maintaining the overall system integrity; it is therefore imperative to keep heartbeats frequent even on big clusters. The name-node is optimized to process thousands of heartbeats per second without affecting other name-node operations.

THE INTERNAL LOAD

The block reports and heartbeats form the *internal load of the cluster*. This load mostly depends on the number of data-nodes. If the internal load is too high, the cluster becomes dysfunctional, able to process only a few, if any, external client operations such as *`ls`*, *`read`*, or *`write`*.

This section analyzes what percentage of the total processing power of the name-node is dedicated to the internal load.

Let's assume the cluster is built of 10,000 data-nodes having eight hard drives with 6TB of effective storage capacity each. This is what it takes, as we learned in previous sections, to conform to the targeted requirements.

As usual, our analysis is based on the assumption that the block-to-file ratio is 1.5.

The ratio particularly means that every other block on a data-node is half full. If we group data-node blocks into pairs having one full block and one half-full block, then each pair will occupy approximately 200 MB ≈ 128 MB + 64 MB on a hard drive. This gives us an estimate that a 6 TB (8 HD x 0.75 TB) node will hold 60,000 blocks. This is the size of an average block report sent by a data-node to the name-node.

The sending of block reports is randomized so that they do not come to the name-node together or in large waves. Thus, *the average number of block reports the name-node receives is* 10,000/hour, which is *about three reports per second*.

The heartbeats are not explicitly randomized by the current implementation and, in theory, can hit the name-node together, although the likelihood of this is very low. Nevertheless, let's assume that the name-node should be able to handle 10,000 heartbeats per second on a 10,000 node cluster.

In order to measure the name-node performance, I implemented a benchmark called NNThroughputBenchmark, which now is a standard part of the HDFS code base.

NNThroughputBenchmark is a single-node benchmark, which starts a name-node and runs a series of client threads on the same node. Each client repetitively performs the same name-node operation by directly calling the name-node method implementing this operation. Then the benchmark measures the number of operations performed by the name-node per second.

The reason for running clients locally rather than remotely from different nodes is to avoid any communication overhead caused by RPC connections and serialization, and thus reveal the upper bound of pure name-node performance.

The following numbers were obtained by running NNThroughputBenchmark on a node with two quad-core Xeon CPUs, 32GB RAM, and four 1TB hard drives.

Table 2 summarizes the name-node throughput with respect to the two internal operations. Note that the block report throughput is measured in the number of blocks processed by the name-node per second.

	Throughput
Number of blocks processed in block reports per second	639,713
Number of heartbeats per second	300,000

TABLE 2: BLOCK REPORT AND HEARTBEAT THROUGHPUT

We see that the name-node is able to process more than 10 reports per second, each consisting of 60,000 blocks. As we need to process only three reports per second, we may conclude that less than 30% of the name-node's total processing capacity will be used for handling block reports.

The heartbeat load is 3.3%, so that the combined internal load of block reports and heartbeats is still less than 30%.

CONCLUSION 4

> The internal load for block reports and heartbeat processing on a 10,000-node HDFS cluster with a total storage capacity of 60 PB will consume 30% of the total name-node processing capacity.

Thus, the internal cluster load directly depends on the average block report size and the number of the reports. The impact of heartbeats is negligible.

Another way to say this is that the internal load is proportional to the number of nodes in the cluster and the average number of blocks on a node. Thus, if a node had only 30,000 blocks, half of the estimated amount, then the name-node would dedicate only 15% of its processing resources to the internal load, because the nodes would send the same number of block reports but the size of the block reports would be smaller by a half compared to the original estimate.

Conversely, if the average number of blocks per node grows, then the internal load will grow proportionally. In particular, it means the decrease in block-to-file ratio (more small files with the same file system size) increases the internal load and therefore negatively affects the performance of the system.

REASONABLE LOAD EXPECTATIONS

The good news from the previous section is that the name-node can still use 70% of its time to process *external client requests*. If all the clients started sending arbitrary requests to the name-node with very high frequency, the name-node most probably would have a hard time coping with the load and would become unresponsive, potentially sending the whole cluster into a tailspin, because internal load requests do not have priority over regular client requests. But this can happen even on smaller clusters with extreme load levels.

The goal of this section is to determine *reasonable load expectations* on a large cluster (10,000 nodes, 60PB of data) and estimate whether the name-node would be able to handle it.

Regular Hadoop clusters run MapReduce jobs. We first assume that all our 100,000 clients running different tasks provide *read-only load* on the HDFS cluster. This is typical for the map stage of a job execution.

Usually a map task produces map output, which is written to a local hard drive. Since MapReduce servers (task-trackers) share nodes with HDFS data-nodes, map output inevitably competes with HDFS reads. This reduces the HDFS read throughput, but also decreases the load on the name-node. Thus, for the sake of this analysis we may assume that our *tasks do not produce any output*, because otherwise the load on the name-node would be lower.

Typically, a map task reads one block of data. In our case, files consist of 1.5 blocks. Thus an average client reads a chunk of data of size 96MB (1.5 * 128MB/2) and we may assume that *the size of a read operation per client is 96MB*.

Figure 1 illustrates that client reads conceptually consist of two stages:

1. Get block locations from the name-node.
2. Pull data (block replica) from the nearest data-node.

We will estimate how much time it takes for a client to retrieve a block replica and, based on that, derive how many "get block location" requests the name-node should expect per second from 100,000 clients.

DFSIO was one of the first standard benchmarks for HDFS. The benchmark is a map-reduce job with multiple mappers and a single reducer. Each mapper writes (reads) bytes to (from) a distinct file. Mappers within the job either all write or all read, and each mapper transfers the same amount of data. The mappers collect the I/O stats and pass them to the reducer. The reducer averages them and summarizes the I/O throughput for the job. The key measurement here is the byte transfer rate of an average mapper.

The following numbers were obtained on a 4000-node cluster [8] where the name-node configuration is the same as in NNThroughputBenchmark and data-nodes differ from the name-node only in that they have 8GB RAM. The cluster consists of 100 racks with 1 gigabit Ethernet inside a rack and 4 gigabit uplink from rack.

Table 3 summarizes the average client read and write throughput provided by DFSIO benchmark.

	Throughput
Average read throughput	66 MB/s
Average write throughput	40 MB/s

TABLE 3: HDFS READ AND WRITE THROUGHPUT

We see that an average client will read 96MB in 1.45 seconds. According to our assumptions, it will then go to the name-node to get block locations for another chunk of data or a file. Thus, 100,000 clients will produce 68,750 get-block-location requests to the name-node per second.

Another series of throughput results [11] produced by NNThroughputBenchmark (Table 4) measures the number of "open" (the same as "get block location") and "create" operations processed by the name-node per second:

	Throughput
Get block locations	126,119 ops/s
Create new block	5,600 ops/s

TABLE 4: OPEN AND CREATE THROUGHPUT

This shows that with the internal load at 30% the name-node will be able to process more than 88,000 get-block-location operations, which is enough to handle the read load of 68,750 ops/sec.

CONCLUSION 5

A 10,000-node HDFS cluster with internal load at 30% will be able to handle an expected read-only load produced by 100,000 HDFS clients.

The write performance looks less optimistic. For writes we consider a different distcp-like job load, which produces a lot of writes. As above, we assume that an average write size per client is 96MB. According to Table 3, an average client will write 96MB in 2.4 seconds. This provides an average load of 41,667 create-block requests per second from 100,000 clients, and this is way above 3,920 creates per second—70% of the possible processing capacity of the name-node (see Table 4). Furthermore, this does not yet take into account the 125,000 confirmations (three per block-create) sent by data-nodes to the name-node for each successfully received block replica.

Although these confirmations are not as heavy as create-blocks, this is still a substantial additional load.

Even at 100% processing capacity dedicated to external tasks (no internal load), the clients will not be able to run at “full speed” with writes. They will experience substantial idle cycles waiting for replies from the name-node.

CONCLUSION 6

> A reasonably expected write-only load produced by 100,000 HDFS clients on a 10,000-node HDFS cluster will exceed the throughput capacity of a single name-node.

Distributed systems are designed with the expectation of linear performance scaling: more workers should be able to produce a proportionately larger amount of work. The estimates above (working the math backwards) show that 10,000 clients can saturate the name-node for write-dominated workloads. On a 10,000-node cluster this is only one client per node, while current Hadoop clusters are set up to run up to four clients per node. This makes the single name-node a bottleneck for linear performance scaling of the entire cluster. There is no benefit in increasing the number of writers. A smaller number of clients will be able to write the same amount of bytes in the same time.

Final Notes

We have seen that a 10,000 node HDFS cluster with a single name-node is expected to handle well a workload of 100,000 readers, but even 10,000 writers can produce enough workload to saturate the name-node, making it a bottleneck for linear scaling.

Such a large difference in performance is attributed to get block locations (read workload) being a memory-only operation, while creates (write workload) require journaling, which is bounded by the local hard drive performance.

There are ways to improve the single name-node performance, but any solution intended for single namespace server optimization lacks scalability.

Looking into the future, especially taking into account that the ratio of small files tends to grow, the most promising solutions seem to be based on distributing the namespace server itself both for workload balancing and for reducing the single server memory footprint. There are just a few distributed file systems that implement such an approach.

Ceph [9] has a cluster of namespace servers (MDS) and uses a dynamic subtree partitioning algorithm in order to map the namespace tree to MDSes evenly. [9] reports experiments with 128 MDS nodes in the entire cluster consisting of 430 nodes. Per-MDS throughput drops 50% as the MDS cluster grows to 128 nodes.

Google recently announced [4] that GFS [2] has evolved into a distributed namespace server system. The new GFS can have hundreds of namespace servers (masters) with 100 million files per master. Each file is split into much smaller size than before (1 vs. 64 MB) blocks. The details of the design, the scalability, and performance facts are not yet known to the wider community.

Lustre [3] has an implementation of clustered namespace on its roadmap for the Lustre 2.2 release. The intent is to stripe a directory over multiple metadata servers (MDS), each of which contains a disjoint portion of the namespace. A file is assigned to a particular MDS using a hash function on the file name.

ACKNOWLEDGMENTS

I would like to thank Jakob Homan and Rik Farrow for their help with the article.

REFERENCES

[1] P.H. Carns, W.B. Ligon III, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 317–327.

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *Proceedings of the ACM Symposium on Operating Systems Principles*, Lake George, NY, October 2003, pp. 29–43.

[3] Lustre: http://www.lustre.org.

[4] M.K. McKusick and S. Quinlan, "GFS: Evolution on Fast-forward," *ACM Queue*, vol. 7, no. 7, ACM, New York, NY. August 2009.

[5] O. O'Malley and A.C. Murthy, "Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds," Yahoo! Developer Network Blog, May 11, 2009: http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte_in_162.html.

[6] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proceedings of FAST '02: 1st Conference on File and Storage Technologies* (USENIX Association, 2002), pp. 231–244.

[7] K.V. Shvachko, "The Hadoop Distributed File System Requirements," Hadoop Wiki, June 2006: http://wiki.apache.org/hadoop/DFS_requirements.

[8] K.V. Shvachko and A.C. Murthy, "Scaling Hadoop to 4000 Nodes at Yahoo!," Yahoo! Developer Network Blog, September 30, 2008: http://developer.yahoo.net/blogs/hadoop/2008/09/scaling_hadoop_to_4000_nodes_a.html.

[9] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proceedings of OSDI '06: 7th Conference on Operating Systems Design and Implementation* (USENIX Association, 2006).

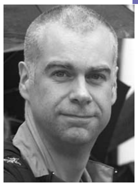
[10] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," *Proceedings of FAST '08: 6th Conference on File and Storage Technologies* (USENIX Association, 2008), pp. 17–33.

[11] "Compare Name-Node Performance When Journaling Is Performed into Local Hard-Drives or NFS," July 30, 2008: http://issues.apache.org/jira/browse/HADOOP-3860.

[12] "Name-Node Memory Size Estimates and Optimization Proposal," August 6, 2007: https://issues.apache.org/jira/browse/HADOOP-1687.

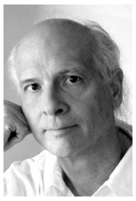
RIK FARROW

the Barrelfish multikernel: an interview with Timothy Roscoe



Timothy Roscoe is part of the ETH Zürich Computer Science Department's Systems Group. His main research areas are operating systems, distributed systems, and networking, with some critical theory on the side.

troscoe@inf.ethz.ch



Rik Farrow is the Editor of *;login:*.

rik@usenix.org

INCREASING CPU PERFORMANCE WITH faster clock speeds and ever more complex hardware for pipelining and memory access has hit the brick walls of power and bandwidth. Multicore CPUs provide the way forward but also present obstacles to using existing operating systems design as they scale upwards. Barrelfish represents an experimental operating system design where early versions run faster than Linux on the same hardware, with a design that should scale well to systems with many cores and even different CPU architectures.

Barrelfish explores the design of a multikernel operating system, one designed to run non-shared copies of key kernel data structures. Popular current operating systems, such as Windows and Linux, use a single, shared operating system image even when running on multiple-core CPUs as well as on motherboard designs with multiple CPUs. These monolithic kernels rely on cache coherency to protect shared data. Multikernels each have their own copy of key data structures and use message passing to maintain the correctness of each copy.

In their SOSP 2009 paper [1], Baumann et al. describe their experiences in building and benchmarking Barrelfish on a variety of Intel and AMD systems ranging from four to 32 cores. When these systems run Linux or Windows, they rely on cache coherency mechanisms to maintain a single image of the operating system. This is not the same thing as locking, which is used to protect changes to data elements which themselves consist of data structures, such as linked lists, that must be changed atomically. In monolithic kernels, a change to a data element must be visible to all CPUs, and this consistency gets triggered when a CPU attempts to read or write this data in its own cache. Cache consistency mechanisms prevent the completion of this read or write if the cache line is invalid, and also mean that execution may be paused until the operation is complete.

In a multikernel, each CPU core runs its own kernel and maintains its own data structures. When a kernel needs to make a change to a data structure (e.g., memory page tables) that must be coordinated with kernels running on other cores, it sends messages to the other kernels.

I asked Timothy Roscoe of the Systems Group at ETH Zurich if he could answer a few questions about Barrelfish, working in a manner similar to Barrelfish, using asynchronous messaging. Before I begin the interview, Mothy wanted me to point out that the development of Barrelfish involves a very large team of people, and he is just one person among many working on this very complex project. You can learn more about this team by visiting the Barrelfish Web site, http://www.barrelfish.org/.

Farrow: Barrelfish maintains separate kernel state, and this seems to me to be one of the key differentiators from monolithic kernels.

Roscoe: Actually, this is not quite, but nearly, true: monolithic kernels started with a single shared copy of kernel state, and to a limited extent they have started to replicate or partition this state to reduce memory contention on multiprocessors. Solaris is probably the most advanced version of this. The model, however, remains one of a single image managing the whole machine, with the replication and/or partitioning of kernel state as an optimization.

In a multikernel, this is the other way around. No kernel state at all is shared between cores by default, and so consistency must be maintained by explicitly sending messages between cores, as in a distributed system. The model is one of replicated or partitioned data which is accessed the same way as one would access replicas in a distributed system. In particular, depending on the consistency requirements, changing some OS state may be a two-phase operation: a core requests a change and, at some point in the future, gets confirmation back that every other core has agreed to it, or, alternatively, that it conflicted with some other proposed change and so didn't happen.

In principle, we could share kernel data between cores in Barrelfish, and this might be a good idea when the cores are closely coupled, such as when they share an L2 or L3 cache or are actually threads on the same core. We also intend to do this at some point, but the key idea is that the model is of replicated data, with sharing as a transparent optimization. In traditional kernels it's the other way around.

Farrow: Barrelfish has a small CPU driver that runs with privilege, and a larger monitor process that handles many of the tasks found in a monolithic operating system. Barrelfish is not a microkernel, as microkernels share a single operating system image, like much larger monolithic kernels. Barrelfish does seem to share some characteristics of microkernels, such as running device drivers as services, right?

Roscoe: You're right that every core in Barrelfish runs its own CPU driver, which shares no memory with any other core. Also, every core has its own monitor process, which has authority (via capabilities) to perform a number of privileged operations. Most of the functionality you would expect to find in a UNIX kernel is either in driver processes or servers (as you would expect in a microkernel) or the distributed network of monitor processes.

Farrow: The SOSP paper talks about a system knowledge base (SKB) that gets built at boot time using probes of ACPI tables, the PCI bus, CPUID data, and measurement of message passing latency. Could you explain the importance of the SKB in Barrelfish?

Roscoe: The SKB does two things. First, it represents as much knowledge as possible about the hardware in a subset of first-order logic—it's a Constraint Logic Programming system at the moment. This, as you say, is populated using resource discovery and online measurements. Second, because it's a reasoning engine, the OS and applications can query it by issuing constrained optimization queries.

This is very different from Linux, Solaris, or Windows: traditional OSes often make some information about hardware (such as NUMA zones) available, but they often over-abstract them, the format of the information is ad hoc, and they provide no clean ways to reason about it (resulting in a lot of non-portable complex heuristic code). The SKB is not a magic bullet, but it drastically simplifies writing OS and application code that needs to understand the machine, and it means that clients can use whatever abstractions of the hardware are best for them, rather than what the OS designer thought useful.

We currently build on ARM, x86_64, x86_32, and Beehive processors. We're currently also porting to Intel's recently announced SCC (Single-chip Cloud Computer), which is a somewhat unconventional variant of x86_32.

One interesting feature of Barrelfish is that you don't really "port" the OS to a different architecture; rather, you add support for an additional CPU driver. Since CPU drivers and monitors only communicate via messages, Barrelfish will in principle happily boot on a machine with a mixture of different processors.

Farrow: While reading the paper, I found myself getting confused when you discussed how a thread or process gets scheduled. Could you explain how this occurs in Barrelfish?

Roscoe: Well, here's one way to explain this: Barrelfish has a somewhat different view of a "process" from a monolithic OS, inasmuch as it has a concept of a process at all. It's probably better to think of Barrelfish as dealing with "applications" and "dispatchers."

Since an application should, in general, be able to run on multiple cores, and Barrelfish views the machine as a distributed system, it follows that an application also, at some level, is structured as a distributed system of discrete components which run on different cores and communicate with each other via messages.

Each of these "components," the representative of the application on the core, so to speak, is called a "dispatcher." Unlike a UNIX process (or thread), dispatchers don't migrate—they are tied to cores. When they are descheduled by the CPU driver for the core, their context is saved (as in UNIX), but when they are rescheduled, this is done by upcalling the dispatcher rather than resuming the context. This is what Psyche and Scheduler Activations did, to first approximation (and K42, which is what we took the term "dispatcher" from, and Nemesis, and a few other such systems).

Farrow: So how do you support a traditional, multi-threaded, shared-memory application like OpenMP, for example?

Roscoe: Well, first of all, each dispatcher has, in principle, its own virtual address space, since each core has a different MMU. For a shared-memory application, clearly these address spaces should be synchronized across the dispatchers that form the application so that they all look the same, whereupon the cache coherence hardware will do the rest of the work for us. We can achieve this either by messages or by sharing page tables directly, but in both cases some synchronization between dispatchers is always required when mappings change.

As an application programmer, you don't need to see this; the dispatcher library handles it. Incidentally, the dispatcher library also handles the application's page faults—another idea we borrowed from Nemesis and Exokernel.

Application threads are also managed by the dispatchers. As long as a thread remains on a single core, it is scheduled and context-switched by the dispatcher on that core (which, incidentally, is a much nicer way to implement a user-level threads package than using signals over UNIX). Note that the CPU driver doesn't know anything about threads, it just upcalls the dispatcher that handles these for the application, so lots of different thread models are possible.

To migrate threads between cores (and hence between dispatchers), one dispatcher has to hand off the thread to another. Since the memory holding the thread state is shared, this isn't too difficult. It's simply a question of making sure that at most one dispatcher thinks it owns the thread control block at a time. The dispatchers can either do this with spinlocks or by sending messages.

Farrow: Why should a multikernel work better than a monolithic kernel on manycore systems? In your paper, you do show better performance than a Linux kernel when running the same parallel tasks, but you also point out that the current Barrelfish implementation is much simpler/less functional than the current Linux kernel.

Roscoe: Our basic argument is to look at the trends in hardware and try to guess (and/or influence) where things are going to be in 10 years.

The main difference between a multikernel like Barrelfish and a monolithic OS like Linux, Windows, or Solaris is how it treats cache-coherent shared memory. In monolithic kernels, it's a basic foundation of how the system works: the kernel is a shared-memory multi-threaded program. A multikernel is designed to work without cache-coherence, or indeed without shared memory at all, by using explicit messages instead.

There are four reasons why this might be important:

First, cache-coherent shared memory can be slower than messages, even on machines today. Accessing and modifying a shared data structure involves moving cache lines around the machine, and this takes hundreds of machine cycles per line. Alternatively, you could encode your operation (what you want to be done to the data structure) in a compact form as a message, and send it to the core that has the data in cache. If the message is much smaller than the data you need to touch, and the message can be sent efficiently, this is going to be fast.

"Fast" might mean lower latency, but more important is that cores are generally stalled waiting for a cache line to arrive. If instead you send messages, you can do useful work while waiting for the reply to come back. As a result, the instruction throughput of the machine as a whole is much higher, and the load on the system interconnect is much lower—there's just less data flying around.

Ironically, in Barrelfish on today's hardware, we mostly use cache-coherent shared memory to implement our message passing. It's really the only mechanism you've got on an x86 multiprocessor, aside from inter-processor interrupts, which are *really* expensive. Even so, we can send a 64-byte message from one core to another with a cost of only two interconnect transactions (a cache invalidate and a cache fill), which is still much more efficient than modifying more than three or four cache lines of a shared data structure.

The second reason is that cache-coherent shared memory can be too hard to program. This sounds counterintuitive—it exists in theory to make things easier. It's not about shared-memory threads vs. messages per se either, which is an old debate that's still running. The real problem is that hardware is now changing too fast, faster than system software can keep up.

It's a bit tricky, but ultimately not too hard to write a correct parallel program for a shared-memory multiprocessor, and an OS is to a large extent a somewhat special case of this. What's much harder, as the scientific computing folks will tell you, is to get good performance and scaling out of it. The usual approach is to specialize and optimize the layout of data structures, etc., to suit what you know about the hardware. It's a skilled business, and particularly skilled for OS kernel developers.

The problem is that as hardware gets increasingly diverse, as is happening right now, you can't do this for general mass-market machines, as they're all too different in performance characteristics. Worse, new architectures with new performance tradeoffs are coming out all the time, and it's taking longer and longer for OS developers, whether in Microsoft or in the Linux community, to come out with optimizations like per-core locks or read-copy-update—there's simply too much OS refactoring involved every time.

With an OS built around inter-core message passing rather than shared data structures, you at least have a much better separation between the code responsible for OS correctness (the bit that initiates operations on the replicated data) and that responsible for making it fast (picking the right consistency algorithm, the per-core data layout, and the message passing implementation). We'd like to think this makes the OS code more agile as new hardware comes down the pipe.

The third reason is that cache-coherent shared memory doesn't always help, particularly when sharing data and code between very different processors. We're beginning to see machines with heterogeneous cores, and from the roadmaps this looks set to continue. You're going to want to optimize data structures for particular architectures or cache systems, and a one-size-fits-all shared format for the whole machine isn't going to be very efficient. The natural approach is to replicate the data where necessary, store it in a format appropriate to each core where a replica resides, and keep the replicas in sync using messages—essentially what we do in Barrelfish.

The fourth reason is that cache-coherent shared memory doesn't always exist. Even a high-end PC these days is an asymmetric, non-shared memory multiprocessor: GPUs, programmable NICs, etc., are largely ignored by modern operating systems and are hidden behind device interfaces, firmware blobs, or, at best, somewhat primitive access methods like CUDA.

We argue that it's the job of the OS to manage all the processors on a machine, and Barrelfish is an OS designed to be able to do that, regardless of how these programmable devices can communicate with each other or the so-called "main" processors.

It's not even clear that "main" CPUs will be cache-coherent in the future. Research chips like the Intel SCC are not coherent, although they do have interesting support for inter-core message passing. I'm not sure there's any consensus among the architects as to whether hardware cache-coherence is going to remain worth the transistor budget, but there's a good chance it won't, particularly if there is system software whose performance simply doesn't need it.

Barrelfish is first and foremost a feasibility study for this—knowing what we now do about how to build distributed systems, message passing, programming tools, knowledge representation and inference, etc., we can build an OS for today's and tomorrow's hardware which is at least competitive with current performance on a highly engineered traditional OS and which can scale out more effectively and more easily in the future.

If a handful of researchers can do that, it sounds like a result.

REFERENCE

[1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania, "The Multikernel: A New OS Architecture for Scalable Multicore Systems," *Proceedings of the 22nd ACM Symposium on OS Principles*, Big Sky, MT, USA, October 2009: http://www.barrelfish.org/barrelfish_sosp09.pdf.

N.B.: The Barrelfish team also includes researchers Jan Rellermeyer, Richard Black, Orion Hodson, Ankush Gupta, Raffaele Sandrini, Dario Simone, Animesh Trivedi, Gustavo Alonso, and Tom Anderson.

MARK BURGESS AND
CAROLYN ROWLAND

the business value of system administration

Mark Burgess is professor of network and system administration at Oslo University College, Norway. He is the author of Cfengine, co-author of the Short Topics Booklet *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine,* and author of many books and research papers on system administration.

Mark.Burgess@iu.hio.no

Carolyn Rowland is a supervisory system administrator working at the National Insitute of Standards and Technology (NIST) in Gaithersburg, MD. She bridges the chasm between sysadmins and business on a daily basis.

carolyn@nist.gov

THE STATUS OF SYSTEM ADMINISTRATORS as experts is at stake as both technology and businesses evolve. To evolve in step, professionals need to become more business aware. In this article we summarize discussions on business alignment that took place at the LISA BDIM (Business Driven IT Management) workshops over the past two years, and we try to place them in a wider context. The outcome points to some straightforward tips to improve sysadmin business value.

The role of the traditional system administrator is changing. It is being packaged, commoditized, and standardized, just like the hardware and the software it relies on. Even the name "system administrator" is being forgotten and replaced by a generation that doesn't know its history. This should not come as any surprise. It is the inevitable process of evolution at work, forcing improvised origins into mainstream commerce.

In many ways, the changes we see are aftershocks from the arrival of the PC and Microsoft Windows, where commercialization began the transformation, starting with the basic tools. The currency of progress in the world of Windows is tied, of course, to business goals: the PC was created for businesses. By contrast, those who came to system administration from the research culture of mainframes and UNIX had mastery of the system as their prize. PCs, like minicomputers, ushered in standardized programs and business recipes so that repeatable simplicity could streamline success.

Today many basic tasks of system administration have been simplified by technologies such as automated configuration management, Web servers, content management systems, and package managers. Where does this leave the system administrator as we understand the term? Today the challenges of IT specialists include new issues such as massive scale, service orientation, business agility, and knowledge management, but the system administrator of the future is going to have to demonstrate new skills and lean business value by steering systems on the fine line between agility and stability.

Business vs Tech—Entrenched and Under Fire

A traditional organization has layers, also known as departments. Some do management, sales, business development, etc., and some do the technical work of the business (whether that be IT services, carpentry, or brickwork). This separation of concerns makes a lot of sense, as the skills and personality types needed to perform sales and management are quite different from those needed for technical work.

Who are these people? The business layers, which traditionally include management, deal with the raison d'etre of the company—where is the money coming from, and what should the company really be about? There is a lot of thinking and soul searching at this level, planning and brainstorming. Technical levels are typically governed by the more predictable processes of engineering and resource management.

Irksome perhaps (for dedicated IT staff), a business succeeds only if the business departments are successful. The technical worker is a helper, and sometimes an enabler, but technical work alone does not a business make.

In commerce, one needs an edge to succeed:

- The perception of *confidence*
- Rapid turnaround of ideas, or *time to market*
- The *unique ("business") value* of what is being sold.

Such concepts were basically absent from discussions on system administration,before the BDIM workshops at LISA '08 and '09, but we believe that to develop as a profession, system administrators must confront the IT/Business divide.

Sysadmins Speak

The attendees of the BDIM LISA workshops had plenty to say about the IT/Business barrier. We wanted to know how people in the field perceived the relationship between IT and business. We began with some questions:

- Do best practices exist?
- What metrics do we have for alignment?
- How does one define business processes?
- Where does research and development fit in?
- Does this apply only to e-commerce?
- What role is played by communications (phone, mail, etc.)?
- What does mission-critical mean?

Not all of these questions were really answered satisfactorily, but the emergent dialogues were a valuable source of insight into the Business/IT relationship at different workplaces. We heard the following issues repeatedly:

- Better communication is needed between Business and IT.
- Sysadmins are often the last to hear about needs and changes for the business. Advance warning of upcoming issues helps. Having something like a five-year plan helps everyone to overcome a day-to-day regimen of putting out fires.
- Sysadmins should be trusted partners, not grumpy slaves offering expert advice. They need communication and people skills.
- Upward visibility of sysadmins is needed—sysadmins need to document the impact of their work in relation to the business. One way is to charge for services in an in-sourcing model.
- Management should provide incentives to document and simplify processes to prevent development of "king of the hill" scenarios, crippling from within.
- Organizations need headroom to meet new challenges. Some departments are optimized to the point of rigidity, which makes an organization brittle.

We expand on some of these points below as we attempt to paint a broader picture of the Business/IT relationship.

Talk between Experts

How do we achieve better communication between Business and IT? Why are IT people often the last to hear about the need for change in the business? Responsibility surely lies on both sides, but one answer is cultural. Rather than working "heads down," IT needs to take an active role in analyzing and advising Business. IT needs to be perceived as a "trusted partner" by Business—increasing that perception of confidence and ensuring the rapid turnaround of ideas alluded to above.

If Business learns that the IT department has valuable input, it will respond by turning to them for advice. Strong communication and "people skills" are important here. Sysadmins need to learn Business language and avoid IT jargon, as well as talk much more about the *impact* of their work, the *mission*, and the *costs*.

A milestone of success is when Business values the opinions of IT enough to make them part of the management team. One way to formalize the relationship is the creation of a five-year plan for IT. This allows the IT department to be involved at the problem-definition level. Ultimately, sysadmins need to develop a relationship of trust with Business. A lack of trust means lack of business credibility and low status for sysadmins. This lack of trust will cost the business too, as it is unable to get the most from IT.

Visibility of the system administrators' work is an important education for Business. Documenting the impact of the work, not just the work itself, shows Business measurable accomplishments from the IT layer. A blind way to achieve this is simply to charge for services, using an in-sourcing model, so that Business can see the actual costs of using specific technologies. Being generous with, rather than protective of, expertise shows the return on investment (ROI) from IT; such visibility is essential if IT personnel are not to be replaced by a lowest-common-denominator workforce. External IT requirements (SOX, HIPAA, FISMA, STIGs, etc.) can be confusing to Business, and expertise is needed to communicate the impact and costs of compliance on the business, plus obtain necessary budgetary support to implement it.

Simplicity vs. Complexity: Any Color You Like so Long as It's Black

Business does not usually have the time or the wherewithal to comprehend complex technicalities, so a simple environment intuitively seems preferable. Often there is a perception that a simpler environment costs less to maintain: fewer hardware and software requirements, fewer IT staff, and fewer skill sets for those IT staff.

For sysadmins, the mood can be to swim upstream. The latest technology might be irresistible, but installing and configuring costs time. Coding private software rather than buying a solution might seem cool, but is it costing the business in the long run? When is it ethical to explore on company time? Does the new item meet a business need? Did anyone ask for that capability? What is the impact? Does it cost more to maintain as the infrastructure becomes more complex or, alternatively, does it aid in compliance with external requirements or close an IT security hole? What is the ROI?

Many organizations distrust change and variation. They expect heterogeneity to be a problem. A major issue is, therefore, comprehension: how can IT explain to Business what the system will do; if it seems overly complicated to them? Doesn't a complex system cost too much? IT must better understand business needs in order to communicate the need for heterogeneity to Business. That said, oversimplification is not a foreign concept in system administration either.

At the earliest LISA conferences, in the 1980s, papers were being written about how to make all machines in an organization identical in order to save work. Thirty years later, it is still a prevalent strategy to create one or two standard "images" and to force these on all machines in the organization to avoid dealing with necessary variation. Cloud computing is even making a business model out of selling people incomplete machines, blank slates, to be configured individually. Alva Couch, Associate Professor at Tufts University, referred to system homogeneity as the "nuclear weapon" of predictability in server management, implying that it is too heavy-handed an approach to managing expectations. The counter-argument is that consistency has advantages if there is no need for variation, because it reduces the amount of IT knowledge required. It boils down to the difference between *intended* or controlled variation and *unintended* (i.e., out of control) variation. A useful middle ground between complete homogeneity and rampant heterogeneity is to foster *enclaves of uniformity*. System administrators could turn complexity vs. simplicity into a conversation with Business on the best way to support the core mission of the organization.

Business can offer incentives to IT to simplify and document the infrastructure. However, forcing a simplistic environment will cause problems when homogeneity is imposed through lack of understanding (such as in a research environment, where freedom to "play" brings long-term value). IT needs to sell this to Business; it could cost more in immediate outlay but serve Business better in the long run.

One way Business has sought to simplify their understanding of IT is through the SLA. The term "SLA" (service level agreement) was originally conceived as a legal agreement between service providers and service consumers, documenting promised operational levels and repercussions if service could not be delivered. Over time, the term has been used in an increasingly casual way to talk about promised objectives. But a more fundamental question is: are we providing the right catalogue of services in the first place?

Simplistic marketing promises like "five nines reliability," i.e., 99.999% uptime, attempt to push people's fear buttons to spend money on verification. Do such slogans make any rational connection with Business goals? What is the cost of keeping such a promise? Would the Business rather be happy with a promise of two nines at a tenth of the cost? To be able to make a promise is a powerful confidence builder, but it has to be one that it can afford to keep. Sysadmins need to bring rationality to this discussion on behalf of Business.

Leaving Room for Change

Business likes to think that it is the driver of change and everyone else is dragging their feet. This is not the case. Usually both sides drag their feet. Business leaders generally like the word "innovation" but sneer at words like "research," so there is a paradox to be faced—as noted above. People tend to stick with what they know, even when it is harmful to them.

Agility in IT or in Business requires us to be able to face sudden challenges. Reactive fire-fighting approaches to IT management hinder this. Creative thinking requires "headroom" or "slack." IT has to plan for this, and Business has to fund it. Business commonly sees IT support as a tax on their organization. If one could create a lean-and-mean IT organization, then there would be more cash to spend on main mission-specific objectives. However, being bogged down with fire-fighting or optimized to the point of rigidity makes an organization brittle, and breakages occur during sudden change. Some organizations, fearful of allowing change, intentionally throw syrup over their staff in the form of red tape and bureaucracy, which makes it hard for them to be agile.

One way to improve agility is to modularize. Modularity of systems is universal lore today. We understand that building systems from "off the shelf" standard parts has many advantages, including economies of scale and the possibility of outsourcing. The service paradigm is part of this phenomenon, and the culture has edged its way steadily into IT since the 1980s.

Best Practices—Do They Exist?

Complexity and fragmentation of IT operations has motivated another kind of standard over the past 25 years. Enter COBIT (Control Objectives for Information and Related Technology) and ITIL (Information Technology Infrastructure Library), which present themselves as "best practice" frameworks. These de facto standards employ various levels of recommended practices for IT governance and service delivery. Individuals can even get certified in the ways of ITIL. Controversy surrounds the efficacy of ITIL, and critics decry the added bureaucracy and lack of agility that come with a full ITIL implementation. Moreover, the frameworks are unable to explain why they deserve the accolade "best." What is interesting is that, once again, their very existence suggests a need, hence, something wanting in the industry. In spite of their paucity of technical content, the frameworks exist to provide Business with a simple language in which to define, describe, and manage IT.

What Did You Do for Business Today?

When all is said and done, consider the question: *What did you contribute to the success of your business today?* We'd wager that most system administrators, junior or senior, would falter if asked this question, mumbling something about technologies and help tickets transacted, rather than business impact. What about: *What does your organization do? What is its primary goal?* Few organizations reflect on their own activities clearly, and it is easy to oversimplify: universities do not just do teaching, Microsoft does not just do software development. Organizations have diverse departmental activities, all contributing to their day-to-day business.

Reflecting on goals is usually shoved into a "manager" tray. "That's not my job to think about!" Perhaps herein lies the root of a problem. Planting this question could provoke an act of infectious cultural awareness.

Here is a very business-like thing to do: a Top Ten list of priorities. Prioritization is an economic imperative. Such lists ask us to make value judgments and confront pressing issues. You can make your own.

1. Answer the question, "What does your business do?" Then ask, "How does IT fit into the big picture?"
2. Show leadership in Business/IT decision-making. Arrange regular meetings between Business and IT; even appoint a permanent liaison who can translate.

3. Communicate effectively. IT should learn the language of Business; drop the jargon and speak in terms Business will understand.
4. IT should start thinking and talking about the impact of the work, rather than the details (think ROI). Find out how others assess IT's efforts and use that knowledge to increase visibility.
5. Understand the business in order to explain complexities (heterogeneity).
6. Cache knowledge. Make procedures as simple as they can possibly be, so that you can scale them in a crisis without having to rely on expertise always being present.
7. IT should know when to spend company time (i.e., money) researching new technology. What does it do for the business? What problem does it solve? Be prepared to defend the time and effort.
8. Create buffers. Weak coupling of modular roles offers much-needed slack or headroom for the IT department.
9. Formulate changes and strategies in terms of promises (who, what, when, where, why) and make it your business to keep them.
10. Be open to personal as well as institutional change.

Summary

Thinking about the interface between Business and IT, it is tempting to place oneself in the trenches and to blame "middle management" from above or below, some poor person who is responsible for oiling the gears. That is merely avoiding the issue. The issue is, rather, the whole organization's collective role in its own management.

If we ask how to align Business and IT, it makes sense to find the common ground. Sysadmins and engineers try to bring that predictability to users. Business folk are trying to engineer predictable streams of revenue in a fickle environment. Are these views compatible? Surely they are, with openness and focus.

We have seen conference discussions that take hours arguing over sysadmin self-image rather than biting the bullet of change and adapting. What will be the professional shape of system administrators to come? They will have to be increasingly in tune with their organization's diverse goals. They will ask, "What are the core promises of my organization, and what did I do to keep these promises today?"

REFERENCE

An extended version of this article is available at: http://www.iu.hio.no/~mark/blog_busval.html.

ACKNOWLEDGMENT

This work is supported by the EC IST-EMANICS Network of Excellence (#26854).

ALVA L. COUCH

from tasks to assurances: redefining system administration

Alva Couch is an associate professor of computer science at Tufts University, where he and his students study the theory and practice of network and system administration. He served as program chair of LISA '02 and was a recipient of the 2003 SAGE Outstanding Achievement Award for contributions to the theory of system administration. He currently serves as Secretary of the USENIX Board of Directors.

couch@cs.tufts.edu

AN ALTERNATIVE CONCEPT OF THE JOB of the system administrator leverages existing techniques of systems engineering and provides a foundation for a more synergistic relationship between system administrators and users.

Historically, there has been much confusion about what a "system administrator" is and does. One great success of the past decade is that we managed to define system administration in terms of the tasks the typical system administrator performs. This definition includes a taxonomy of system administration tasks [1], as well as the *Job Descriptions for System Administrators* booklet [2]. So far, these have served well as a de facto definition of the profession of system administration. We have obtained much leverage from this definition, including that the profession of system administration is now included as an option on the 2010 census.

But defining the profession in terms of tasks has a dark side; it invites naive observers to assume—as they do for plumbers and electricians—that our tasks define our profession [3]. In fact, our actual deliverables are much more abstract, including availability, integrity, and security. These are elements of the social contract between system administrators and users. I propose that this social contract, and not the tasks, is the real definition of the profession. What we are is not "what we do" but, rather, "what assurances we provide." Tasks support assurances, but are *not* the essence of the profession.

This is probably obvious to the average system administrator, but not at all obvious to management, who still on average consider system administration to be a task-based profession. We are to some extent "victims of our own success" in defining the profession via tasks. While tasks are easy to understand, social contracts are more abstract. How can we even write down the contract? Is the social contract defined in that ethereal thing called "policy," or something else? In the following, we explore some approaches to documenting the oft invisible and implicit social contract that is—already—a central component of the profession of system administration.

This article arose from teaching requirements analysis to aspiring software engineers last fall. The key principle of requirements analysis is to separate "requirements" from "design," in the sense that what a system should do ("requirements") is separate from how that is accomplished ("design"). *Separating requirements from design has many positive effects, including allowing the designer the freedom to address requirements in creative ways.* I asked myself, "Can these principles be applied to system administration to obtain similar benefits?" I realized that the prevalent definition of the profession in terms of tasks is actually "design," and that we seldom write down requirements in any other form. This article is the result of that train of thought.

This article might be loosely considered the third in a series. In the first article [4], I described the semantic wall between "high-level" and "low-level" specifications of system configuration and concluded that a new way of thinking is necessary to utilize "high-level" specifications. In the second article [5], I challenged the popular definition of system administration as managing system configuration, and redefined the profession as "closing open worlds," i.e., creating zones of predictable system behavior in an otherwise unpredictable world. In this article I take the next step, considering *which* worlds to close. This step comes with its own quandaries: the user wants the administrator to close "every" world, and boundaries must be drawn between what is "supported" and what is not. The decision as to which worlds to close is a social contract between system administrator and user.

From Tasks to Assurances

My first step is to drastically redefine the profession in a subtle but profound way. System administrators do not "perform tasks" or "apply expertise" but, rather, "provide assurances." An assurance is a clear statement of intent to address some user need. The set of assurances that a system administrator provides are part of the *social contract* between administrator and users. In very much the same way, while a plumber or electrician needs the prerequisites of being able to plumb or wire your house, in actuality these professionals honor a social contract that includes requirements for quality and reliability of the work they perform.

Converting between the old task-based definition and the new contract-based definition can be tricky. Sometimes the conversion between tasks and assurances is easy to make. For example, a system administrator is often seen as "managing printing" (a task), while the real job is "assuring that printing works" (a contractual obligation). Sometimes an assurance is based upon ability to perform a hopefully very infrequent task: for example, "recover from disasters" (a task) becomes "assure data integrity" (a contractual obligation). Some simply stated assurances are very difficult to map to tasks, e.g., "provide high-availability file service" requires mastery of many interrelated tasks.

THE SOCIAL CONTRACT

The social contract between administrator and user includes many facets. The most obvious of these are "ethics" and "privacy" assurances, which are now increasingly defined in writing as part of the job. But, at a deeper level, the social contract includes the assurances that the system administrator group makes about system behavior, as well as the priority of each assurance. A high-priority assurance will be addressed before a lower-priority assurance: if both the file server and a user application die at the same time, for example, the file server obviously takes priority.

PRIORITIES

Priorities are almost never documented in practice, and I think they should be! At one time or another, every system administrator gets into the situation of having to assure too much and has to make difficult decisions. What if a crucial program drops out of the user environment at the same time that the file server becomes unreliable? Luckily, a good manager is often there to defend decisions, but the priorities of assurances are often known far in advance. Writing them down changes the job from a "management decision" into "working from a pattern."

FLEXIBILITY

As in systems engineering, the main reason for describing assurances rather than tasks ("requirements" rather than "design") is to give the system administrator flexibility in providing those assurances. Assuring data integrity is a rather complex obligation, involving techniques for backup, recovery, and data security. The beauty of documenting assurances is that when some heretofore unknown technology comes along (e.g., using unused disk space as online backup), the assurance does not change, even though the tasks that provide that assurance might change drastically.

REUSABILITY

Are assurances reusable? The good news is that the kinds of assurances a system administrator makes do not vary much from site to site, that is, the list of assurances is (somewhat) reusable and relatively high-level compared to the task-based description of the job. Some of the most basic are present everywhere, including assurances of ethical behavior and appropriate safeguards for personal privacy. The taxonomy of assurances is really quite simple compared to the taxonomy of tasks. The bad news is that the priorities of various assurances differ greatly based upon the site. For example, empowering the user to do self-directed work might be the highest-priority assurance at an academic site and the lowest-priority assurance at a bank.

ASSURANCES ARE NOT POLICY

One might think that the definition of the job of system administrator arises from that ethereal thing we call "policy." It does not. "Policy" describes what systems and users should do, not who assures them and what forms that assurance takes. Many assurances that a system administrator makes are an implicit part of policy; use of a service implies reliability of the service. The transformation that turns policy into the social contract comes from asking, "What assurances are required to implement policy?"

Assurances and Requirements

At the most basic level, assurances for system administration are a list of system behaviors that should form a set of reasonable expectations on the part of users. At a deeper level, assurances are driven by (and are a proper superset of) user requirements: the things that users need in order to get their work done. The skilled system administrator converts the list of user requirements into a set of assurances by adding the implicit assurances of security, integrity, stability, etc., just as an electrician does not ask a customer whether to make outside wiring waterproof! At the next level, requirements become a set of service level objectives (SLOs) or even service level agreements (SLAs) defining response-time assurances: if and when things go wrong, how long should it take to correct problems? For example, an expectation is that “printing should work” and an SLO for that is that “a malfunctioning printer should be repaired in one day or less.”

System administration is a rather unusual profession in that the actual behavioral requirements often take second place to the techniques and practices by which behaviors are assured, and documentation of practices often serves as the sole documentation of requirements. One obvious reason for this is that documentation of practice is currently the *only* common language we have for describing behavior! It is easy in this situation to confuse that documentation with requirements and, when we do that, our practice becomes a parody of satisfying user needs rather than the real thing.

For example, consider the task of managing printing. The “tasks” include doing various things that ostensibly keep printing working, including managing the service, repairing printers, etc. Our documentation of managing printing includes details on how to accomplish these tasks. But these tasks by themselves cannot be converted easily into SLOs. The corresponding assurance, by contrast, is much simpler: “Everyone is able to print in a timely fashion.” This is easily converted into an SLO.

We are very lucky that the task of describing behaviors and requirements has been studied in great detail by others. In systems engineering and software engineering practice, this practice is called “requirements analysis” [6]. A “requirement” is something that the managed system should do, some behavior it should exhibit. There are many ways to document requirements, and there are several established techniques for accurately teasing requirements from user desires. One way to describe requirements is through first documenting “use cases,” from which we then extract and describe a “requirements model.”

USE CASES

Our first step in establishing a language for describing behavior is the same as in software or systems engineering. “Use cases” describe what the user should be able to do: for example, “users should be able to send and receive electronic mail.” Note that the use case does not specify how or why any behavior should be assured, and is thus much simpler and broader than a practice for assuring behavior.

Several issues arise immediately when we write down the use cases. First, use cases are not definitive; they describe some things that should be possible, but not absolutely everything. To assure the use cases, we are left to fill in the details of other things that should be possible. Use cases describe mission-critical behavioral objectives but not peripheral objectives that users might desire. For example, “checks should be printable” is included but “personal greeting cards should be printable” is not. Use cases often include SLOs for how quickly something *should* happen, which can even, in some cases, become SLAs on how quickly something *must* happen. There is a big difference, for example, between the use case statements “sales transactions *should* be posted within two seconds” and “sales transactions *must* be posted within two seconds.” Finally, use cases should not describe in any way *how* objectives are to be assured.

REQUIREMENTS MODELING

The next step is to abstract the use cases into patterns and concise representations. In software engineering, this phase is called "requirements analysis." Requirements analysis involves determining the classes of users and services (from the use cases) and documenting the relationships between classes of users (e.g., assignment-of-privilege classes to user classes and documenting inheritance between kinds of user and privilege classes). This is commonly referred to as a "modeling step," and the process is called "requirements modeling."

One powerful tool in requirements modeling is to express capabilities in terms of similarities between user roles, using object-oriented modeling. For example, there might be two kinds of users, "doctors" and "nurses," with different privileges. It might be that "doctors" are allowed to do things "nurses" cannot, but "doctors" can do anything "nurses" can do. Regardless of the real-world relationships between doctors and nurses, the behavior of the system in response to their queries is a simple inheritance relationship between behavioral classes: "doctor" system behavior is a subclass of "nurse" system behavior.

HOMOGENEITY AND HETEROGENEITY

In system administration, the terms "homogeneity" and "heterogeneity" usually refer to variation in the operating systems or hardware deployed at a site; we say that a site with a mix of Windows and Linux is "heterogeneous," for example, while a Linux-only site is "homogeneous." In requirements analysis, however, it is the user classes and behaviors that are homogeneous or heterogeneous, and not the managed systems! Users are "heterogeneous" if there are many user classes with different privileges, and "homogeneous" if all users have more or less the same privilege. Behavior is "heterogeneous" if there are different behaviors for each user class and "homogeneous" if not. These are properties of the mission and structure of the organization and not of the hardware on everyone's desks.

Like operating system heterogeneity, requirements heterogeneity costs more to assure. Thus it is prudent to question whether heterogeneity that naturally arises in requirements is necessary. For example, suppose that there is a *requirement* that user George has access to software to which no one else has access. This is going to be expensive, and one should ensure that this is *really* a requirement before proceeding. I believe that in many cases, heterogeneity of requirements is no less expensive than if George had a different *operating system* on his desktop machine.

AMBIGUITY

Once you have written down the explicit requirements, a pattern will emerge that is not unique to system administration. What you do not write down is as important as what you do. In any high-level description of requirements there will be some necessary ambiguity. Whether the requirements are useful at all depends on how we handle this ambiguity.

Suppose, for example, that one requirement is that "George should be able to compile files with gcc." Alas, this just isn't enough to describe precisely what George should be able to do. It does not say which header files should be present, or whether the kernel sources should be present to make kernel headers available. There are a multitude of factors exterior to gcc that might affect whether George can compile *his* files with gcc. George's real requirements are thus ambiguous, based upon your description.

Drift

Ambiguity is not nearly as bad in itself as in its social consequences. Anyone who goes to the trouble of writing down requirements will quickly discover that users are doing things that are outside the requirements—and getting away with them. In a modern computing environment, there is a prevalent idea that anything you are allowed to do is "supported" (or "assured"). But things you just happen to be allowed to do that are not requirements may go away at any time, due to policy changes, side effects of other changes, or simple mistakes.

A few years ago, at the beginning of the anti-spam effort, Tufts' Department of Computer Science closed down all access to SMTP from outside Tufts except for a few designated servers. The result was an outcry from students who had been running their own SMTP servers inside our network. The affected students claimed that our actions were costing them money by prohibiting business communications to their computers. The students were given a polite choice between relocating their business computing outside the Tufts network, and facing disciplinary charges for operating private businesses inside the university network!

This is an extreme example of a more general phenomenon that makes it difficult to specify requirements. Requirements are not what one can manage to do but, rather, what one *should* be able to do. They are not about what users *want* but, instead, about what users *need* in order to accomplish useful work, which is their end of the social contract.

REFINEMENT

Users are fairly good at describing their functional requirements, but less able to voice their requirements for privacy, security, integrity, and availability. Thus, the system administrator must often augment the list of user needs with implicit needs that users usually cannot voice. In requirements analysis, we might call the derived requirements a "refinement" of the basic user requirements.

Refinement is a matter of listing the requirements that are obvious to system administrators but not to the user. A good refinement consists of new requirements that are "obvious once written down." If one is refining correctly, the user's response will be, "Of course I need that."

BASELINING

One useful technique for the system administrator is to define behavioral requirements in terms of a baseline set of behaviors. This is a set of behaviors everyone should have access to in order to get their work done. It is a handy way of distinguishing between what users *need* (baseline behaviors) and what users *want* (non-baseline behaviors).

For example, at Tufts we have placed a limit on what users can expect from the support organization by establishing a "baseline configuration" for a desktop computer. This configuration satisfies a set of requirements necessary for interoperating with Tufts network services. But it has another social function, which is to define and delimit the responsibility of the support organization. Systems that fail to function according to the baseline will be returned to a baseline state, but functions that users desire outside the baseline are not supported.

The distinction between baseline and non-baseline behaviors can lead to major cost savings. Some organizations have reported that deploying thin clients that support only baseline functions (and, for example, prohibit the installation of custom software) results in up to 50% savings in cost of operations. Allowing users to install seemingly innocent software (e.g., MP3 players) can lead to substantially increased support costs.

As another example, in some system administration circles the words "reasonable faculty member" are an oxymoron. My support staff and I have an unusual social contract. I need high volatility of software configuration, much higher than staff can provide. So the staff provides a baseline configuration that I do not touch. I install my own software on top of this baseline, being careful not to change anything in the baseline itself. If I need a baseline change, I ask *them* to make it so that it becomes persistent. In this way my systems are co-managed by myself and my staff in a nearly ideal way, with staff doing what they do best and me doing what I do best. Their side of the social contract is to provide reliability and recovery; my side is not to make their job difficult. They have recovered from complete system failure by building a new system to my baseline requirements, after which I made the few customizations I needed and everything came back up. Thus high synergy can be obtained from proper use of baselining as a basis for a two-way social contract.

REQUIREMENTS AND DESIGN

Another reason that we really need a requirements step in system administration is that specifying requirements clearly leads to better "designs." As in software and systems engineering, in the design step we decide how to configure systems to provide requirements. Design can best satisfy requirements when those requirements are minimally constrained. For example, specifying the hardware composition of a user's workstation in the requirements step is very limiting, especially if the specified machine proves incapable of functions the user requires. It is better to have the option of satisfying requirements by replacing a user's workstation with another physical machine.

Effective design does not just satisfy requirements but also minimizes cost of operations. For example, even if only George needs gcc, it might be easiest to install it everywhere. This is a design decision, while the needs are a requirements decision. This gives other users additional privileges "by design" and not "according to requirements," in order to reduce management cost rather than to satisfy needs. There has been some controversy—especially in defense circles—about providing *any* capabilities to users that they do not need, but in the modern Linux environment, the homogeneity of a common core of software is more or less assumed.

Rethinking the Profession

In summary, I have redefined the system administration process as providing a set of assurances, derived from a refined set of user requirements, augmented with the requirements of our profession, and implemented via baselining and proactive tracking of ambiguities and drift in requirements and assurances. Why go to such trouble?

There are many reasons for documenting requirements. They clarify the actual job of system administrator. They leave one free to assure users of their requirements in the best possible ways. They protect the system administrator from outrageous demands. They appropriately focus discussion upon the mission of the enterprise. Using techniques from systems engineering, they can be used to predict cost of management and suggest an appropriate number of system administrators to hire.

One obvious reason that clear requirements are beneficial is that one can measure objectively whether requirements are met. It has often been said that the better a system administrator is doing, the less people know his or her name. By defining tangible and realistic requirements, rather than broad and sweeping impossibilities, we provide something that can be measured and offer a fairer estimate of system administrator performance than the alternative of remaining anonymous!

In a deep sense, our profession is about "closing open worlds," i.e., creating islands of predictability in which useful work can be accomplished, in an otherwise unpredictable universe. Some islands that we create are due to requirements; others are due to design considerations. Some islands of predictability rise up out of no clear intention on anyone's part! Understanding the landscape of predictability is the real job of the system administrator.

Caveat: That understanding does *not* solve the ongoing and significant problems system administrators have with public relations. A local discount store I frequent has a large sign on the door: "Confusion is our most important product." At present, many users think that this sign describes their system administrators! We need to get to the point where users understand instead that "Peace of mind is our most important product" and that the assurances we make are far more important and crucial than the services we provide.

This article is only a beginning at straightening out some long-term confusion about the profession. We started by defining a taxonomy of tasks, We now must face the harder problem of defining and managing a taxonomy of assurances and expectations. Most important, we have to see "managing systems" for what it is: beating a dead horse. When we can instead "manage assurances," the profession will truly be "at the next level."

REFERENCES

[1] Rob Kolstad et al., "The System Administration Book of Knowledge": http://ace.delos.com/taxongate.

[2] Tina Darmohray, ed., *Job Descriptions for System Administrators*, Short Topics in System Administration 8, USENIX Association, 2001.

[3] Alva Couch, "Should the Root Prompt Require a Road Test?" *;login:*, vol. 32, no. 4, August 2007.

[4] Alva Couch, "From x=1 to (setf x 1): What Does Configuration Management Mean?" *;login:*, vol. 33, no. 1, February 2008.

[5] Alva Couch, "Configuration Management Phenomenology," *;login:*, vol. 35, no. 1, February 2010.

[6] See, e.g., Roger Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. (McGraw-Hill, 2010), chapters 5–7.

WILLIAM K. JOSEPHSON, LARS A. BONGO, DAVID FLYNN, AND KAI LI

DFS: a file system for virtualized flash storage

William Josephson is in the PhD program at Princeton University, where he works with Kai Li. His research interests include high-performance storage and systems support for search in large-scale, high-dimensional data sets.

wkj@CS.Princeton.EDU

Lars Ailo Bongo is a post-doctoral researcher at the Lewis-Sigler Institute for Integrative Genomics at Princeton University. He received his PhD at the University of Tromsø.His research interests include system support for bioinformatics applications.

lbongo@Princeton.EDU

As President and CTO of Fusion-io and one of the company's founders, David Flynn is the visionary behind Fusion-io's innovative technology. Mr. Flynn is responsible for providing business-focused oversight of the company's research and development efforts, as well as driving the company's short- and long-term technological direction.

dflynn@FusionIO.COM

Kai Li is a Paul M. Wythes and Marcia R. Wythes Professor in the computer science department of Princeton University, with research interests in operating systems, parallel and distributed systems, storage systems, and analyzing and visualizing large datasets. He co-founded Data Domain, Inc., which pioneered deduplication storage systems.

li@CS.Princeton.EDU

WHILE FLASH MEMORY HAS TRADI-tionally been the province of embedded and portable consumer devices, there has been recent interest in using flash devices to run primary file systems for laptops as well as file servers. Compared with magnetic disk drives, flash can substantially improve reliability and random I/O performance while reducing power consumption. However, file systems originally designed for magnetic disks are not optimal for flash memory. In this article we examine a flash device used as a disk replacement and how a file system that delegates block allocation to the device driver outperforms the ext3 [15] file system when used with the same device.

Past research work has focused on building firmware and software to support the traditional layers of abstractions used in file systems. For example, techniques such as the flash translation layer (FTL) are typically implemented in a solid-state disk controller that exports a traditional disk drive abstraction [3, 5, 6, 12]. Systems software then uses a traditional block storage interface to support file systems and database systems designed and optimized for magnetic disk drives. Since flash memory has very different performance characteristics from magnetic disks (there is no seek or rotation latency), we wanted to study and design new abstraction layers, including a file system to exploit the potential of next-generation NAND flash storage devices.

We describe the design and implementation of the Direct File System (DFS) and the virtualized flash memory (storage) abstraction layer it uses for FusionIO's ioDrive hardware. The virtualized storage abstraction layer provides a very large, virtualized block-addressed space, which can greatly simplify the design of a file system while providing backward compatibility with the traditional block storage interface. Instead of pushing the FTL into disk controllers, this layer combines virtualization with intelligent translation and allocation strategies for hiding the bulk erasure latencies and performing wear leveling required by flash memory devices.

DFS is designed to take advantage of the virtualized flash storage layer for simplicity and performance. A traditional file system is known to be complex and typically requires four or more years to become mature. The complexity is largely due to three factors: complex storage block allocation strategies, sophisticated buffer cache designs, and methods to make the file system crash-recoverable. DFS uses virtualized storage *directly* as a true single-level store and leverages the virtual to physical block allocations in the virtualized flash storage layer to avoid explicit file block allocations and reclamations. By doing so, DFS uses an extremely simple metadata and data layout. As a result, DFS has a short data path to flash memory and encourages users to access data directly instead of going through a large and complex buffer cache. DFS also leverages the atomic update feature of the virtualized flash storage layer to achieve crash recoverability.

We have implemented DFS for the FusionIO's virtualized flash storage layer and evaluated it with a suite of benchmarks [9]. We have shown that DFS has two main advantages over the ext3 file system. First, our file system implementation is about one-eighth the size of that of ext3, with similar functionality. Second, DFS has much better performance than ext3, while using the same memory and less CPU. Our micro-benchmark results show that DFS can deliver 94,000 I/O operations per second (IOPS) for direct reads and 71,000 IOPS direct writes with the virtualized flash storage layer on FusionIO's ioDrive. For direct access performance, DFS is consistently better than ext3 on the same platform, sometimes by 20%. For buffered access performance, DFS is also consistently better than ext3, sometimes by over 149%. Our application benchmarks show that DFS outperforms ext3 by 7% to 250%, while requiring fewer CPU resources.

NAND Flash

Flash memory is a type of electrically erasable solid-state memory that has become the dominant technology for applications that require large amounts of non-volatile solid-state storage. Flash memory consists of an array of individual cells, each of which is constructed from a single floating-gate transistor. Flash cells support three operations: read, write (or program), and erase. In order to change the value stored in a flash cell it is necessary to perform an erase before writing new data. Read and write operations typically take tens of microseconds whereas the erase operation may take more than a millisecond.

The memory cells in a NAND flash device are arranged into pages which vary in size from 512 bytes to as much as 16KB each. Read and write operations are page-oriented. NAND flash pages are further organized into erase blocks; erase operations only apply to entire erase blocks, and any data that is to be preserved must be copied. There are two main challenges in building storage systems using NAND flash. The first is that an erase operation typically takes about one or two milliseconds. The second is that an erase block may be erased successfully only a limited number of times.

Our Approach

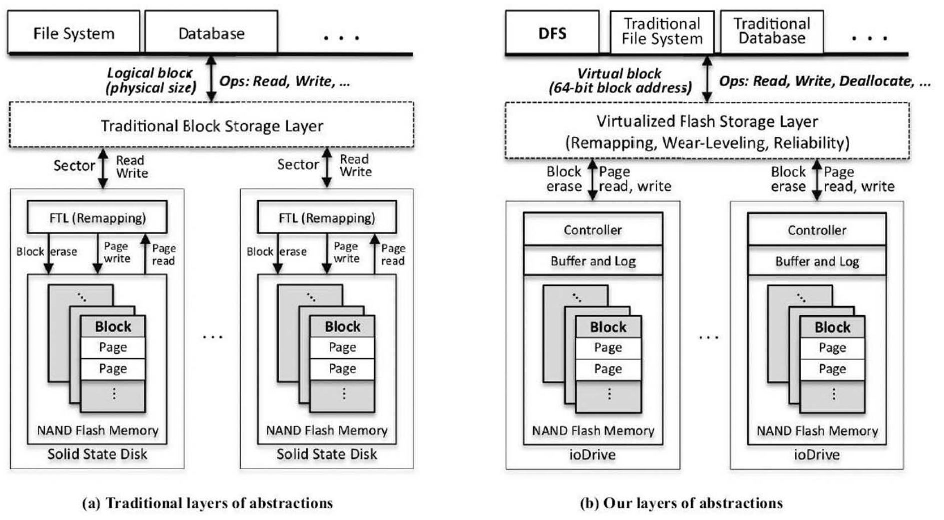

File System
Database
Logical block (physical size)
Ops: Read, Write, ...
Traditional Block Storage Layer
Sector
Read Write
FTL (Remapping)
Block erase
Page write
Page read
Block
Page
NAND Flash Memory
Solid State Disk
(a) Traditional layers of abstractions
DFS
Traditional File System
Traditional Database
Virtual block (64-bit block address)
Ops: Read, Write, Deallocate, ...
Virtualized Flash Storage Layer (Remapping, Wear-Leveling, Reliability)
Block erase
Page read, write
Controller
Buffer and Log
ioDrive
(b) Our layers of abstractions

FIGURE 1: FLASH STORAGE ABSTRACTIONS

Figure 1 shows the architecture block diagrams for existing flash storage systems and our proposed architecture. The traditional approach is to package flash memory as a solid-state disk (SSD) that exports a disk interface such as SATA or SCSI. An advanced SSD implements the flash translation layer in its controller and maintains a dynamic mapping from logical blocks to physical flash pages to hide bulk erasure latencies and to perform wear leveling. SSDs use the same electrical and software interfaces as magnetic disk drives. The block storage layer above the disk interface supports traditional file systems, database systems, and other software. This approach has the advantage of not disrupting the application-kernel or kernel-physical storage interfaces. On the other hand, it has a relatively thick software stack and makes it difficult for the software layers and hardware to take full advantage of the benefits of flash memory.

We advocate an architecture in which a greatly simplified file system is built on top of a virtualized flash storage layer implemented by the cooperation of the device driver and novel flash storage controller hardware. The controller exposes direct access to flash memory chips to the virtualized flash storage layer, which is implemented at the device driver level and can freely cooperate with specific hardware support offered by the flash memory controller. The virtualized flash storage layer implements a large virtual block-addressed space and maps it to physical flash pages. It handles multiple flash devices and uses a log-structured allocation strategy to hide bulk erasure latencies, perform wear leveling, and handle bad-page recovery.

The virtualized flash storage layer can still provide backward compatibility to run existing file systems and database systems. Existing software can benefit from the intelligence in the device driver and hardware. More importantly, flash devices are free to export a richer interface than that exposed by disk-based interfaces.

Direct File System (DFS) is designed to utilize the functionality provided by the virtualized flash storage layer. In addition to leveraging the support for wear-leveling and for hiding the latency of bulk erasures, DFS uses the virtualized flash storage layer to perform file block allocations and reclamations and uses atomic flash page updates for crash recovery. Our main observation is that the separation of the file system from block allocations allows the storage hardware and block management algorithms to evolve jointly and independently from the file system and user-level applications. This approach makes it easier for the block management algorithms to take advantage of improvements in the underlying storage subsystem.

VIRTUALIZED FLASH STORAGE LAYER

The virtual flash storage layer provides an abstraction that allows client software such as file systems and database systems to take advantage of flash memory devices while providing a backward-compatible block storage interface. The primary novel feature of the virtualized flash storage layer is the provision for a very large, virtual block-addressed space. There are three reasons for this design. First, it provides client software with the flexibility to directly access flash memory in a single-level store fashion across multiple flash memory devices. Second, it hides the details of the mapping from virtual to physical flash memory pages. Third, the flat virtual block-addressed space provides clients with a familiar block interface.

The mapping from virtual blocks to physical flash memory pages deals with several flash memory issues. Flash memory pages are dynamically allocated and reclaimed to hide the latency of bulk erasures, to distribute writes evenly to physical pages for wear-leveling, and to detect and recover bad pages to achieve high reliability. Unlike a conventional flash translation layer, the mapping supports a very large number of virtual pages—orders of magnitude larger than the available physical flash memory pages.

The virtualized flash storage layer currently supports three operations: read, write, and trim or deallocate. All operations are block-based operations, and the block size in the current implementation is 512 bytes. The write operation triggers a dynamic mapping from a virtual to a physical page; thus, there is no explicit allocation operation. The deallocate operation deallocates a range of virtual addresses and notifies the garbage collector.

The current implementation of the virtualized flash storage layer is a combination of a closed source Linux device driver and FusionIO's ioDrive special-purpose hardware. The ioDrive is a PCI Express card populated with either 160GB or 320GB of SLC NAND flash memory. The software for the virtualized flash storage layer is implemented as a device driver in the host operating system and leverages hardware support from the ioDrive itself.

The ioDrive uses a novel partitioning of the virtualized flash storage layer between the hardware and device driver to achieve high performance. The overarching design philosophy is to separate the data and control paths and to implement the control path in the device driver and the data path in hardware. The data path on the ioDrive card contains numerous individual flash memory packages arranged in parallel and connected to the host via PCI Express. As a consequence, the device achieves highest throughput with moderate parallelism in the I/O request stream. The use of PCI Express rather than an existing storage interface such as SCSI or SATA simplifies the partitioning of control and data paths between the hardware and the device driver.

The device provides hardware support of checksum generation and checking to allow for the detection and correction of errors in case of the failure of individual flash chips. Metadata is stored on the device in terms of physical addresses rather than virtual addresses in order to simplify the hardware and allow greater throughput at lower economic cost. While individual flash pages are relatively small (512 bytes), erase blocks are several megabytes in size in order to amortize the cost of bulk erase operations.

The mapping between virtual and physical addresses is maintained by the kernel device driver. The mapping between 64-bit virtual addresses and physical addresses is maintained using a variation on B-trees in memory. Each address points to a 512-byte flash memory page, allowing a virtual address space of 2^{73} bytes. Updates are made stable by recording them in a log-structured fashion: the hardware interface is append-only. The device driver is also responsible for reclaiming unused storage using a garbage collection algorithm. Bulk erasure scheduling and wear-leveling algorithms for flash endurance are integrated into the garbage collection component of the device driver.

DFS

DFS is a full-fledged implementation of a UNIX file system that is designed to take advantage of the virtualized flash storage layer. The implementation runs as a loadable kernel module in the Linux 2.6 kernel. The DFS kernel module implements the traditional UNIX file system APIs via the Linux VFS layer. It supports the usual methods such as open, close, read, write, pread, pwrite, lseek, and mmap. The Linux kernel requires basic memory-mapped I/O support in order to execute binaries residing on DFS file systems.

LEVERAGING VIRTUALIZED FLASH STORAGE

We have configured the ioDrive to export a sparse 64-bit logical block address space. Since each block contains 512 bytes, the logical address space spans 2^{73} bytes. DFS can then use this logical address space to map file system objects to physical storage. DFS delegates I-node and file data block allocations and deallocations to the virtualized flash storage layer.

DFS allocates virtual address space in contiguous "allocation chunks." The size of these chunks is configurable at file system initialization time but is 2^{32} blocks, or 2TB, by default. User files and directories are partitioned into two types: large and small. A large file occupies an entire chunk, whereas multiple small files reside in a single chunk. When a small file grows to become a large file, it is moved to a freshly allocated chunk. The size of these allocation chunks and the maximum size of small files can be chosen in a principled manner when the file system is initialized. There have been many studies of file size distributions in different environments (e.g., Tanenbaum et al. [13], Douceur and Bolosky [8]). By default, small files are those less than 32KB.

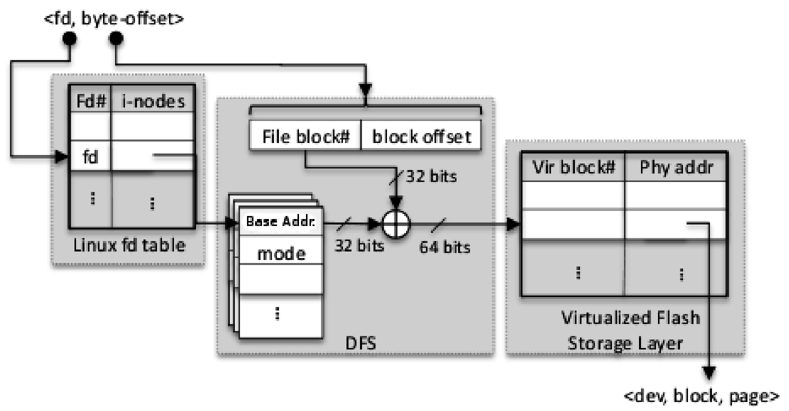


FIGURE 2: DFS LOGICAL BLOCK ADDRESS MAPPING FOR LARGE FILES. ONLY THE WIDTH OF THE FILE BLOCK NUMBER DIFFERS FOR SMALL FILES.

The current DFS implementation uses a 32-bit I-node number to identify individual files and directories and a 32-bit block offset into a file. This means that DFS can support a total of up to $2^{32} - 1$ files and directories (since the first I-node number is reserved for the system). The largest supported file size is 2TB with 512-byte blocks, since the block offset is 32 bits. The I-node itself stores the base virtual address for the logical extent containing the file data. This base address together with the file offset identifies the virtual address of a file block. Figure 2 depicts the mapping from file descriptor and offset to logical block address in DFS.

The very simple mapping from file and offset to logical block address has the added benefit of making it straightforward for DFS to combine multiple small I/O requests to adjacent regions of a file into a single larger I/O. This strategy can improve performance, because the flash device delivers higher transfer rates with larger I/Os.

DFS LAYOUT AND OBJECTS

As shown in Figure 3, there are three kinds of files in the DFS file system. The first file is a system file which includes the boot block, superblock, and all I-nodes. This file is a "large" file and occupies the first allocation chunk at the beginning of the raw device. The boot block occupies the first few blocks (sectors) of the raw device. A superblock immediately follows the boot block. The remainder of the system file contains all I-nodes as an array of block-aligned I-node data structures.

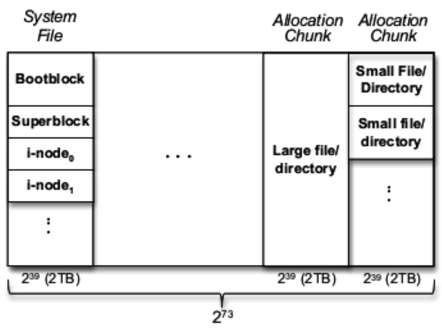


FIGURE 3: LAYOUT OF DFS SYSTEM AND USER FILES IN VIRTUALIZED FLASH STORAGE. THE FIRST 2TB ARE USED FOR SYSTEM FILES. THE REMAINING 2TB ALLOCATION CHUNKS ARE FOR USER DATA OR DIRECTORY FILES. A LARGE FILE TAKES THE WHOLE CHUNK; MULTIPLE SMALL FILES ARE PACKED INTO A SINGLE CHUNK.

Each I-node is identified by a 32-bit unique identifier or I-node number. Given the I-node number, the logical address of the I-node within the I-node file can be computed directly. Each I-node data structure is stored in a single 512-byte flash block. Each I-node contains the I-number, base virtual address of the corresponding file, mode, link count, file size, user and group IDs, any special flags, a generation count, and access, change, birth, and modification times with nanosecond resolution. These fields take a total of 72 bytes, leaving 440 bytes for additional attributes and future use. Since an I-node fits in a single flash page, it will be updated atomically by the virtualized flash storage layer.

The implementation of DFS uses a 32-bit block-addressed allocation chunk to store the content of a regular file. Since a file is stored in a contiguous, flat space, the address of each block offset can be simply computed by adding the offset to the virtual base address of the space for the file. A block read simply returns the content of the physical flash page mapped to the virtual block. A write operation writes the block to the mapped physical flash page directly. Since the virtualized flash storage layer triggers a mapping or re-mapping on write, DFS does the write without performing an explicit block allocation. Note that DFS allows holes in a file without using physical flash pages, because of the dynamic mapping. When a file is deleted, the DFS will issue a deallocation operation provided by the virtualized flash storage layer to deallocate and unmap the virtual space of the entire file.

A DFS directory is mapped to flash storage in the same manner as ordinary files. The only difference is its internal structure. A directory contains an array of name, I-node number, and type triples. The current implementation is very similar to that found in FFS [11]. Updates to directories, including operations such as rename, which touch multiple directories and the on-flash I-node allocator, are made crash-recoverable through the use of a write-ahead log. Although widely used and simple to implement, this approach does not scale well to large directories. The current version of the virtualized flash storage layer does not export atomic multi-block updates. We anticipate reimplementing directories using hashing and a sparse virtual address space made crash recoverable with atomic updates.

DIRECT DATA ACCESSES

DFS promotes direct data access. The current Linux implementation of DFS allows the use of the buffer cache in order to support memory mapped I/O, which is required for the `exec` system call. However, for many workloads of interest, particularly databases, clients are expected to bypass the buffer cache altogether. The current implementation of DFS provides direct access via the direct I/O buffer cache bypass mechanism already present in the Linux kernel. Using direct I/O, page-aligned reads and writes are converted by the kernel directly into I/O requests to the block device driver.

There are two main rationales for this approach. First, traditional buffer cache design has several drawbacks. The traditional buffer cache typically uses a large amount of memory. Buffer cache design is quite complex, since it needs to deal with multiple clients, implement sophisticated cache replacement policies to accommodate various access patterns of different workloads, maintain consistency between the buffer cache and disk drives, and support crash recovery. In addition, having a buffer cache imposes a memory copy in the storage software stack.

Second, flash memory devices provide low-latency accesses, especially for random reads. Since the virtualized flash storage layer can solve the write latency problem, the main motivation for the buffer cache is largely eliminated. Thus, applications can benefit from the DFS direct data access approach by utilizing most of the main memory space typically used for the buffer cache for a larger in-memory working set.

CRASH RECOVERY

The virtualized flash storage layer implements the basic functionality of crash recovery for the mapping from logical block addresses to physical flash storage locations. DFS leverages this property to provide crash recovery. Unlike traditional file systems that use non-volatile random access memory (NVRAM) and their own logging implementation, DFS piggybacks on the flash storage layer's log.

Since flash memory is a form of NVRAM, DFS leverages the support from the virtualized flash storage layer to achieve crash recoverability. When a DFS file system object is extended, DFS passes the write request to the virtualized flash storage layer, which then allocates a physical page of the flash device and logs the result internally. After a crash, the virtualized flash storage layer runs recovery using the internal log. The consistency of the contents of individual files is the responsibility of applications, but the on-flash state of the file system is guaranteed to be consistent.

DISCUSSION

The current DFS implementation has several limitations. The first is that it does not yet support snapshots. The second is that we are currently implementing support for atomic multi-block updates in the virtualized flash storage layer. The log-structured, copy-on-write nature of the flash storage layer makes it possible to export such an interface efficiently. In the interim, DFS uses a straightforward extension of the traditional UFS/FFS directory structure. The third is the limitation on the number and on the maximum size of files.

Evaluation

Application	Description	I/O Patterns
Quicksort	A quicksort on a large dataset	Mem-mapped I/O
N-gram	A program for querying n-gram data	Direct, random read
KNNImpute	Processes bioinformatics microarray data	Mem-mapped I/O
VM Update	Update of an OS on several virtual machines	Sequential read & write
TPC-H	Standard benchmark for decision support	Mostly sequential read

FIGURE 4: APPLICATIONS AND THEIR CHARACTERISTICS

We are interested in answering two main questions:

- How do the layers of abstraction perform?
- How does DFS compare with existing file systems?

To answer the first question, we use a micro-benchmark to evaluate the number of I/O operations per second (IOPS) and bandwidth delivered by the virtualized flash storage layer and by the DFS layer. To answer the second question, we compare DFS with ext3 by using a micro-benchmark and an application suite. Ideally, we would compare with existing flash file systems as well; however, file systems such as YAFFS [10] and JFFS2 [16] are designed to use raw NAND flash and are not compatible with the FusionIO hardware.

	Wall Time		
Application	*Ext3*	*DFS*	*Speedup*
Quicksort	1268	822	1.54
N-gram (Zipf)	4718	1912	2.47
KNNImpute	303	248	1.22
VM Update	685	640	1.07
TPC-H	5059	4154	1.22

FIGURE 5: APPLICATION BENCHMARK EXECUTION TIME IMPROVEMENT: BEST OF DFS VS. BEST OF EXT3

All of our experiments were conducted on a desktop with an Intel quad core processor running at 2.4GHz with a 4MB cache and 4GB DRAM. The host operating system was a stock Fedora Core installation running the Linux 2.6.27.9 kernel. Both DFS and the virtualized flash storage layer implemented by the FusionIO device driver were compiled as loadable kernel modules.

We used a FusionIO ioDrive with 160GB of SLC NAND flash connected via PCI-Express x4 [1]. The advertised read latency of the FusionIO device is 50µs. For a single reader, this translates to a theoretical maximum throughput of 20,000 IOPS. Multiple readers can take advantage of the hardware parallelism in the device to achieve much higher aggregate throughput. For the sake of comparison, we also ran the micro-benchmarks on a 32GB Intel X25-E SSD connected to a SATA II host bus adapter [2]. This device has an advertised typical read latency of about 75µs.

We have evaluated our design and implementation with both a collection of micro-benchmarks and an application benchmark suite. Figure 4 summarizes the applications in the benchmark and their characteristic I/O request patterns. Figure 5 shows the elapsed wall time for each of the applications for both ext3 and DFS and the speedup, which varies from 1.07 to 2.47.

The quicksort application is a single-threaded sort of 715 million 24-byte key-value pairs memory mapped from a single 16GB file that is four times larger than main memory. Although quicksort exhibits good locality of reference, this benchmark program nonetheless stresses the memory-mapped I/O subsystem.

The n-gram benchmark issues random queries against a single large hash table index of the 5-grams in the Google n-gram corpus [7], which contains a large set of n-grams and their appearance counts taken from a crawl of the Web. The resulting index, which contains 26GB worth of small key-value pairs for 5-grams alone, has proved valuable for a variety of computational linguistics tasks. We present the results for a Zipf-distributed query distribution over the 5-grams.

The KNNImpute [14] benchmark program is a very popular bioinformatics code for estimating missing values in data obtains from wet lab microarray experiments. The program is a multi-threaded implementation using memory-mapped I/O.

The virtual machine update benchmark consists of a full operating system update of several VirtualBox instances running Ubuntu 8.04 hosted on a single server. Since each virtual machine typically runs the same operating system but has its own copy, operating system updates can pose a significant performance problem in some environments, as each instance needs to apply critical and periodic system software updates simultaneously. In our benchmark environment there were a total of 265 packages updated, containing 343MB of compressed data and about 38,000 distinct files.

The last benchmark program is the standard Transaction Processing Council's Benchmark H (TPC-H) [2]. We used the Ingres database to run the benchmark at scale factor 5, which corresponds to about 5GB of raw input data and 90GB for the data, indexes, and logs stored on flash once loaded into the database.

Our results show that the virtualized flash storage layer delivers performance close to the limits of the hardware, both in terms of IOPS and bandwidth. Our results also show that DFS is much simpler than ext3 and achieves better performance in both the micro- and application benchmarks than ext3, often using less CPU power. Our paper includes the results of several additional benchmarks, including micro-benchmarks. These results were excluded from this article due to space constraints.

Conclusion

This article presents the design, implementation, and evaluation of DFS and describes FusionIO's virtualized flash storage layer. We have demonstrated that novel layers of abstraction specifically for flash memory can yield substantial benefits in software simplicity and system performance.

We have learned several things from the DFS design process. First, it is possible to implement DFS so that it is both simple and has short, direct-path flash memory. Much of its simplicity comes from leveraging the virtualized flash storage layer for large virtual storage space, block allocation and deallocation, and atomic block updates.

Second, the simplicity of DFS translates into performance. Our micro-benchmark results show that DFS can deliver 94,000 IOPS for random reads and 71,000 IOPS random writes with the virtualized flash storage layer on FusionIO's ioDrive. The performance is close to the hardware limit.

Third, DFS is substantially faster than ext3. For direct access performance, DFS is consistently faster than ext3 on the same platform, sometimes by 20%. For buffered access performance, DFS is also consistently faster than ext3, and sometimes by over 149%. Our application benchmarks show that DFS outperforms ext3 by 7% to 250% while requiring less CPU power.

We have also observed that the impact of the traditional buffer cache diminishes when using flash memory. When there are 32 threads, the sequential read throughput of DFS is about twice that of direct random reads with DFS, whereas ext3 achieves only a 28% improvement over direct random reads with ext3.

REFERENCES

[1] FusionIO ioDrive specification sheet: http://www.fusionio.com/products/iodrive/.

[2] Intel X25-E SATA solid-state drive: http://download.intel.com/design/flash/nand/extreme/extreme-sata-ssd-datasheet.pdf.

[3] Understanding the Flash Translation Layer (FTL) Specification: Technical report AP-684, Intel Corporation, December 1998.

[4]TPC Benchmark H Decision Support (Transaction Processing Performance Council, 2008): http://www.tpc.org/tpch.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proceedings of the 2008 USENIX Annual Technical Conference* (USENIX Association, 2008).

[6] A. Birrell, M. Isard, C. Thacker, and T. Wobber, "A Design for High-Performance Flash Disks," *ACM Operating Systems Review*, vol. 41, no. 2 (April 2007).

[7] T. Brants and A. Franz, Web 1T 5-gram Version 1, 2006.

[8] J.R. Douceur and W.J. Bolosky, "A Large Scale Study of File-System Contents," *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (1999).

[9] W. Josephson, L. Bongo, D. Flynn, and K. Li, "DFS: A File System for Virtualized Flash Storage," *Proceedings of FAST '10: 8th USENIX Conference on File and Storage Technologies* (USENIX Association, 2010), pp. 85–100.

[10] C. Manning, "YAFFS: The NAND-Specific Flash File System," *LinuxDevices.Org,* September 2002.

[11] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, vol. 2, no. 3, August 1984.

[12] A. Rajimwale, V. Prabhakaran, and J.D. Davis, "Block Management in Solid State Devices," unpublished technical report, January 2009.

[13] A.S. Tanenbaum, J.N. Herder, and H. Bos, "File Size Distribution in UNIX Systems: Then and Now," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1 (January 2006), pp. 100–104.

[14] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastieevor, R. Tibshirani, D. Botstein, and R.B. Altman, "Missing Value Estimation Methods for DNA Microarrays," *Bioinformatics*, vol. 17, no. 6 (2001), pp. 520–525.

[15] S. Tweedie, "Ext3, Journaling Filesystem," *Ottowa Linux Symposium*, July 2000.

[16] D. Woodhouse, "JFFS: The Journalling Flash File System," *Ottowa Linux Symposium*, 2001.

JAKE WIRES AND ANDREW WARFIELD

beyond blocks and files

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

Jake.Wires@Citrix.com

Andrew Warfield is an assistant professor in the Department of Computer Science at the University of British Columbia.

andy@cs.ubc.ca

VIRTUAL MACHINES (VMS) CHANGE HOW file and storage systems need to work. Most conventional file systems were designed with the assumption that files would be accessed only through the operating system's file interface. This assumption seemed innocuous when operating systems owned their hardware, but virtual machines use virtual disks owned by virtual machine monitors (VMMs)—and now VMMs want an interface to access VM files too. Presently, VMMs are mostly limited to operating at the block layer, but in order to efficiently provide features such as versioning and deduplication they need to operate at the file system layer. Moreover, the problem of managing large numbers of VMs would be greatly simplified if VMMs better understood files. New file systems designed for use in virtualized operating systems should expose a file interface to VMMs and should better express data dependencies so that files can be safely manipulated from outside VMs.

As fans of virtualization may already well know, too much convenience can be a burden. In an era where the proliferation of real, expensive hardware already frequently motivates "spring cleaning" mass emails from IT departments, the emerging ability of users to spawn virtual machines at their pleasure can lead to managerial headaches. For example, while a system administrator might be quite pleased when she first discovers how easy it is to create a thousand Windows XP VMs, her spirits may falter a bit after she finds that each VM must be customized with individual SIDs and AD credentials if it is to be very useful on the corporate LAN. And she may grow downright frustrated when, a few months after distributing all these shiny new VMs, she finds that every one of them needs to be upgraded—without disrupting any changes users might have made.

Current technologies offer appealing solutions for managing the storage consumed by VMs, but managing the data produced by VMs is still very much an open problem. In many ways, this is an issue of perspective: the advent of VMMs challenges the traditional view that a disk and its files belong primarily to an operating system. The more popular VMs become, the more important it will be to expose OS data to VMMs in meaningful ways.

The familiar debate between block and file-oriented interfaces is no less germane to VMs than physical hardware, although virtualization may add a few new twists. The block interface, as the argument goes, is sublime in its simplicity: it is stateless, straightforward, and OS-agnostic. The file system interface, on the other hand, is often more relevant: it defines much richer storage abstractions and is better aligned with the way users typically reason about their data. This relevance comes at a cost, though:

- File systems are complex and often intricately entwined with other components of the operating system, such as page caches and virtual memory managers.
- File systems must satisfy sophisticated consistency requirements along performance-critical data paths.
- File systems tend to exhibit much greater variation across operating systems.

In general, it is easier for VMMs to interpose on guest VMs at the block layer than at the file system layer. Essential features such as thin provisioning and fast cloning are simple to implement behind the block interface, where they can easily support legacy OSes. Additional features such as versioning and deduplication can be implemented at the block layer as well, although purists might offer arguments for moving these features into the file system. There is very little benefit, for instance, in versioning things like Windows page files and hibernation files—old versions of such files are, for all practical purposes, worthless—but when operating at the block level, it is very difficult to avoid doing so. The upshot in this case is that with block-level versioning, disk snapshots intended to preserve a few kilobytes of user data may end up wasting gigabytes of disk space.

But putting matters of expediency aside, these block-level technologies share a noteworthy characteristic: they all contribute to making a mess of the otherwise simple block layer. While cloning a virtual disk is almost free, merging diverged clones is nearly impossible. Copy-on-write disks provide a quick path to versioning, but they introduce cumbersome dependency chains. Deduplication can reclaim storage space, but it also effectively invalidates disks for use with any tools that don't understand the deduplicator metadata. It may be tempting to ignore these issues when one's main concern is ticking feature check-boxes, but as systems begin to see extended use in the real world, the growing accumulation of interdependent but divergent virtual disks can pose unwieldy problems.

If block-level implementations suffer from such drawbacks, why don't VMMs start plugging into file systems? One major obstacle is that, irrespective of all the hooks and probes and monitors we have thus far attached to VMs, file systems have remained black boxes, and efforts to expose their interfaces to the VMM seem to call for more of the pickax than the scalpel. Even if it is feasible to teach VMMs about the on-disk layout of file systems, this alone would not be enough to provide features such as versioning and deduplication, because of issues such as write ordering and cache consistency within the VM. Interposing on VM file systems is a major effort that would require OS-specific implementations, introduce considerable security risks, and likely require a great deal of maintenance over time as VM file systems grow and evolve.

Such challenges have led to the proposal of new storage abstractions such as object-based disks and file/block hybrids like "flocks" (*not* your standard mutex primitive—perhaps it's inevitable that we'll one day hear clamoring for the widespread adoption of "biles"). These abstractions offer some intriguing new properties. Imagine, for instance, that object-based storage had been adopted 10 or 20 years ago: VMMs would be well positioned to provide features like file-grained versioning and single-instance storage while still hiding behind an arguably tractable, OS-agnostic interface.

But what, after all, is in a name? That which we call a file, by any other name would be as complex. While most OS interfaces are designed to isolate system resources, file systems (and particularly file system namespaces) are peculiar in that they offer opportunities to introduce odd dependencies and circumvent isolation. With a bit of hand waving we can relegate the problem of files to object-based disks, and in so doing we can even congratulate ourselves a bit for better separating storage and namespace implementations, but ultimately we're left with containers of application-level information. If VMMs were able to manipulate these containers they could provide new features to a variety of OSes, but would we really be satisfied?

An especially prickly example here is VM upgrades. Administrators would like the ability to push OS and application updates down onto VM images without disturbing individual users' data. If VMMs recognized file objects, they could enforce read-only or copy-on-write policies for system-administered files, offering greater confidence that these files could be upgraded safely. But it seems doubtful that policies could be derived which would offer users the flexibility they demand while still guaranteeing that their personal customizations would be completely impervious to disruption, direct or otherwise, by system updates. In the end, no matter how transparent the structure of persistent data becomes, there will always be some amount of semantic information that will reside beyond the purview of administrators and limit their ability to safely manipulate VM disk images.

But maybe there are things we can do to mitigate these problems. For starters, the emerging presence of large VM deployments warrants a reevaluation of what a file is and who it is for. Perhaps we should even look beyond blocks and files to see if we can't find better ways of structuring VM semantics. Developing more effective methods of expressing data dependencies and enforcing isolation in the storage stack should be a high priority. As well, new standards of scalability are called for; just as current file systems allow us to manage thousands of files, new storage environments should let us manage thousands of file systems.

ELIZABETH ZWICKY

brute force and ignorance

Elizabeth has been involved with Internet security, voluntarily or involuntarily, since the Morris worm in 1989, but retains hope.

Zwicky@otoh.org

VARIOUS NON-SECURITY BLOGS I READ have been busily urging people to choose good passwords, partly because of the *New York Times* [1] and its coverage of the stupidity of 32 million passwords stolen from RockYou. Now, I wouldn't want to discourage you from choosing a good password. In fact, I think it's a good habit to get into. Go long; stuff some punctuation into the middle; have a good time!

But, honestly, it's a strange thing to worry about based on the RockYou data. The RockYou story goes something like this: RockYou offers services that connect a whole pile of different social networking sites. They had an SQL injection bug. This revealed the contents of not only their main user database, but also the stored information they used to connect to other sites on behalf of users—including passwords for RockYou and other sites, each and every one stored in the clear. RockYou's response to this was, to say the least, underwhelming, although under pressure they did inform users that perhaps it might be a good idea for them to change their passwords.

Meanwhile, Imperva, a security company, laid their hands on RockYou's stolen data, did some analysis of the cleartext passwords, and sent out press releases about the shockingly poor passwords people have chosen, and the success brute force attacks would have against them. This was followed by the wave of admonishments I noted earlier, exhorting people not to choose these terrible passwords.

And, indeed, the data suggest that the passwords were terrible. "123456" was the most popular password, and it was dauntingly popular, accounting for nearly 1% of the passwords. But, you know, it doesn't really matter how useful a brute force attack would have been. Sure, with 683 attempts per account (by Imperva's calculations, which I have no reason to doubt), you could have compromised 10% of the accounts. But that's a lot more effort than it took the attackers to compromise *all* the accounts, with a bonus helping of accounts on other sites. The strength of people's passwords at RockYou was totally irrelevant, and the strength of their third-party passwords was only relevant for those people cunning enough not to hand them over to RockYou.

But, you say, not every Web service is designed by people who are better at fluffy kitten pictures than securing passwords; some of them have already been broken into and now know something about security. Surely at those sites, password strength is good for something other than saving you from public ridicule that ought to have been directed at the people who set free your password in the first place. Well, maybe. But probably not.

The economics of brute force attacks depend greatly on the environment. Brute force is absolutely the way to go if you're attacking a password you have on disk and can fiddle with in the privacy of your own computer. But if you have to try brute force across a wire against a public Web site, you are pitting yourself directly against the site's security. There are two possibilities there. Perhaps the site won't notice, but in that case, it's run by clueless goons, and there's a good chance that the same effort could be invested into attacks with much better payoffs; that was definitely a win for the attackers at RockYou, and it's neither the first nor the last site to have that sort of experience.

And perhaps the site will notice, in which case it's the black hats against the white hats, locked in battle. It's not a battle the white hats can ever win, but they can effectively slow down brute force attacks a lot. Disabling an account altogether is not their only option; they can delay login attempts, they can selectively disable access from individual IP addresses or blocks or specific browser types or cookies, they can insist that the password be changed, they can try to verify that there's a human making login attempts, they can temporarily disable an account, they can send warnings to a contact address, they can arbitrarily change the login process when there are multiple attempts . . . the possibilities are endless.

Meanwhile, the black hats have several fronts where they can pit their cleverness against much weaker opponents. For instance, instead of trying to brute force passwords, they could try to phish for them; there, the white hats are still fighting, but the immediate point of contact is the user, usually a much easier target. Or, the black hats can go attack other Web sites. The effort of breaking into RockYou not only yielded all the RockYou passwords, it also turned up a pile of passwords to other sites, a pile much larger than you could have gathered by attacking the other sites directly.

Brute force attacks against big Web services still exist, of course; attackers are not, on the whole, any brighter than defenders, and old ineffective practices are still rampant on all sides. But on Web services, brute force attacks aren't a major threat, and the current stupidity of passwords isn't enough to skew the economics towards them. There is some level of password stupidity at which brute force starts paying off, and it would be good not to get there, but if you have to pick one lesson to learn from RockYou, it would be, "Don't give away your password." Better yet, learn two lessons; the other one is, "Use different passwords at different sites."

Meanwhile, if you're registering at a site you don't much care about, and you use reasonable passwords at the sites you do care about, why, you have my permission to use "123456" as a password. That way, when the site hands it over on a platter to the miscreants of the Internet, you won't have compromised a password you have some fondness for.

REFERENCE

[1] Ashlee Vance, "If Your Password Is 123456, Just Make It HackMe": http://www.nytimes.com/2010/01/21/technology/21password.html.

DAVID N. BLANK-EDELMAN

practical Perl tools: let me help you get regular

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.

dnb@ccs.neu.edu

DIFFERENT PEOPLE HAVE VERY DIFFERENT opinions of Perl as a language, but I think you might find a healthy majority who agree on the value of the regular expression engine it introduced. You probably get no better vote of confidence in the language world than to have a feature and sometimes its syntax copied almost verbatim. Python, Ruby, Java, .NET, the list goes on, all support some version of Perl-ish regular expressions. There's even a C library called "PCRE—Perl Compatible Regular Expressions," found at http://www.pcre.org/, which can bring that power to your program. Perl 6 aims to introduce further innovation on this front (see http://perlcabal.org/syn/ for more details).

In the meantime, there are a number of ways to make the existing support even more useful and powerful. This column will show you a few Perl regular expression–related modules in that vein.

Regexp Porn

If you want to get really serious about regular expressions, and I'd like to suggest that you do because they are often key to Perl programs, there's one book you need to read. Go buy Jeffrey Friedl's *Mastering Regular Expressions*. I'm not saying this just to shill for a fellow O'Reilly author. The book's a little short on plot and character development, but it is truly the best text on the subject. It will improve your ability to write and understand regular expressions in a number of languages and tools besides Perl (such as awk/grep).

Don't Write Your Own Regular Expressions

Regular readers of this column are familiar with this shtick where I say something is the best thing since split() bread in the first breath and then tell you not to use it in the second, unless . . .

Here's the latest one: don't write your own regular expressions for common items. First check to make sure it isn't already included in the Regexp::Common family of modules. Lots and lots of effort by smart people (certainly smarter than me) has gone into creating a collection of robust, reusable regular expressions for a whole slew of things. In just the Regexp::Common distribution itself, you can find regular expressions for matching:

- credit card numbers
- Social-Economical Numbers (e.g., social security numbers)
- URIs of all sorts
- strings with balanced delimiters
- lists
- IP addresses
- numbers
- profanity
- whitespace
- postal codes

Using Regexp::Common is pretty simple. First you load the module and specify which subset of regular expressions you'd like to use:

```
use Regexp::Common qw /net/;
```

Regexp::Common will then populate a tied hash called %RE that will be filled with the patterns you need. We can then use that hash in the regular expression match of our choice, like so:

```
/^$RE{net}{IPv4}$/ and print "$_ is a dotted decimal IP address\n";
```

The module uses further sub-hash syntax to select more specific options, such as:

```
/^$RE{net}{IPv4}{oct}{-sep => ':'}$/ # matches colon-separated octal IP addresses
```

Many of the pattern sets take an option -keep, as in:

```
$contains_ipaddr =~ /$RE{net}{IPv4}{-keep}/;
```

The -keep option lets you capture all or parts of the match. In this last example, $1 gets set to the full match and $2 through $5 store the components of the match. For example, if $contains_ipaddr was the string 'Your address is 192.168.0.5', $1 would contain 192.168.0.5, $2 would be 192, $3 would be 168, and so on.

And It's a Tie

The following idea is either incredibly useful or it is just a parlor trick, depending on your specific needs. I say that so you'll use it with caution. Mutating the standard hash semantics always makes your scripts a little harder to maintain, because it defies the usual expectations of the code reader. But perhaps it will be worth it to you.

There exist two modules, Tie::RegexpHash and Tie::Hash::Regex, that bring some regular expression magic to your hash data structures. The former lets you write code to store a regular expression as the hash key instead of a scalar. Here's the example from the documentation:

```
use Tie::RegexpHash;

my %hash;

tie %hash, 'Tie::RegexpHash';

$hash{ qr/^5(\s+|-)?gal(\.|lons?)?/i } = '5-GAL';

$hash{'5 gal'};      # returns "5-GAL"
$hash{'5GAL'};       # returns "5-GAL"
$hash{'5  gallon'};  # also returns "5-GAL"
```

Tie::Hash::Regex takes this idea in a different direction. Instead of storing the regular expression as the key, as we just saw, Tie::Hash::Regex first tries the usual exact match during a key lookup. If that fails, it then attempts a regular expression match to find that key. From its documentation:

```
use Tie::Hash::Regex;
my %h;

tie %h, 'Tie::Hash::Regex';

$h{key}      = 'value';
$h{key2}     = 'another value';
$h{stuff}    = 'something else';

print $h{key};    # prints 'value'
print $h{2};      # prints 'another value'
print $h{'^s'};   # prints 'something else'
```

Muchos Matching

There is a class of problems you are bound to run into at some point that entails having to run a (potentially large) number of matches over the same text. For example, if you need to find out if a mail message contains a certain set of keywords, you may find yourself initially writing code that looks like this:

```
my @keywords = qw( urgent acute critical dire exigent pressing serious grave );

foreach my $keyword in (@keywords){
   do_something() if $text =~ /$keyword/;
}
```

If you have a large set of keywords or a large set of repetitions, this gets old/inefficient very quickly, because you are spinning up the regexp engine and forcing it to traipse through the same text over and over again. One standard way to improve on this method is to use regular expression alternation and do a single match on the text, as in:

```
my @keywords = qw( urgent acute critical dire exigent pressing serious grave );
# quotemeta is used to neuter regexp chars in the keyword list
my $match = join '|', map { quotemeta } @keywords;
do_something() if $text =~ /$match/;
```

This is far more efficient even (and especially) if the keyword list is very large. But we can do better than this. The Text::Match::FastAlternatives module is meant to handle exactly this case. It will analyze your list and create a "matcher" which you can use on the text you are checking:

```
use Text::Match::FastAlternatives;
my @keywords = qw( urgent acute critical dire exigent pressing serious grave );
my $keymatch = Text::Match::FastAlternatives->new(@keywords);
do_something() if $keymatch->match($text);
```

People who follow the latest developments in Perl might say at this point, "Wait! But what about the trie-based optimization improvements in 5.10? Don't they make the regexp alternative code we just saw fast too?" It is an excellent question, albeit incomprehensible for those people who don't follow the latest developments in Perl. One of the cool things the Perl developers added in the 5.10 release was some modifications to the regular expression engine that would automatically handle alternation cases like this using a more efficient internal representation. If you use 5.10 and above, you get this speedup for free. Text::Match::FastAlternatives is actually faster than the improved regular expression engine, so it is still potentially the best option for even 5.10+ users. See the Text::Match::FastAlternatives documentation for more details.

But what if we're dealing with something a little more complicated than a list of keywords? What if, instead, we had a set of regular expressions we needed to check against a piece of text? If you need something more in that direction, you would be well served to look at the Regexp::Assemble module. Its documentation says:

> Regexp::Assemble takes an arbitrary number of regular expressions and assembles them into a single regular expression (or RE) that matches all that the individual REs match.
>
> As a result, instead of having a large list of expressions to loop over, a target string only needs to be tested against one expression. This is interesting when you have several thousand patterns to deal with. Serious effort is made to produce the smallest pattern possible.
>
> It is also possible to track the original patterns, so that you can determine which, among the source patterns that form the assembled pattern, was the one that caused the match to occur.

The example from the documentation looks like this:

```
use Regexp::Assemble;

my $ra = Regexp::Assemble->new;
$ra->add( 'ab+c' );
$ra->add( 'ab+-' );
$ra->add( 'a\w\d+' );
$ra->add( 'a\d+' );
print $ra->re; # prints a(?:\w?\d+|b+[-c])
```

Turning on pattern tracking (so you can figure out which regexp matched) is a matter of adding a `track => 1` option to the new() call above and using the source() method. There is one fiddly bit related to pattern tracking and security for people running versions of Perl earlier than 5.10, so be sure to read the documentation before you start to use this feature. When you do consult the docs, you'll discover that the module has a fairly rich set of features. For example, it can read the list of patterns to assemble directly from a file using add_file(). It can also return the assembled pattern as a string so you can store it for later use.

One last Regexp::Assemble tip to mention before moving on to our last module of this column: Regexp::Assemble does a good job of creating "the smallest pattern possible," but another author has written an add-on module called Regexp::Assemble::Compressed which purports to "assemble more compressed regular expressions." It is a subclass of Regexp::Assemble, so you would use it in the same way as its parent module. I haven't had a chance to test it, but you might want to give it a look if smaller results would be helpful.

Do It All at Once

So far we've only talked about using regular expressions for matching purposes. For the last module I'd like to mention, let's consider the other main use of regular expressions: substitution. One cool module you may not have heard of is Regexp::Subst::Parallel, which claims to "safely perform multiple substitutions in parallel." Let's take a simple example of how this could be useful. Imagine we had to change the gender of the English words in a piece of text. If we wanted to do this by running a set of regular expressions against the text, we'd quickly run into trouble if our code looked like this:

```
$text =~ s/\bshe\b/he/;
$text =~ s/\bhe\b/she/;
$text =~ s/\bher\b/him/;
$text =~ s/\bhim\b/her/;
$text =~ s/\bfemale\b/male/;
$text =~ s/\bmale\b/female/;
```

. . . and so on

Ordinarily we'd be forced to switch to a different parsing and transform approach, but Regexp::Subst::Parallel lets us write code that will do the intended substitutions:

```
use Regexp::Subst::Parallel;
my $text = subst($text,
   qr/\bshe\b/      => 'he',
   qr/\bhe\b/       => 'she',
   qr/\bher\b/      => 'him',
   qr/\bhim\b/      => 'her',
   qr/\bfemale\b/   => 'male',
   qr/\bmale\b/     => 'female',
);
```

Hopefully, after this set of tips you are feeling more regular already. Take care, and I'll see you next time.

PETER BAER GALVIN

Pete's all things Sun: open source and free deduplication

IN THE PREVIOUS ISSUE OF *;LOGIN:* I concluded the column with a quick introduction to the new deduplication feature of OpenSolaris. In this issue of "Pete's All Things Sun" I dive deeper into the details of how to gain access to that feature, how it works, and how to use it.

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

pbg@cptech.com

Overview

There is certainly a lot of industry-wide interest in deduplication. Companies like Data Domain (now purchased by EMC) were founded on the premise that companies are willing to add complexity (e.g., appliances) in exchange for reducing the number of blocks used to store their data. For instance, deduplication seems to be a perfect addition to a backup facility. Consider the power of a device that can be a backup target: as it receives blocks of a backup stream, it throws out blocks it has previously stored, replacing that block with a pointer to the duplicate block.

A quick logic exercise of analyzing the types of data that are being backed up should convince you that there is quite a lot of duplication in general (operating system images, binaries, and repeated backups of the same user and application data) and that there is quite a huge potential for savings of disk space via deduplication. Virtual machine images are very deduplicatable, for example, while user and application files are less so. But even when data is not intrinsically duplicated, from the time it is created through its life-cycle there may end up being many, many copies of it. Consider that deduplication can be used as part of a business continuance (disaster recovery) scenario, in which the deduplicated on-disk backup is replicated to a second site. Only sending a given block once can be quite a savings in network bandwidth, as well as the obvious savings of only needing enough storage at the second site to hold one copy of each block.

It's an established pattern in IT that a new feature implemented first by a startup as part of a separate product goes on to become a standard component of other companies' products. That pattern certainly seems true of Sun's implementation of deduplication as part of ZFS, included in an open source and free OpenSolaris distribution. The announcement of the integration of deduplication into ZFS and details of the implementation are available in a blog post by Jeff Bonwick, Sun's lead engineer on the project [1]. I would expect to see deduplication, just like snapshots, thin provisioning, compression, replication, and myriad other features, becoming a component of many storage devices. Thus, even if you are not interested in ZFS deduplication, you may be interested in how deduplication works and what problems it can solve.

How It Works

Deduplication works by "thumb-printing," in which an entity (either a file or a block, typically) is checksummed, resulting in a hash value. Hashing is very effective, providing in almost all cases a unique value for a unique entity. If the values match, the entities are probably the same, and the new entity is not stored; rather, a pointer is stored pointing to the already stored matching entity.

The checksumming occurs at one of two possible times, depending on the implementation. The checksum analysis is overhead, taking up CPU cycles and I/O cycles as an inbound block is checksummed, and that result is checked against the checksums of all other blocks currently stored. For that reason and others, some implementations perform deduplication in post-processing. That is, they store all entities on write request, and then later compare the entities and remove duplicates. That is how NetApp deduplicates on their filers.

Alternately, the deduplication can occur at the time of writing, which is how Data Domain and ZFS deduplication works. This case takes a performance penalty at write time, but does not use up as much space as the post-processing method.

ZFS deduplication, as with other features of ZFS such as compression, only works on data written after the specific feature is enabled. If a lot of data already exists in a ZFS pool, there is no native way to have that deduplicated. Any new data will be deduplicated rather than written, but for the existing data to be deduplicated, that data would need to be copied to another pool (for example) or replicated to a ZFS file system with enabled deduplication.

In ZFS, once deduplication is enabled, the ZFS variable dedupratio shows how much effect deduplication is having on data in a ZFS pool. ZFS has file system checksumming enabled by default. Deduplication uses checksumming too, and enables a "stronger" checksum for the file system when enabled. (,"Stronger" means less likely to have a hash collision. See Bonwick's blog for more details.) By default it uses sha256. As mentioned above, hashing almost always results in matches only when the hashed entities exactly match. But there is a long way between "almost" and "always." Hashes can have collisions in which hashes of two non-matching entities have the same values. In those cases, there could be corruption as one entity is thrown out and replaced by a pointer to the other entity, even though the entities are not the same. See the discussion below about the ZFS deduplication "verify" option for details on how to solve this problem within ZFS.

Getting to the Right Bits

Deduplication was integrated into OpenSolaris build 128. That takes a little explanation. Solaris is Sun's current commercial operating system. OpenSolaris has two flavors—the semiannual supportable release and the frequently updated developer release. The current supportable release is called 2009.06 and is available for download [2]. Also at that location is the SXCE latest build. That distribution is more like Solaris 10—a big ol' DVD including all the bits of all the packages. OpenSolaris is the acknowledged future of

Solaris, including a new package manager (more like Linux) and a live-CD image that can be booted for exploration and installed as the core release. To that core more packages can be added via the package manager.

For this example I started by downloading the 2009.06 OpenSolaris distribution. I then clicked on the desktop install icon to install OpenSolaris to my hard drive (in this case inside VMware Fusion on Mac OS X, but it can be installed anywhere good OSes can live). My system is now rebooted into 2009.06. The good news is that 2009.06 is a valid release to run for production use. You can pay for support on it, and important security fixes and patches are made available to those with a support contract. The bad news is that deduplication is not available in that release. Rather, we need to point my installation of OpenSolaris at a package repository that contains the latest OpenSolaris developer release. Note that the developer release is not supported, and performing these next steps on OpenSolaris 2009.06 makes your system unsupported by Sun. But until an official OpenSolaris distribution ships that includes the deduplication code, this is the only way to get ZFS deduplication.

```
host1$ pfexec pkg set-publisher -O http://pkg.opensolaris.org/dev
opensolaris.org
Refreshing catalog
Refreshing catalog 1/1 opensolaris.org
Caching catalogs ...
```

Now we tell OpenSolaris to update itself, creating a new boot environment in which the current packages are replaced by any newer packages:

```
host1$ pfexec pkg image-update
Refreshing catalog
Refreshing catalog 1/1 opensolaris.org
Creating Plan . . .
DOWNLOAD      PKGS     FILES      XFER (MB)
entire        0/690    0/21250    0.0/449.4
SUNW1394      1/690    1/21250    0.0/449.4
. . .

A clone of opensolaris-1 exists and has been updated and activated. On the
next boot the Boot Environment opensolaris-2 will be mounted on /. Reboot
when ready to switch to this updated BE. You should review the release notes
posted at [3] before rebooting.
```

A few hundred megabytes of downloads later, OpenSolaris adds a new grub (on x86) boot entry as the default boot environment, pointing at the updated version. A reboot to that new environment brings up the latest OpenSolaris developer distribution, in this case build 129:

```
host1$ cat /etc/release
                  OpenSolaris Development snv_129 X86
          Copyright 2009 Sun Microsystems, Inc. All Rights Reserved.
                     Use is subject to license terms.
                       Assembled 04 December 2009
```

At this point, ZFS deduplication is available in this system.

```
host1$ zfs get dedup rpool
NAME     PROPERTY     VALUE     SOURCE
rpool    dedup        off       default
```

Testing Deduplication

Now that we have the deduplication bits of OpenSolaris, let's try using them:

```
host1$ pfexec zfs set dedup=on rpool
cannot set property for 'rpool':
pool and or dataset must be upgraded to set this property or value
```

Hmm, the on-disk ZFS format is from the 2009.06 release. We need to upgrade it to gain access to the deduplication feature.

```
host1$ zpool upgrade
This system is currently running ZFS pool version 22.

The following pools are out of date and can be upgraded. After being up-
graded, these pools will no longer be accessible by older software versions.
VER     POOL
---     ------------
14      rpool

Use zpool upgrade -v for a list of available versions and their associated fea-
tures.

host1$ zpool upgrade -v
This system is currently running ZFS pool version 22.

The following versions are supported:
VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
 4   zpool history
 5   Compression using the gzip algorithm
 6   bootfs pool property
 7   Separate intent log devices
 8   Delegated administration
 9   refquota and refreservation properties
10  Cache devices
11  Improved scrub performance
12  Snapshot properties
13  snapused property
14  passthrough-x aclinherit
15  user/group space accounting
16  stmf property support
17  Triple-parity RAID-Z
18  Snapshot user holds
19  Log device removal
20  Compression using zle (zero-length encoding)
21  Deduplication
22  Received properties

For more information on a particular version, including supported releases, see
http://www.opensolaris.org/os/community/zfs/version/N, where N is the
version number.

host1$ pfexec zpool upgrade -a
This system is currently running ZFS pool version 22.

Successfully upgraded 'rpool'
```

Now we are ready to start using deduplication.

```
host1$ zfs get dedup rpool
NAME    PROPERTY    VALUE    SOURCE
rpool   dedup       off      default
host1$ pfexec zfs set dedup=on rpool
host1$ zfs get dedup rpool
NAME    PROPERTY    VALUE    SOURCE
rpool   dedup       on       local
host1$ zpool list rpool
NAME    SIZE    ALLOC    FREE    CAP    DEDUP    HEALTH    ALTROOT
rpool   19.9G   10.7G    9.19G   53%    1.00x    ONLINE    -
```

To test out the space savings of deduplication, let's start with a fresh zpool. I added another virtual disk to my OpenSolaris virtual machine. Now let's make a pool, turn on deduplication, copy the same file there multiple times, and observe the result:

```
host1$ pfexec zpool list
NAME     SIZE     ALLOC    FREE    CAP    DEDUP    HEALTH    ALTROOT
rpool    19.9G    10.8G    9.08G   54%    1.05x    ONLINE    -
host1$ pfexec zpool create  test c7d1
host1$ zpool list
NAME     SIZE     ALLOC    FREE    CAP    DEDUP    HEALTH    ALTROOT
rpool    19.9G    10.8G    9.08G   54%    1.05x    ONLINE    -
test     19.9G    95.5K    19.9G   0%     1.00x    ONLINE    -
host1$ zfs get dedup test
NAME       PROPERTY    VALUE    SOURCE
test       dedup       off      default
host1$ pfexec zfs set dedup=on test
host1$ zfs get dedup test
NAME       PROPERTY    VALUE    SOURCE
test       dedup       on       local
host1$ df -kh /test
Filesystem    Size    Used    Avail    Use%    Mounted on
test          20G     21K     20G      1%      /test
host1$ ls -l /kernel/genunix
-rwxr-xr-x 1 root sys 3358912 2009-12-18 14:37 /kernel/genunix
host1$ pfexec cp /kernel/genunix /test/file1
host1$ pfexec cp /kernel/genunix /test/file2
host1$ pfexec cp /kernel/genunix /test/file3
host1$ pfexec cp /kernel/genunix /test/file4
host1$ pfexec cp /kernel/genunix /test/file5
host1$ df -kh /test
Filesystem    Size    Used    Avail    Use%    Mounted on
test          20G     14M     20G      1%      /test
host1$ zpool list test
NAME   SIZE    ALLOC   FREE    CAP    DEDUP   HEALTH    ALTROOT
test   19.9G   3.43M   19.9G   0%     4.00x   ONLINE    -
```

So, a file approximately 3MB in size and copied five times to a deduplicated ZFS pool seemingly takes up 14MB but in reality only uses 3.43MB (this space use must include the file, but also deduplication data structures and other metadata).

Also, according to PSARC (architecture plan) 557, deduplication also applies to replication, so in essence a deduplicated stream is used when replicating data [4]. Let's take a look. Fortunately, I have another (virtual) OpenSolaris system to use as a target of the replication (which we will call host2):

```
host2$ pfexec zpool create test c7d1
host2$ pfexec zfs set dedup=on test
host2$ zfs list test
NAME    USED    AVAIL    REFER    MOUNTPOINT
test    73.5K   19.6G    21K      /test
```

Now I take a snapshot on host1 (as that is the entity that can be replicated) and send it to host2:

```
host1$ pfexec zfs snapshot test@dedup1
host1$ pfexec zfs send -D test@dedup1 | ssh host2 pfexec /usr/sbin/zfs
receive -v test/backup@dedup1
Password:
receiving full stream of test@dedup1 into test/backup@dedup1
received 3.30MB stream in 1 seconds (3.30MB/sec)
```

On the receiving end, we find:

```
host2$ zfs list test
NAME    USED    AVAIL    REFER    MOUNTPOINT
test    16.4M   19.6G    21K      /test
host2$ zpool list test
NAME   SIZE    ALLOC   FREE    CAP   DEDUP   HEALTH   ALTROOT
test   19.9G   3.48M   19.9G   0%    5.00x   ONLINE   -
```

Sure enough, ~3MB were sent as part of the replication, and although the receiving system thinks it has ~16MB of date, it only has ~3.4MB.

Unfortunately, the current zfs send -D functionality is only a subset of what is really needed. With -D, within that send, a given block is only sent once (and thus deduplicated). However, if additional duplicate blocks are written, executing the same zfs send -D again would send the same set of blocks again. There is no knowledge by ZFS of whether a block already exists at the destination of the send. If there was such knowledge, then zfs send would only transmit a given block once to a given target. In that case ZFS could become an even better replacement for backup tape: a ZFS system in production replicating to a ZFS system at a DR site, only sending blocks that the DR site has not seen before. Hopefully, such functionality is in the ZFS development pipeline.

Let's try that final experiment. First I'll create more copies of the file, then create another snapshot and send it to host2:

```
host1$ pfexec cp /kernel/genunix /test/file6
host1$ pfexec cp /kernel/genunix /test/file7
host1$ pfexec cp /kernel/genunix /test/file8
host1$ pfexec cp /kernel/genunix /test/file9
host1$ df -kh /test
Filesystem    Size    Used    Avail    Use%    Mounted on
test          20G     30M     20G      1%      /test
host1$ zpool list test
NAME   SIZE    ALLOC   FREE    CAP   DEDUP   HEALTH   ALTROOT
test   19.9G   3.45M   19.9G   0%    9.00x   ONLINE   -
host1$ pfexec zfs snapshot test@dedup2
host1$ pfexec zfs send -D test@dedup2 | ssh host2 pfexec /usr/sbin/zfs
receive -v test/backup2@dedup2
Password:
receiving full stream of test@dedup2 into test/backup2@dedup2
received 3.34MB stream in 1 seconds (3.34MB/sec)
```

Note that, even though host2 already had all the blocks it needed, one copy of the file was sent again because the sending host has no knowledge of what the receiving host already has stored. On the receiving side:

```
host2$ df -kh /test/backup
Filesystem     Size    Used    Avail   Use%    Mounted on
test/backup    20G     17M     20G     1%      /test/backup
host2$ df -kh /test/backup2
Filesystem     Size    Used    Avail   Use%    Mounted on
test/backup2   20G     30M     20G     1%      /test/backup2
host2$ zpool list test
NAME   SIZE    ALLOC   FREE    CAP    DEDUP    HEALTH    ALTROOT
test   19.9G   3.46M   19.9G   0%     14.00x   ONLINE    -
```

Even though host2 was sent the extraneous copy of the file, it discarded it, leaving it to store only one copy of the file.

Additional Analysis

No hash algorithm is perfect, in that two blocks only have the same hash if they are exactly the same. There is a very small chance that two blocks could have matching hashes even if they are not identical. By default ZFS trusts the hash values and will declare a block to be a duplicate if the hash matches. To increase safety you can set ZFS to do a byte-by-byte comparison of two blocks if the hashes match, to ensure that the blocks are identical before declaring them to be duplicates.

```
$ pfexec zfs set dedup=verify rpool
```

Of course this will negatively affect performance, using more CPU time per duplicate block.

On another performance note, the jury is still out on the performance impact of deduplication in ZFS. Theoretically, the increased overhead of checking for an existing matching hash whenever a block is about to be written may be counterbalanced by the saved write I/Os when there is a duplicate block that need not be written. But, in fact, it is too early to tell what the net result will be.

Deduplication can cause a bit of confusion about exactly what is using how much space. For example, the results of du can be grossly wrong if the data in the directories has been well deduplicated. Only zpool list is dedupe-aware at this point. df and even other ZFS commands are not aware of deduplication and will not provide use information taking deduplication into account.

Conclusion

As it stands, ZFS deduplication is a powerful new feature. Once integrated into production-ready operating system releases and appliances, it could provide a breakthrough in low-cost data reduction and management. I plan to track that progress here, so stay tuned. For more details on the current state of ZFS deduplication, including bugs, features, and performance, please see the ZFS wiki [5].

Tidbits

As of this writing, Oracle has just acquired Sun Microsystems. Likely this will mean long-term changes with respect to which of Sun's products come to market and how Sun customers continue on as Oracle/Sun customers. At first blush (and first announcement), however, there seem to be very few changes for Sun customers. There were no massive layoff announcements (as some analysts had predicted), and so far, very little change in product direction. SPARC and x86 servers, storage arrays, Java, and Solaris all appear to have bright futures, as Oracle not only continues those products but increases the R&D budgets for most of them. At least in the immediate shadow of the merger, all seems to be well in Sun's product portfolio and direction. For more details on Sun under Oracle, including replays of the Oracle presentations about the purchase, have a look at http://www.oracle.com/us/sun/.

REFERENCES

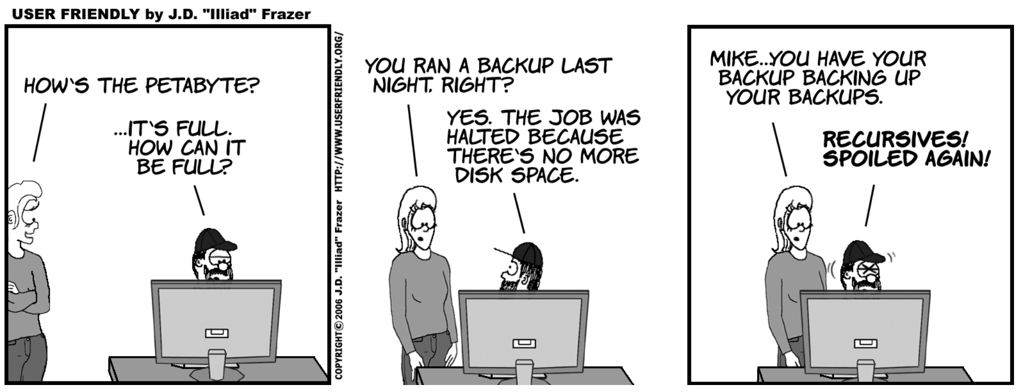
[1] http://blogs.sun.com/bonwick/entry/zfs_dedup.

[2] http://www.opensolaris.com/get/index.jsp.

[3] http://opensolaris.org/os/project/indiana/resources/relnotes/200906/x86/.

[4] http://arc.opensolaris.org/caselog/PSARC/2009/557/20091013_lori.alt.

[5] http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup.


USER FRIENDLY by J.D. "Illiad" Frazer
HOW'S THE PETABYTE?
...IT'S FULL. HOW CAN IT BE FULL?
COPYRIGHT © 2006 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
YOU RAN A BACKUP LAST NIGHT, RIGHT?
YES. THE JOB WAS HALTED BECAUSE THERE'S NO MORE DISK SPACE.
MIKE...YOU HAVE YOUR BACKUP BACKING UP YOUR BACKUPS.
RECURSIVES! SPOILED AGAIN!

DAVE JOSEPHSEN

iVoyeur: a question of scale

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

"MUST I REALLY BE LECTURED EVERY time I want to use the Internet?!" demands my wife, looking at me over the lid of the laptop. This strikes me as the sort of question that warrants a politically savvy answer but I, as usual, am at a loss.

"Sorry?" I reply.

"It's always going on about how the local sysadmin should be lecturing me and what great power I'm being given. There's a word for this, what is it? Oh yes, *annoying*."

Ah-hah. The rest I can guess; she picked up my laptop wanting to get online and, finding it configured for the office network, has opened a term to run the script to reconfigure it for the home network. Upon typing "sudo home eth0", she was presented with sudo's "Are you sure you know what you're doing?" prompt. You've seen it, I'm sure. It looks something like this:

```
We trust you have received the usual lecture
from the local System Administrator. It usually
boils down to these three things:
#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.
```

Great responsibility indeed. On the right box and in the wrong hands all manner of mischief might ensue. Millions of stolen credit cards, mixed up MRIs, disabled battleships, who knows? Sudo doesn't. Linux is a scalable beast if nothing else; we might be running on a wristwatch, or we might be running on . . . well, something really big, but, either way, sudo cautions us with the same message.

Although no doubt annoying to the wives of the world's techno-curmudgeons, it is the mark of great software that it scales beyond the architecture it was intended to run on. There's probably a natural law inherent here; I can only imagine what the spouses of typical VoIP systems engineers have to endure. By this yardstick, however, some classes of software that we sysadmins rely upon daily fall surprisingly short. This being a monitoring column, it shouldn't be hard for the reader to guess the general area on which my gaze is currently falling.

Every monitoring system I've ever worked with, from lofty Open View to humble Big Brother, has scalability problems. So, at least in the context of systems monitoring, I don't think this is necessarily endemic of bad design. Rather, my suspicion, simply stated, is that monitoring 1,000 services is in fact an entirely different problem from monitoring 100,000 services. Like the difference between crossing the Pacific by air and by orbit, the problems are closely related, separated mostly by scale, and require drastically different design considerations. We shouldn't be surprised to find the vehicle designed to make both trips a bit unwieldy.

Even my beloved Nagios has well publicized [1] scalability issues on the high end, but it has one thing going for it that other monitoring systems do not: the Event Broker. Over the years, sysadmins have come up with some pretty kludgy solutions to work around Nagios's scalability problems, so in this month's article I'd like to share with you two (in my opinion) very elegant Event Broker–based solutions for scaling Nagios to very large environments.

DNX

If you attended LISA '09's Nagios Guru session, you met Kyle Martin and Adam Augustine from The Church of Jesus Christ of Latter-day Saints, who spoke in depth about Nagios and Unnoc. What I wish they had been there to talk about, however, was their excellent Event Broker module "DNX" (Distributed Nagios eXecutor) [2].

As you probably already know, a normal Nagios server executes host and service checks by scheduling the execution of small, single-purpose, locally stored programs called plug-ins. These programs may be written in any language and can do any sort of checking, as long as they return standardized output back to the Nagios daemon to interpret. A DNX-enabled Nagios server does pretty much the same thing, with the small exception that just before Nagios executes the plug-in, the DNX event broker module wakes up, and checks to see if any subordinate worker nodes have asked for jobs to execute. If a worker node has asked for a job, the DNX module hands the service check to the worker node instead.

Worker nodes are remote machines in the network that are not running Nagios but have a copy of the Nagios plug-ins and are running the DNX client. The client software is run from init and requests jobs from the central Nagios host on a regular basis. They request and receive a job from the central Nagios server using a network socket and then perform the task and provide the input back to the Nagios server on a different socket.

If a worker node goes down, it, ipso facto, stops requesting jobs from the Nagios server. If a worker node goes down after it's been given a job and before it returns the status of that job, the check will time out and Nagios will reschedule it. If all the worker nodes go down, the DNX module will not have any available worker nodes to hand jobs to, and Nagios will operate as it normally does.

The Church reports running 2000 checks per minute with their DNX setup (10,000 checks in a five-minute interval) [3], and they feel it could go much higher. Steven Morrey reported to the Nagios-devel list that the Church's Nagios daemon spends two-thirds of its time reaping check results from the results ring-buffer [4], which is promising news. A DNX-enabled Nagios server is limited mainly by the speed of the reaper process.

DNX requires no modification to your existing configuration files beyond the addition of a single line to your Nagios.cfg to load the module. No replication of configuration to the client nodes is required, as there would be to configure passive checks. The clients need to know where the server is, the server needs to know what clients are allowed to connect, and that's about it; configuration is as easy as it gets for a Nagios box. Simple, efficient, and elegant, DNX is such great design that it makes me want to buy those folks beer . . . or whatever it is that's analogous to beer in their universe.

OP5 Merlin

Merlin (Module for Endless Redundancy and Loadbalancing In Nagios) is, by comparison, a fair bit more difficult to describe. This is because its goals are—despite the name—appreciably loftier than redundancy and load balancing [5].

If you've read any of my previous articles about the Event Broker, then you know the Broker allows you to hook into pretty much any aspect of a running Nagios daemon by allowing your module to register for callbacks, which trigger when an event happens. The expectation is that a given module will register for the event types it is interested in, and do something useful with the event callbacks it's given. DNX, for example, registers for the NEBCALLBACK_SERVICE_CHECK_DATA callback and uses the NEBTYPE_SERVICECHECK_INITIATE event to preempt service check execution and insert its own load-balancing framework.

Rather than registering for a particular callback and writing event handler functions inside the module, Merlin registers for *all* callbacks and exports them all to an external daemon to handle. The Merlin daemon gets events from the Merlin module inside the Nagios daemon and either sends them to other Merlin daemons on other systems or to a database of your choosing. Events that come from other Merlin daemons can be injected back into a running Nagios daemon via the Merlin Event Broker module.

So there are two very powerful things that Merlin makes possible. The first is database synchronization (and a far better, more usable DB synchronization than NDOUtils, in my opinion), which in turn enables all manner of third-party UIs, add-ons, data export, and backup scenarios. The second, and more topical for our current purposes, is load balancing, clustering, and failover. With Merlin, it's possible to update the state of one Nagios daemon with events generated by another, remote Nagios daemon. In fact, the Merlin developers describe Merlin as a "cross-host event transportation layer," and this is an accurate description. Indeed, given the extent to which Nagios stores state data in memory, I find myself thinking of Merlin as a cluster shared memory system reminiscent of the SGI Onyx.

The possibilities here are pretty interesting. DNX-like arrangements may be created where "master" Nagios daemons send their checks to subservient Nagios daemons for processing, peering clusters may be set up where two or more Nagios daemons cooperate and update each other, or various permutations of the two may be achieved. A single Nagios daemon, for example, may be a peer to its peers, a master to its nodes, and a node to its masters, all at the same time. More complex relationships are also possible whereby, for example, a rollup server in Chicago might collect and display the state of remotely administered daemons in India, Brazil, and Australia.

I'm not able to find any solid information on the practical upper limits of a "Merlinized" Nagios cluster. The op5 folks seem to believe protocol overhead to be the primary limiting factor. An interesting question (at least to me) is whether DNX and Merlin could help each other scale. For example, one could imagine a DNX cluster with multiple master nodes sharing the reaper load via the Merlin protocol. Such an arrangement would minimize the overhead wrought by the Merlin protocol as well as share the reaper load, while at the same time helping to minimize the amount of configuration necessary, since DNX doesn't require full Nagios installations for load balancing the way Merlin does.

I've been really excited about both of these projects for a while now. They're both just the kind of great tools I envisioned when the Nagios Event Broker interface was first implemented, and I look forward to the day that add-ons like this are the norm.

I should get going—my wife just got her first Nagios pages about the Pepsi supply falling below the warning threshold.

Take it easy.

REFERENCES

[1] Carson Gaspar's invited talk at LISA '07: "Deploying Nagios in a Large Enterprise Environment": http://www.usenix.org/media/events/lisa07/tech/videos/gaspar.mp4.

[2] DNX Home: http://dnx.sourceforge.net/.

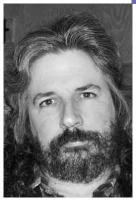
[3] About DNX: http://dnx.sourceforge.net/about.html.

[4] Nagios dev archives: http://archive.netbsd.se/?ml=nagios-devel&a=2009-09&t=11599144.

[5] Merlin: http://www.op5.org/community/projects/merlin.

ROBERT G. FERRELL

/dev/random: less successful network protocols



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

rgferrell@gmail.com

I SPENT SOME TIME IN THE ’90S WORKing with the IETF, and during those heady formative years of the public Internet there were a number of ideas that, for one reason or another, never truly made it into the technology mainstream. We can only speculate over a biscotti and scalding hot chai latte whatever became of the sincere but misguided proponents of these ill-fated engineering marvels. My guess would be the Senate Banking Committee.

Nippon Transport Protocol: A well-designed, robust competitor for TCP/IP with one fatal drawback: instead of exchanging handshake packets during connection negotiation, the nodes merely bowed to one another. For reasons never fully elucidated, the negotiation was considerably facilitated by the presence of raw seafood and rice wine.

Tokin’ Ring Interface Protocol: An early entrant in the LAN collision-avoidance arena, Tokin’ Ring suffered from two basic issues: the nodes involved often forgot which one of them was supposed to be broadcasting, and the ones who did broadcast tended to send the same packet out multiple times. Fragmentation and reassembly were mostly randomized, which had rather deleterious effects on information integrity. This topology was also highly susceptible to a specific race condition known as “The Munchies.”

SewNET: This protocol actually considerably predates the digital era, but was formalized as an emergency information sharing topology in the event of widespread failure of the telecommunications infrastructure due to natural disaster, coordinated terrorist attack, or forgetting to pay the phone bill again. It relies on the apparent quantum phenomenon known colloquially as “gossip entanglement,” whereby information provided to one sewing/knitting circle (also works with most book clubs) is simultaneously shared with all of them, no matter how widely dispersed geographically.

Interurban Coordinated Ballistic Messaging: An update to the messenger pigeon concept (see RFC 1149) using model rocketry. Bandwidth was rather limited in the early days, but with the advent of flash memory and other solid-state storage technologies this isn’t really an issue any longer. This protocol is still decidedly connectionless, and quite frankly makes UDP look like the epitome of reliable communication, but the adrenalin factor is unsurpassed in the metropolitan area networking arena.

Dynamic Hostess Control Protocol: Developed for use in Internet cafes, this protocol provided a weighted hierarchical scheme for ensuring that wait staff were evenly distributed among patrons, even during shift changes. It had no provisions for authentication or data integrity, however, and the inevitable competition among hungry/thirsty hackers that quickly ensued hopelessly clogged the control channel with conflicting instructions. This was the original denial of service attack. As a result, a related and salutary technology, Tip Calculation Protocol, sadly never really matured.

Trivia Data Transfer Protocol: A protocol broadly employed by social networking sites, designed especially for handling acronyms, abbreviations, emoticons, and the abomination that is tweeting. Collectively, these fall under the ASSKEY text standard.

NoTelnet: An experimental database protocol in use for a while by certain segments of the hospitality industry and characterized by randomly assigned last name fields hard coded with "Smith," "Jones," or, in the international version, "Patel." Eventually discarded at the request of the law enforcement community. Elements of the technology were later incorporated into generating the "no-fly" list adopted by the TSA.

Internet Chaff Relay: The short-lived predecessor to Twitter; see TDTP.

NutBIOS: A file-sharing protocol developed for use in the conspiracy theory community. Allows for user-generated handles such as NoLoneGunman, FakedMoonLanding, and ClimateChangeIsALie. Not compatible with smart cards.

Lightweight Dumb User Management Protocol: A simplified protocol intended to automate certain system administration tasks and provide canned responses to a wide variety of common requests from userland. Contains preset functions for issuing new passwords, finding the power button, restarting services killed by user error, and, most importantly, a randomized list of interesting Web sites to distract users when they get a "Please Contact Your System Administrator" error message.

Chaotic Resolution Addressing Protocol: A malicious protocol installed on compromised routers and gateways that arbitrarily changes destination addresses on incoming packets originating from machines with Webcams and posts the resulting puzzled/angry/oblivious user videos to share sites. Harbinger of the Antisocial Networking movement.

File transfer Incorporating Super Heterodyne Networks (FISHNet): An eccentric attempt to use nearby AM radios for file sharing. Difficulties encountered included the need to install tuning capacitors into modems and susceptibility to electrical interference from vacuum cleaners, blenders, and garbage disposals. Abandoned when it was realized that files with adult content could be inadvertently intercepted by neighborhood children with dental appliances.

Nethernet: A protocol employed by certain elements of the computer underground, mostly for bragging about their latest lame Web defacements, asking each other for keys to pirated software, and pretending to have read and understood anything in *The Anarchist Cookbook*. Finally brought to its knees by script kiddies whining, "Teach me to be a hacker!!!!"

IPSex: On sober reflection, this probably isn't the proper forum for discussing this one. I'll be happy to host a technical poster session and BoF roundtable during happy hour at the Lion and Rose next Friday.

book reviews

ELIZABETH ZWICKY
WITH BRANDON CHING

A PRACTICAL GUIDE TO LINUX: COMMANDS, EDITORS, AND SHELL PROGRAMMING

Mark G. Sobell

Prentice Hall, 2010. 988 pp.
ISBN 978-0-13-136736-4

This book is ideal for somebody who's bright, motivated, and ready to go beyond the GUI on Linux or OS X, particularly if they also need to deal with those of us who are not so much beyond the GUI as before the GUI. I could quibble with a large number of its choices, but mostly I'm arguing about matters of taste, and about the inevitable compromises that are made when you try to cram a really large set of stuff with poorly defined boundaries into a book, which has to start somewhere, end somewhere else, and still be possible to lift. If you've got people around who can learn things from books and who really need to be able to cope with the magical world of the UNIX command line and twisty mazes of pipes as they were intended to be, this is a great book to have around.

Here are some of the important choices the author made: first, the book is as platform-agnostic as possible. That means there are fascinating platform-dependent features that aren't covered. If you want to know every bell and whistle, you'll need something specific to your platform, and probably to some small set of releases. I don't see that as a big deal, because I think the important things to start with are the things that carry across platforms.

Second, the book covers commands, not internals. This is a hard line to draw, and in some cases I think this is problematic. Without understanding something about file systems, it's hard to make any sense of hard and soft links, or of holes in files. The author does his best, but it's often not exactly correct, or it's confusing. I'm not sure there's any winning this one, as many of the people who most need to understand the internals think of them as irrelevant or intimidating, and talking about internals only worsens the problems with cross-platform compatibility.

Third, the book covers programming, without assuming any programming experience. I think this is a valiant effort, but I'm dubious about how well it's going to serve most naive readers. If you don't understand if-then constructs, you're going to need more help than this book can offer, and you probably ought to learn to program in something, anything, before you dive into writing shell scripts. On the other hand, there are some brave, even foolhardy, souls out there (I may have been one of them) who can actually use this sort of thing.

I would have chosen a different set of commands to cover (personally, I've always found dc more useful than sed, not that I've used either in years), but that's very much a question of taste. Even more pettily, I twitch every time I see "TC shell." I don't suppose there's a solution a copy editor won't whine about, but "T C shell" would at least get across the idea that it's a kind of C shell.

But, as I said, this is all quibbling. Fundamentally, it's a strong book that goes a long way toward bridging the gap between good old-fashioned UNIX hackers and those whippersnappers who only know Linux or Macintoshes.

INSIDE CYBER WARFARE

Jeffrey Carr

O'Reilly, 2010. 205 pp.
ISBN 978-0-596-80215-8

Here's another book that faces some nasty challenges. In the case of cyber warfare, the problem is that it is difficult to know anything with certainty, and what you do know it is probably unethical to talk about. For instance, if you know for certain that our country is vulnerable to certain sorts of attacks, how much can you say about that? What's the line between proving that you do know what you're talking about, and enabling idiots to cripple vital national infrastructure? Note that any security practitioner of a reasonable degree of expertise knows at least one way to cripple vital national infrastructure without unreasonable expenditure of resources, and most of them have been known to hang out in groups chatting about these things. But one must assume that there are lots of ill-intentioned people out there who don't know how, and you wouldn't want to draw them a map.

Then there's the question of just who is attacking whom and why. It's really not in the best interest of any ill-intentioned party for this to be clear. Who's in it for the money? Who's in it for politics? Who's linked to a government? Nobody wants you to know this. If you're in it for the money, it's to your advantage to make people think you have political goals and government sponsorship. If you're in it for the politics, you probably want to hide among the garden-variety criminals. If you're a government sponsoring attacks, you *really* don't want anybody to know. And if you've found out for certain? Well, how much would you like to annoy some political terrorists, organized crime, and a hostile government or two? Plus, quite likely, all the people trying to foil them? That's a lot of people with the knowledge and ability to use various kinds of unpleasant force against you, so you don't have a lot of incentive to go telling people what you know.

On the plus side, spies are kind of fascinating, and attacks against networks are an immediate threat to almost everybody, so there's a lot of potential interest in a book about cyber warfare. Unfortunately, this book doesn't manage to make the most of that potential. It does manage to make a convincing case that cyber warfare exists, mostly in the form of collusion between governments and non-governmental entities, and that the laws of war allow nations to do stuff about it. But it's not a fun ride, for both editorial reasons and technical ones.

The book is patched together from a number of sources (the author might more reasonably be described as a contributing editor, since whole chapters are written by other people). The seams show, often badly, in the form of differing tone, style, and background assumptions, and in the lack of an overarching structure to the book. Worse yet, there are straightforward editorial errors, such as content repeated between chapters apparently unintentionally.

There are also some significant technical flaws. It is not fair to say that anti-virus solutions are always based on signature detection; there are heuristic-based solutions available now, which work by looking at behavior rather than signatures, and that hardly begins to scratch the surface of what's possible, particularly in secured environments. Similarly, complaining that Microsoft Word's binary format is bad because you can't detect hostile content by human visual inspection is just silly. Trust me, it could say "VIRUS HERE" in the source, and nobody would notice.

Personally, I think there are important lessons here for various people I know who are worried about controlling corporate environments against government-sponsored spying. First, even in military environments, people are terrible about information security. That is, people who have been made to sit in small rooms while people with guns tell them about the vital need for secrecy still discuss their jobs on Facebook. You might as well give up on getting your employees to keep their employer a secret. (Particularly if you give your employees things to wear in public with your logo on them.)

Second, governments do a lot of their cyber attacks via third, fourth, and fifth parties, who are the same sometimes-smart sometimes-not crew who bring you spam. You can stop worrying about whether or not you're being attacked by a government, because you're never going to know. On the other hand, there's also no point just deciding you can't defend against a government, because most of the time, they're not going to be any brighter or better resourced than the usual range of attackers.

All of this is in the book, but this is my interpretation, which doesn't particularly relate to the author's. To the extent that the book has a clear audience, it's aimed at government, rather than business.

INTRODUCING STATISTICS: A GRAPHIC GUIDE

Eileen Magnello and Borin Van Loon

Totem Books, 2009. 174 pp.
ISBN 978-184831056-8

Another entry into the list of "statistics through pictures" books. *Head First Statistics* is still my favorite for practical statistics, but this is a fun tour of statistics, concentrating on history but picking up the numbers on the way past. The illustrations are both practical and amusing, the examples are mostly drawn from real data, and it covers the important common statistical flaws. It's a whirlwind tour rather than an in-depth introduction, and it covers a lot of territory, so it's easy to scan quickly without absorbing. If you're looking for something non-threatening that gives you a sense of the historical background, this is a fun choice, but you'll need more help to actually run the numbers.

HARD FACTS, DANGEROUS HALF-TRUTHS & TOTAL NONSENSE: PROFITING FROM EVIDENCE-BASED MANAGEMENT

Jeffrey Pfeffer and Robert I. Sutton

Harvard Business School Press, 2006. 264 pp.
ISBN 978-1-59139-862-2

This is quite possibly the geekiest business book ever, because its main thesis is that you ought to actually care about data when deciding how to do management. Interestingly, the net result of paying attention to data is that you prefer a management style that involves being nice (for some definition of nice). It's hard to avoid the desire to be intolerably snide about some of this, since it is overwhelmingly pleasant to be able to say, "Actually, people have done studies, and behaving like a reasonable human being? It's not just a good idea! It's what actually works!"

Get this book if you have the nagging feeling that there is something horribly wrong with your management culture, particularly if what's bothering you is an insatiable desire for apparently pointless change, or a rigid adherence to the idea that one must reward the top performers and get rid of the bottom ones, always, in every group no matter how small and how talented. It will not necessarily enable you to change your company, but at least you will no longer feel alone, or believe that this is your problem. There are people in suits with degrees in management who feel just like you and are at least as angry about it, only they have more research and a publisher.

WEB DESIGN FOR DEVELOPERS: A PROGRAMMER'S GUIDE TO DESIGN TOOLS AND TECHNIQUES

Brian P. Hogan

Pragmatic Bookshelf, 2009. 311 pp.
ISBN 978-1934356135

REVIEWED BY BRANDON CHING

Few people outside of Web development understand the difference between a Web designer and a Web developer. This distinction, while subtle in conversation, is considerable in application. Web developers like myself generally do not do design very well. We know what looks good, but we won't always be able to make an application as pretty as someone with an eye for art and a heart for design.

Enter Brian Hogan and *Web Design for Developers*. Unlike general Web design and CSS books, *Web Design for Developers* takes a look at approaching design through the eyes of a programmer. This isn't a CSS handbook or a high-in-the-sky artsy design guide; it is a no-nonsense guide to the basic principles and techniques that make for visually appealing Web applications.

The author's writing is very approachable and to the point, yet full of enough content and whim to keep it interesting. Under the guise of redesigning a recipe-sharing application, the author guides you through the process of beautifying an existing site as opposed to designing one from scratch. The text is full of references to Web sites, design guides, and helpful utilities, such as color tools, which will aid any aspiring artistic programmer.

The book has four sections: The Basics of Design, Adding Graphics, Building the Site, and Preparing for Launch. The Basics of Design covers the fundamentals of requirements gathering and design planning. The first two chapters deal with client communication procedures and the basics of idea generation and requirements gathering. While not intended to be all-inclusive or in-depth, the coverage should be sufficient for most developers. Remember, this book is geared towards professional developers who are assumed to already have at least a rudimentary grasp of how to tease requirements from clients.

The remaining two chapters of this section quickly get to the heart of what most artistically challenged developers are after: style, and I don't mean CSS. Hogan presents two fluid and fundamental chapters on colors and fonts. He breaks down basic artistic design principles such as color schemes, color mood, font types, and font selection. While not terribly interesting, colors and fonts are the meat and potatoes of good Web design, and Hogan does a good job of conveying what you need to know about both.

Weighing in at a light 40 pages, Adding Graphics is a short and to-the-point introduction to using Photoshop to generate the logo and mock-up for the site. This is not the place to learn Photoshop techniques for the Web, but that is not the intention of this section. Rather, Hogan provides formatting and layout principles of mock-ups while also introducing the Photoshop tools and techniques (such as layering and masks) that you will need to reproduce them.

The third section, Building the Site, is where we finally break into the code! Now, given that this book is dedicated to professional developers, I'm not entirely sure why there is an entire chapter dedicated to HTML tags and compliance. These are things that the book's target audience should already be intimately familiar with. However, the next few chapters in this section are quite handy, in that they deal with asset creation from the mock-up generated earlier in the book, the integration of CSS to define the layout, CSS/browser compatibility issues, and even a chapter on printer-friendly tweaks.

The final section, Preparing for Launch, is a hodgepodge of topics that any designer or developer needs to be aware of. Topics include a more in-depth analysis of browser compatibility, search engine optimization, accessibility and usability, mobile device support, testing and performance, and even a brief chapter on favicon creation. The chapters on mobile content and browser compatibility are good, but the chapter on accessibility and usability is exceptional, covering issues such as color blindness and hearing and motor impairment and telling you what you need to do about them.

Overall, *Web Design for Developers* is a bookshelf-worthy buy. While the title suggests professional developers as the target audience, I would venture to say that it would be more useful to novice and intermediate-level developers, as a significant portion of the book covers material that most professional developers would already know. However, the principles of design, general designer tips, layout and mock-up techniques, color and font selection, and the accessibility and usability sections are definite jewels that artistically challenged developers should at least take a look at.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

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

DISCOUNTS on registration fees for all USENIX conferences.

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

THE RIGHT TO VOTE on matters affecting the Association, its bylaws, and election of its directors and officers.

FOR MORE INFORMATION regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

IN MEMORIAM: LEWIS A. LAW, 1932–2010

Lou Katz, Founding President, USENIX Association, and Peter H. Salus, former Executive Director, USENIX Association

It was with great sadness that we heard of the death of Lew Law, who served as the first Secretary of the USENIX Association. When the first organizational meeting was held, at Columbia University in May 1978, Lew was selected as a representative of universities, as was Mel Ferenz. Mars Gralia was selected to represent government labs, and Peter Weiner came from a commercial UNIX site. Lou Katz was the fifth wheel, selected to give the organizing board an odd number of (equally odd) people. Lew was elected chairman of this organizational committee.

Ultimately, Mel was made Treasurer of the nascent organization, Lew received the role of Secretary, and Lou Katz was elected President. In his role as Secretary, Lew was an amazing asset. His meeting minutes were not merely simple recitals of those present and of votes on motions. Rather, they summarized in a very readable form the discussions that had taken place, so that readers could see not only what decisions had been taken but what considerations had led to the decisions.

The committee of which Lew was voted the chairman had "the purpose of proposing a set of bylaws for an organization of users of UNIX* installations. . . . The name of the committee shall be the USENIX** committee." The name USENIX was coined by Lew's wife, Margaret.

Lew died on Sunday, February 14, 2010, after a long struggle with Alzheimer's disease. He was 77. Born June 18, 1932, in Rubery, England, he attended Kings Norton Grammer School in Birmingham and graduated from Birmingham University with a B.S. in Physics in 1953 and from Northeastern University with a Master's degree in Electrical Engineering in 1972. Prior to Lew's position as Assistant and then Associate Director at the Harvard University Science Center, he was the head of the Electronics Group at the Harvard/MIT Cambridge Electron Accelerator. He started the computer group at the Science Center in 1975; when he finally retired, he was Director of Computer Services for the Faculty of Arts and Sciences. He and Margaret had been married for 52 years.

Lew had a wicked sense of humor. In the paragraph above, his footnotes to the asterisks read:

* UNIX is a trademark of Bell Laboratories, Inc.

** USENIX is not a trademark of Bell Laboratories, Inc.

After frustrations with AT&T lawyers about the UNIX manuals, Lew had announced in *;login:*, 30 April 1976, that he was "willing to undertake the task of duplicating and distributing the manuals for UNIX . . . The 'UNIX PROGRAMMER'S MANUAL' Sixth Edition dated May 1975 will be reproduced in its entirety." Two years later, in March 1978, Lew announced the availability of the PWB [Programmer's Workbench] manuals in four volumes (at $26.50!).

Lew served on the USENIX Board of Directors until 1986, and he was active at meetings for another decade.

I (Lou Katz) have always considered Lew to be the rock upon which USENIX was built. I could always rely upon him to give clear and useful advice and not get entangled in the idiosyncrasies, personalities, or irrationalities of the players, and many times he helped me to get past/through/around ridiculous, annoying, or infuriating situations. I last saw Lew at a USENIX conference years ago, when his memory problems were weighing him down but hadn't yet knocked him out.

We miss him.

CALLING ALL BLOGGERS

Anne Dickison, Marketing Director, USENIX

USENIX is looking for experienced bloggers to contribute to the official USENIX blog. Everyone, from university students to blogging professionals, is encouraged to apply.

We're looking for bloggers with experience in technical writing. We especially welcome those with expertise in system administration, software engineering, security, virtualization, green IT, file and storage systems, Web development, or cloud computing.

All participants will receive a discount on one conference of their choice, as well as the opportunity to post a bio link on the USENIX blog team bio page. Build your portfolio and help spread the word about the latest developments in systems computing.

To apply, please send usenixbloggers@usenix.org a technical writing sample and a brief statement telling us why you would like to be a part of the USENIX blogging team and two topic areas you would like to cover.

Thanks to USENIX and SAGE Corporate Supporters

USENIX Patrons
Facebook
Google
Microsoft Research

USENIX Benefactors
Hewlett-Packard
IBM
Infosys
Linux Journal
Linux Pro Magazine
NetApp
VMware

USENIX & SAGE Partners
Ajava Systems, Inc.
BigFix
DigiCert® SSL Certification
FOTO SEARCH Stock Footage and Stock Photography
Splunk
SpringSource
Xssist Group Pte. Ltd
Zenoss

USENIX Partners
Cambridge Computer Services, Inc.
GroundWork Open Source Solutions
Xirrus

SAGE Partner
MSB Associates

Save the Date!

LISA '10

24TH LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE

November 7–12, 2010, San Jose, CA

SPONSORED BY USENIX IN COOPERATION WITH LOPSA and SNIA

Photo courtesy of the San Jose Convention & Visitors Bureau

Come to LISA '10 for training and face time with experts in the sysadmin community.

LISA '10 will feature:

6 days of training on topics including:

- Virtualization
- Security
- Configuration management
- And more!

Plus a 3-day Technical Program:

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

New for 2010!
Practice and Experience Reports

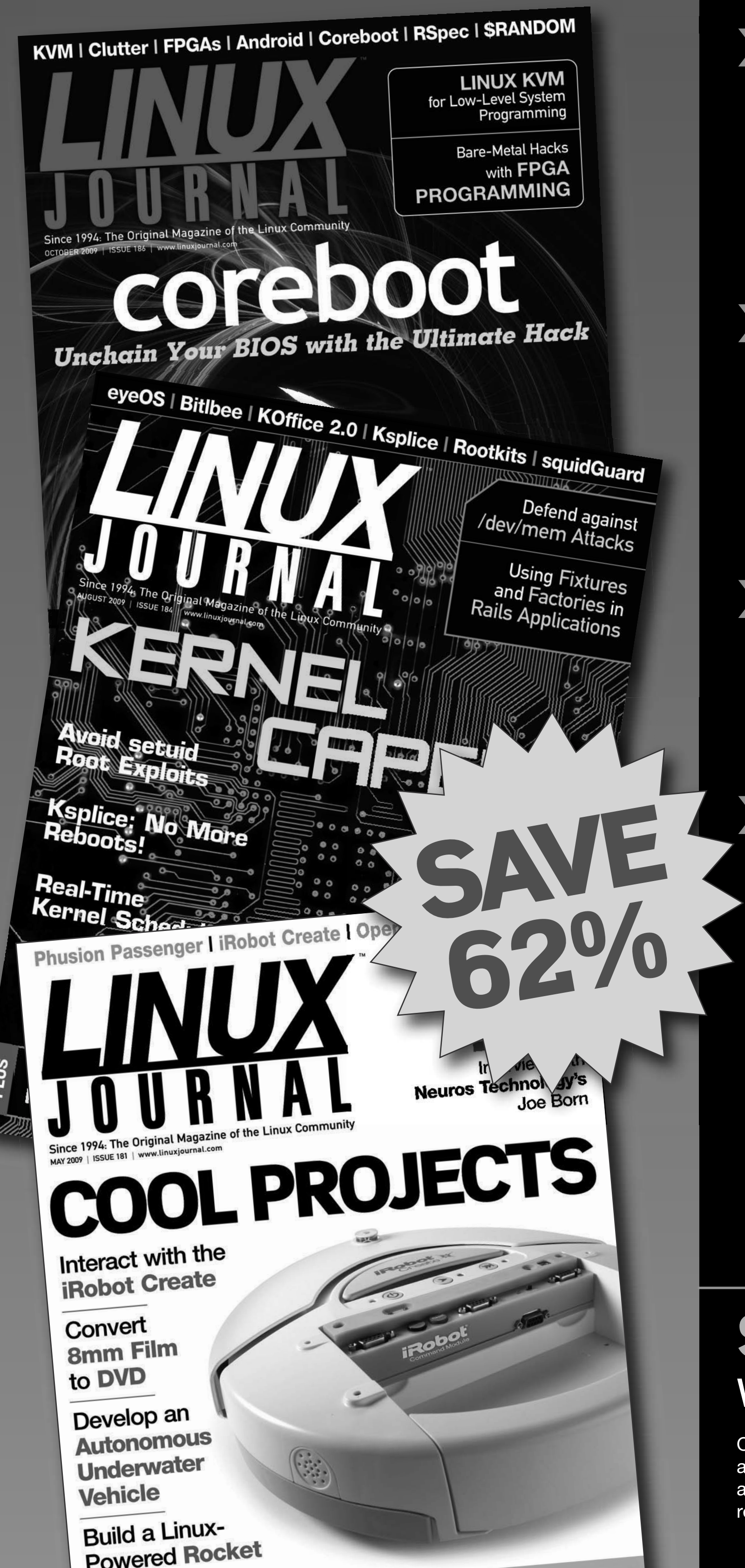
Find Out More and Stay Connected...

http://www.usenix.org/events/lisa10/facebook

http://twitter.com/LISAConference #lisa10

www.usenix.org/lisa10

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

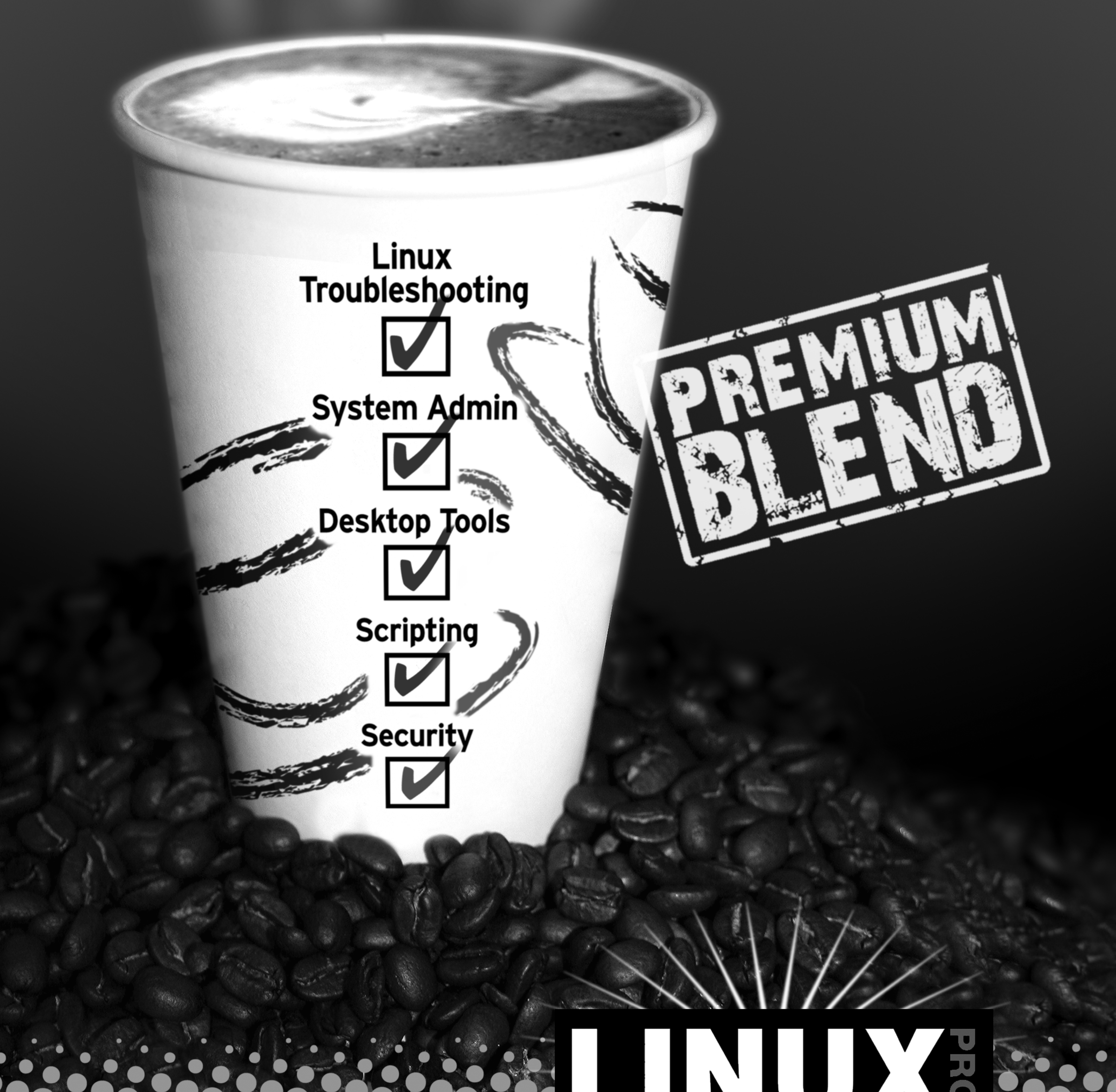


- In-depth information providing a full 360-degree look at featured topics relating to Linux
- Tools, tips and tricks you will use today as well as relevant information for the future
- Advice and inspiration for getting the most out of your Linux system
- Instructional how-tos will save you time and money

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

SUBSCRIBE NOW AT:
WWW.LINUXJOURNAL.COM/SUBSCRIBE

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



LINUX PRO MAGAZINE

If you like the taste of Linux, why not treat yourself to the best?

Linux Pro Magazine delivers real-world solutions for the technical reader. In every issue, you'll find advanced techniques for configuring and securing Linux systems. Learn about the latest tools and discover the secrets of the experts in Linux Pro. Each issue includes a full Linux distribution on DVD.

www.linuxpromagazine.com

USENIX

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

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

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

USENIX Federated Conferences Week

June 22–25, 2010 • Boston, MA

http://www.usenix.org/events/#june10

USENIX Federated Conferences Week will feature:

- USENIX ATC '10: 2010 USENIX Annual Technical Conference
- WebApps '10: USENIX Conference on Web Application Development
- WOSN 2010: 3rd Workshop on Online Social Networks
- HotCloud '10: 2nd USENIX Workshop on Hot Topics in Cloud Computing
- HotStorage '10: 2nd Workshop on Hot Topics in Storage and File Systems
- And more!

USENIX

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