conference reports

THANKS TO OUR SUMMARIZERS

The LSF '08 summaries were substantially abbreviated for publication. For the complete summaries, see http://www.usenix.org/events/lsf08/lsf08sums.pdf.

FAST '08: 6th USENIX Conference on File and Storage Technologies

San Jose, CA
February 26–29, 2008

KEYNOTE ADDRESS: "IT'S LIKE A FIRE. YOU JUST HAVE TO MOVE ON": RETHINKING PERSONAL DIGITAL ARCHIVING

Cathy Marshall, Senior Researcher, Microsoft

Summarized by Swapnil Bhatia (sbhatia@cs.unh.edu)

To a storage systems researcher, all user bytes are created opaque and equal. Whether they encode a timeless wedding photograph, a recording of a song downloaded from the WWW, or tax returns of yesteryear is not germane to the already complex problem of their storage. But according to Cathy Marshall, this is only one stripe of the storage beast. There is a bigger problem that we will have to confront eventually: The exponentially growing size of our personal digital estate will soon—if it has not done so already—surpass our management abilities.

In a visionary keynote address delivered in her own unique style, Marshall successfully argued the increasing importance, complexity, and enormity of the problem of personal digital archiving: the need for tools and methods for the collection, preservation, and long-term care of digital artifacts valuable to us.

In the first part of her talk, using data gathered from extensive surveys, interviews, and case studies, Marshall characterized the prevailing attitudes of users toward the fate of their digital data. She discovered a strange mix of reckless resignation, paranoia, and complacency based on flawed assumptions about the role of backups, the perceived permanence of the content on the WWW, and a misperception of one's own ability to manage one's data over a variety of time scales.

Marshall outlined four fundamental challenges in personal digital archiving: value, variation, scale, and context. According to Marshall, personal digital archiving is not just about backing up data; it is the selective preservation and care of digital artifacts that are of value to us. Quantifying the value of a digital artifact appears to be a difficult task, even for its owner. Furthermore, the variability in the value of a digital artifact over time to its user only adds complexity to the problem. Current methods of backup are value oblivious and therefore promote blind duplication rather than selective archival.

The second challenge arises from the distributed storage of our digital assets. Personal digital artifacts are almost never stored in a single central repository. Typically, they end up copied from their source to a number of personal computers and servers, some of which may not be controlled by the owner(s) of the digital artifacts. Moreover, all the many copies created in the process of the distribution may not be identical; for example, differences in resolution of images or associated metadata may result in many differing versions of essentially the same digital artifact. Without any supervening method of preserving provenance, the problem of archiving a family of related but different digital artifacts dispersed across many locations—building a digital Noah's ark of sorts—quickly becomes intractable.

Archiving would still be a manageable problem, were it not for its sheer enormity. According to Marshall, there are about seven billion pictures on the Yahoo! and Facebook sites. Most users are simply incapable of dealing with large numbers of digital artifacts because of a lack of either technological savvy or the time and effort needed. Archiving personal data requires stewardship, but no tools currently exist to facilitate it at this scale.

Finally, even the perfect archiving tool augmented with the best search interface would be of no help if, years later, one has forgotten the content and the context of one's archive. Such forgetfulness is—as Marshall found through her user interviews—an often underestimated problem. Marshall suggested that re-encountering archived data is a promising solution to this problem. The idea behind re-encountering is to enable archived data to remind the user of their own provenance by facilitating periodic review or revisitation.

Marshall concluded by saying that solving these challenges would require a method for assessing the value of digital artifacts and better curatorial tools and services with built-in facilities for re-encountering. Marshall also pointed out that this problem will require cooperation and partnership among social Web sites, software companies, data repositories, ISPs, and content publishers.

In response to a question from an audience member, Marshall mentioned that, as yet, users were not willing to pay for an archiving service. Another questioner asked Marshall to explain why it was not enough to back up all user data. The same misequation of backup with archiving also arose in audience discussions after the talk. Marshall explained that users were not looking to save everything: This would only make the problem of context and re-encounter harder. Rather, what users need is a selective way of preserving personally valuable data, along with the context that makes the data valuable, and doing so over a time scale that is significantly longer than that of a backup.

DISTRIBUTED STORAGE

Summarized by Chris Frost (frost@cs.ucla.edu)

- ***Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage***
  Mark W. Storer, Kevin M. Greenan, and Ethan L. Miller, University of California, Santa Cruz; Kaladhar Voruganti, Network Appliance

Mark Storer spoke on the Pergamum system, which uses disks, instead of tape, for archival storage, and their work toward reducing power costs. With Pergamum, Storer et al. wanted to achieve power cost and data space scalability similar to tape systems but achieve random access performance similar to disk array and Massive Array of Idle Disk systems. Their approach uses an evolvable distributed network of disk-based devices, called tomes. Each tome can function independently and is low-power enough to run on Power over Ethernet. Pergamum uses intradisk and interdisk reliability to protect against corruption and tome failure, including trees of data algebraic signatures to efficiently detect and locate corruption. A tome spins down its disk when inactive and stores metadata in nonvolatile RAM (NVRAM) to help keep its disk spun down even longer.

Storer et al.'s experiments show that adding one to three backup tomes per tome increases the mean time to failure by orders of magnitude. A tome's low-power CPU and SATA disk sustain a 5 MB/s write speed; they anticipate raising this with further CPU optimizations. One thousand tomes with a spin rate of 5% could ingest 175 MB/s. They cite costs as being 10%–50% of today's systems.

David Rosenthal of Stanford asked how well they understood drive failures, especially given a tome's difference from the expected environment. Storer noted that accelerated drive testing is a good idea and also that this is one reason they use interdisk redundancy and data scrubbing. Geof Kuenning of Harvey Mudd asked what would happen if a tome's NVRAM were to fail. Storer replied that they would replace the device and use interdisk reliability to rebuild. Another person asked about how much energy routers use, since disk arrays do not have this component. Storer answered that router and disk array backplane energy costs are similar.

- ***Scalable Performance of the Panasas Parallel File System***
  Brent Welch, Marc Unangst, and Zainul Abbasi, Panasas, Inc.; Garth Gibson, Panasas, Inc., and Carnegie Mellon University; Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou, Panasas, Inc.

Brent Welch presented Panasas's parallel file system, which has installations as large as 412 TB with sustained throughput of 24 GB/s using thousands of Panasas servers. The Panasas system is composed of storage and manager nodes; a storage node implements an object store and a manager node runs a metadata service that layers a distributed file system (PanFS, NFS, or CIFS) over the object store. Scalability is the goal of Panasas and is primarily achieved by distributing metadata and supporting scalable rebuilds.

Each file and its metadata are stored together, in an Object Storage Device File System. The system automatically selects redundancy (RAID) levels on a per-file basis based on file size, optimizing space usage for small files and run-time performance for large files. By randomly placing data among storage nodes, failure recovery is made scalable, an essential feature for such a large, distributed system. Panasas servers also include integrated batteries to permit them to treat

RAM as nonvolatile RAM (NVRAM), significantly increasing metadata manipulation performance. Welch et al. find that system write performance scales linearly with the number of servers; read performance scales fairly well, although effective readahead becomes difficult at large scales.

One audience member asked how many objects backed a typical file. Welch said, "Several; a new 512-kB file would consist of about ten objects." Another person asked how much of Panasas's scalability is due to using an object store. Welch replied that the object store primarily simplified the software. Another person, worried about the lack of true NVRAM, asked what would happen if a hardware fault lost the contents of RAM. Welch said an exploded storage server would not harm the file system, because data and logs are replicated synchronously.

- ***TierStore: A Distributed Filesystem for Challenged Networks in Developing Regions***
  Michael Demmer, Bowei Du, and Eric Brewer, University of California, Berkeley

Michael Demmer spoke on sharing data among computers in developing regions, where network connectivity is intermittent and of low bandwidth. Many approaches exist for working in such environments, but all start from scratch and use ad hoc solutions specific to their environment–application pair. Demmer et al. aim to extend the benefits of Delay Tolerant Networking to provide replicated storage for applications to easily use for storage and communication.

TierStore's design has three goals. (1) Software should be able to easily use TierStore. They achieve this goal by exposing TierStore as a file system. This permits many existing programs to use TierStore and takes advantage of the well-known, simple, programming language-agnostic, and relatively weakly consistent filesystem interface. (2) TierStore should provide offline availability. Therefore TierStore provides locally consistent views with a single-file coherence model and uses application-specific conflict resolvers to merge split views after a write–write conflict. (3) TierStore should distribute data efficiently. Delay-tolerant publish–subscribe is used to provide transport portability. A simple publish–subscribe protocol is used to manage interest on fine-grained publications (e.g., a user's mailbox or an RSS feed).

One audience member asked how TierStore deals with churn. Demmer replied that their current prototype is manually configured and has a fairly stable overall topology. Another person asked why Demmer et al. propose a fancy solution when they also say one cannot use fancy technology in such environments. Demmer answered that economics is often the barrier. When asked about the usability of conflict resolution in their system, Demmer said that, thus far, applications have been designed to be conflict-free (e.g., Maildirs). They hope applications will continue to be able to store data so that conflicts are not a problem.

YOU CACHE, I CACHE . . .

Summarized by Swapnil Bhatia (sbhatia@cs.unh.edu)

- ***On Multi-level Exclusive Caching: Offline Optimality and Why Promotions Are Better Than Demotions***
  Binny S. Gill, IBM Almaden Research Center

Binny S. Gill presented the two primary contributions of his work on a multi-level cache hierarchy: a new PROMOTE operation for achieving exclusivity efficiently, and policies that bound the optimal offline performance of the cache hierarchy.

Gill started his talk with a discussion of the DEMOTE technique used for achieving exclusivity. When a higher-level cache evicts a page, it DEMOTEs it to the lower cache, which in turn makes room for the new page, possibly demoting a page in the process itself. Gill argued that DEMOTE is an expensive operation and performs poorly when bandwidth is limited. Furthermore, for workloads without temporal locality, DEMOTEs are never useful.

Gill proposed a new technique for achieving exclusivity based on a new operation called PROMOTE. PROMOTE achieves exclusivity by including an ownership bit with each page traversing the hierarchy, indicating whether ownership of the page has been claimed by some lower-level cache. Each cache on a path PROMOTEs a page with a certain probability, which is adapted so as to equalize the cache life along a path. (A higher-level cache periodically sends its cache life to the next-lower-level cache.) Experimental results show that PROMOTE is better than DEMOTE when comparing average response time, and it cuts the response time roughly in half when intercache bandwidth is limited.

Gill also presented two policies, OPT-UB and OPT-LB, which bound the performance of an optimal offline policy for a multi-level cache hierarchy. Essentially, both policies use Belady's optimal offline policy for a single cache and apply it incrementally to a path. In his paper, Gill proves that no other policy can have a better hit rate, intercache traffic, and average response time than OPT-UB. Gill presented experimental results that showed that the two bounds were close to each other.

One audience member pointed out that using PROMOTE would require a change in the command set of the protocol used for intercache communication. Gill argued that the performance gained by using PROMOTE would hopefully incentivize such a change. In response to another question, Gill clarified that the response times of the caches need only be monotonically increasing, with no constraint on the magnitude of the differences. Another questioner provided a counter-example scenario in which the working set of the workload is highly dynamic and asked what impact this would have on the adaptive PROMOTE probabilities. Gill pointed out that when cache lives are equalized, the behavior of the PROMOTE scheme would be no different—in terms of hits—than that with DEMOTE.

■ ***AWOL: An Adaptive Write Optimizations Layer***

Alexandros Batsakis and Randal Burns, Johns Hopkins University; Arkady Kanevsky, James Lentini, and Thomas Talpey, Network Appliance, Inc.

Although an application writes data to the disk, in reality the file system caches writes in memory for destaging later. Alexandros Batsakis's talk tried to answer the following questions about such write-behind policies: How much and which dirty data should be written back? And when should this be done? The answers require careful consideration of the tradeoffs between writing too quickly or waiting too long, and using available memory for caching writes versus reads. To this end, Batsakis proposed a three-part solution: adaptive write-back, ghost-caching, and opportunistic queuing.

Batsakis explained an adaptive high–low watermark algorithm in which write-back commences when the "dirtying" rate crosses the high mark and stops when it falls below the low mark. The two watermarks are dynamically adapted in harmony with the dirtying and flushing rates.

Batsakis proposed ghost-caching to balance memory usage across reads and writes. The scheme involves the use of two ghost caches. The Ghost Miss Cache (GMC) records metadata of evictions resulting from write buffering. A cache miss with a GMC hit is used to deduce that write buffering is reducing the cache hit ratio. The Ghost Hit Cache (GHC) records a subset of the cached pages. A read hit that falls outside the GHC is used to deduce that additional write buffering will lower the read hit rate. Thus, the GHC is used to prevent interference from writes early on, rather than recover from it using the GMC later.

Batsakis proposed the use of opportunistic queuing to decide which data to write back. An I/O scheduler maintains separate queues for blocking (read) and nonblocking (writes) requests with requests sorted by block number to minimize seek time. In Batsakis's scheme, dirty blocks are added to a third (nonblocking) opportunistic queue. When a page is flushed from any of the other queues, the scheduler is free to service a "nearby" page from the opportunistic queue. Overall, experiments show that the three optimizations can improve performance by 30% for mixed workloads.

■ ***TaP: Table-based Prefetching for Storage Caches***

Mingju Li, Elizabeth Varki, and Swapnil Bhatia, University of New Hampshire; Arif Merchant, Hewlett-Packard Labs

Mingju Li presented the two primary contributions of her work: a method for detecting sequential access patterns in storage-level workloads and a method for resizing the storage-level prefetch cache optimally.

Table-based prefetching (TaP) is a sequential detection and cache resizing scheme that uses a separate table for recording workload history and resizing the prefetch cache optimally. TaP records the address of a cache miss in a table. If a contiguous request arrives later, then TaP concludes that a sequential access pattern exists in the workload, and it begins prefetching blocks on every subsequent cache hit from that stream. Separating the workload history needed for detection from other prefetched data prevents cache pollution and allows TaP to remember a longer history. As a result, TaP can detect sequential patterns that would have otherwise been lost by interleaving.

When the prefetch cache is full, TaP evicts a cache entry, but it records its address in the table. If a request for this recorded address arrives later, then TaP concludes this to be a symptom of cache shortage and expands the cache. Thus, TaP uses the table to resize the cache to a value that is both necessary and sufficient and hence optimal. As a result, TaP exhibits a higher useful prefetch ratio, i.e., the fraction of prefetches resulting in a hit. In many cases, TaP can achieve a given hit ratio using a cache that is an order of magnitude smaller than competing schemes.

In response to audience questions, Li mentioned that she planned to address the design of a prefetching module, which is responsible for deciding the size and time of prefetching, in her future work. Another question called attention to the cost of prefetch cache resizing: What is the impact of the frequent change in the size of the cache on performance? Li responded by saying that the rate at which the cache size is decreased can be controlled and set to a reasonable value. The final questioner asked Li how TaP would compare in performance to AMP (Gill and Bathen, FAST '07). Li pointed out that AMP could in fact be incorporated into TaP as one possible prefetching module and that this would be addressed in her future work.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Shafeeq Sinnamohideen (shafeeq@cs.cmu.edu)

■ ***Byzantine Fault-Tolerant Erasure-Coded Storage***

James Hendricks and Gregory R. Ganger, Carnegie Mellon University; Michael K. Reiter, University of North Carolina at Chapel Hill

Hendricks presented a scheme that provides Byzantine fault tolerance for a slight overhead over non-Byzantine-fault-tolerant erasure-coded storage. Traditionally, storage systems have used ad hoc approaches to deciding which faults to tolerate and how to tolerate them. As storage systems get more complex, the kinds of faults that can occur get harder to predict; thus, tolerating arbitrary faults will be useful. Providing Byzantine fault tolerance in an erasure-coded system requires each storage server to be able to validate that the data fragment stored on that server is consistent with the data stored on the other servers. This is difficult because no server has a complete copy of the data block. Using the recent technique of homomorphic fingerprinting, however, each server can validate its fragment against a fingerprint of the complete data block, and a client can validate that all the fragments it received from the servers are consistent with a single fingerprint. In a prototype system, Hendricks's scheme provides write throughput almost as good as a crash-only tolerant system, and with far better performance than existing Byzantine fault-tolerant schemes.

- ***Mirror File System***
  John Wong, Twin Peaks Software

Wong presented an overview of Twin Peaks's filesystem replication product. Since files stored on a local file system are vulnerable to failures of the local machine, and files stored on remote servers are vulnerable to failures of that server or the network, the Mirror File System (MFS) attempts to get the best of both worlds by mirroring the state of a local EXT3 file system onto a remote NFS server. It does so by transparently intercepting file system operations at the VFS layer and applying them to both the local and remote file systems, while requiring no modifications to applications, EXT3, or NFS.

- ***Quantifying Temporal and Spatial Localities***
  Cory Fox, Florida State University

Fox states that accurately describing workloads is critical to comparing different real workloads and creating accurate synthetic workloads. Because locality is an important property of a workload, understanding how the locality of a workload is transformed as it passes through system components, such as caches, is crucial. Fox suggests that current metrics such as cache hit ratios, reference distance, and block distance do not adequately describe locality. Instead he proposes a metric called "affinity," which builds on block and reference distances, while being less sensitive to hardware details. Much future work is anticipated in showing how well it captures locality and in studying how the workloads seen by individual components relate to the overall workload.

- ***Filesystems Should Be Like Burger Meals: Supersize Your Allocation Units!***
  Konstantin Koll, University of Dortmund, Germany

Koll discussed the tradeoff between small and large filesystem allocation units, through a humorous analogy to fast food meal sizes. In both file systems and restaurants, excessively small allocation units reduce waste, but at the same time they reduce performance and add administrative overhead. Koll stated that fast food vendors have realized that since food is cheap, avoiding waste is less important than reducing overheads, and since a study of filesystem workloads reveals that the amount of wasted space grows less than linearly with allocation unit size, filesystem designers should use larger allocation units. An unnamed questioner stated that XFS and other extent-based filesystems do use large allocation units. Koll responded: Then I hope you found this talk amusing.

- ***Zumastor: Enterprise NAS for Linux***
  Daniel Phillips

Zumastor is a NAS product, built on Linux, that provides live volume snapshots, remote volume replication, online volume backup, NFS, Samba, and CIFS interfaces, and easy administration. It is based on the ddsnap engine, which is a mostly userspace driver presenting a block device interface with copy-before-write snapshots. The snapshots can be replicated to other Zumastor servers using techniques such as compression or binary differencing to reduce the amount of data to be transmitted. Future work includes adding a graphical administrative console, better volume management, online resizing, and incremental backup.

- ***View-based Collective I/O for MPI-IO***
  Javier Garcia Blas, Florin Isaila, and Jesus Carratero, University Carlos III of Madrid

Blas proposed an alternative to two-phase collective I/O with the goal of increasing performance by reducing the cost of data scatter-gather operations, minimizing the overhead of metadata transfer, and reducing the amount of synchronous communication. The alternative, called view-based collective I/O, relies on clients once providing the aggregators with additional information about the type of the data the client will be accessing. This reduces the amount of information the client must send with every access, as well as permitting the aggregators to cache file data across operations, allowing one operation to benefit from a previous operation's cache miss. In the MPI-IO benchmark performed, view-based I/O reduced write times by more than a factor of 2 and read times by at least 10%.

- ***Towards a Performance Model for Virtualised Multi-Tier Storage Systems***
  Nicholas Dingle, Peter Harrison, William Knottenbelt, Abagail Lebrecht, and Soraya Zertal, Imperial College London, UK

Lebrecht's goal is to model the performance of a complex system using nested queuing network models of its basic components. Starting with models of simple disk and RAID system, the models she has developed closely match the performance of the real devices for random workloads. Future work includes extending these results to more complex and heterogeneous workloads.

- ***Adapting RAID Methods for Use in Object Storage Systems***
  David Bigelow, Scott A. Brandt, Carlos Maltzahn, and Sage Weil, University of California, Santa Cruz

Bigelow proposes characterizing the tradeoffs among various methods of implementing RAID in object-based storage systems. These include "client-based RAID," in which the client performs parity calculations, "RAID across objects," in which the storage system stores an object on one node but includes a different object on each node in the parity calculation, and "RAID within objects," in which the storage system stores a portion of each object on each node and computes parity across all of them. Bigelow is currently working on implementing these schemes in the Ceph Object

Storage System and evaluating their relative performance, as well as developing more complex and hierarchical schemes.

■ ***How Shareable Are Home Directories?***
Carlos Maltzahn, University of California, Santa Cruz

Maltzahn hypothesizes that most users manage a subset of their files in a way identical to other users. Thus, it should be possible to share the work of managing files across users. He proposes to quantify this shareability by categorizing files through a user survey. His survey, in which users categorized files as "unshareable," "shareable within one user," "shareable within one group of users," and "publicly shareable," revealed that 75% of users have at least half their files shareable in some way, and 50% of users have at least half their files in common with a different user.

■ ***Load Balancing in Ceph: Load Balancing with Pseudorandom Placement***
Esteban Molina-Estolano, Carlos Maltzahn, and Scott Brandt, University of California, Santa Cruz

Molina described several issues that could be encountered in Ceph owing to its use of pseudo-random placement of objects, along with potential solutions to these issues. In the case of one node that happens to hold the primary replica of many popular objects, that node can be switched to be a secondary replica of some of them, moving the load to the new primary replica. In the case of a read flash crowd, some of the readers can be directed to other nodes that hold a secondary replica of the object, or even to other clients that have a recently cached copy of the object. In the case of a write flash crowd, some clients can be directed to write to the secondary replicas, but this relies on HPC I/O extensions that allow the application to describe ordering and dependencies. Preliminary results show that these techniques allow load to be shifted away from an overloaded storage node.

■ ***Ringer: A Global-Scale Lightweight P2P File Service***
Ian Pye, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz

Pye presented a global file service that provides filesystem semantics as well as indexing based on document contents. Filesystem semantics are necessary for application compatibility, and indexing is necessary to help users find the files they are interested in. The architecture Pye proposes is a Hybrid P2P approach in which metadata servers maintain the filesystem indices and perform searches, but file data is transferred directly from the peer that has it. Future work includes implementing, testing, and evaluating the system.

■ ***The New and Improved FileBench***
Eric Kustarz, Spencer Shepler, and Andrew Wilson, Sun Microsystems

Spencer described the current state of the FileBench framework. It provides a collection of configurable workloads and can apply them to a number of storage server types. It recently underwent a large code cleanup and is distributed in OpenSolaris and through SourceForge.net. Features in development include support for random workloads, NFS, CIFS, and multiple clients.

■ ***HyFS: A Highly Available Distributed File System***
Jianqiang Luo, Mochan Shrestha, and Lihao Xu, Wayne State University

Luo proposes a Linux filesystem that uses erasure coding to provide redundancy against hardware failure. HyFS is implemented at the user level by using FUSE and erasure codes file data across a user-configured number of NFS servers. Performance and scalability evaluations are ongoing.

■ ***Virtualizing Disk Performance with Fahrrad***
Anna Povzner, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz; Richard Golding and Theodore M. Wong, IBM Almaden Research Center

Povzner extends existing work in providing soft performance isolation to provide hard isolation guarantees. The Fahrrad disk scheduler allows clients to reserve disk time and attempts to minimize the seeks required to serve the client workloads. If seeks between streams are necessary, they are accounted to the streams that required them and thus the necessary overhead time can be reserved, allowing for hard isolation. Experiments show that this can provide complete isolation of competing workloads, with only a 2% overhead.

■ ***RADoN: QoS in Storage Networks***
Tim Kaldeway and Andrew Shewmaker, University of California, Santa Cruz; Richard Golding and Theodore M. Wong, IBM Almaden Research Center

Kaldeway presented RADoN, which aims to coordinate the individual network, cache, and storage QoS parameters in order to provide end-to-end QoS guarantees for a given application. Doing so requires discovering how the parameters of individual system components affect the overall QoS. This project seeks to discover the important parameters through modeling and simulating the system and coordination strategies and will build a framework for applications to specify their QoS requirements.

■ ***Improving Efficiency and Enhancing Concurrency of Untrusted Storage***
Christian Cachin, IBM Zürich; Idit Keidar and Alexander Shraer, Technion: Israel Institute of Technology

Cachin summarized recent improvements in protecting against storage servers that present different views of history to different clients. Fork linearizability is a useful building block because it ensures that once a server has forked a view, it must remain forked. The authors reduce the communication cost of a fork-linearizable protocol to $nO(n)$ messages instead of $O(n2)$ and show that such a protocol can never be wait-free. Instead they introduce a weak fork-linearizable protocol that is wait-free and has the same communication cost.

- ***Reliability Markov Models Are Becoming Unreliable***
  Kevin M. Greenan and Jay J. Wylie, Hewlett-Packard

Greenan described the Markov models traditionally used for reliability analysis. Whereas the single disk and RAID-5 models accurately model the reliability of such systems, a naive RAID-6 model underestimates the MTTDL of a RAID-6 system by a factor of 2. This is because the memorylessness of the model ignores data that may have been rebuilt onto other disks after a disk failure. Although not accurate, Markov models may provide the correct intuition in reasoning about failures, and future work is necessary to develop them further or devise new models.

KEYNOTE ADDRESS: SUSTAINABLE INFORMATION TECHNOLOGY ECOSYSTEM

Chandrakant D. Patel, HP Fellow, Hewlett-Packard Labs

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

In a far-looking presentation, Chandrakant Patel explored the hidden costs of disks and the data center, along with the potential to deliver more efficient and reliable services. Patel suggested that in the future, bringing IT to all levels of developing economies will require a sustainable ecosystem. This is based on the conviction that the technologies with the least footprint, lowest power consumption, and least materials will eventually have the lowest total cost of ownership. To realize this transformative change, we must recharacterize the costs associated with IT, drawing on deep technical knowledge of the physical processes involved.

The costs of IT can be measured at several levels. Initially, materials are extracted and formed into a usable product; next, the product is used in operation for some time; finally, the product is recycled, allowing some resources to be reclaimed while others are irreversibly lost. Other considerations such as transportation and human effort can also be incorporated. A quantifiable metric for these costs is "exergy," which is drawn from thermodynamics as the measure of the available and usable energy in a system. In the illustrative case of a cellular phone, the CPU draws power from the battery in order to perform its function. In the process, exergy consumed by the CPU is converted to heat, which is dispelled passively in the body of the phone. In the future, owing to high power density and the difficulty of removing heat passively from stacked chip packages, even heat removal will require powered solutions. To mitigate the added power requirements, Patel suggested an active-passive solution—a phone filled with a phase-change material such as wax, which absorbs heat for short conversations and switches to active solid state heat removal. Such active-passive cooling solutions will be necessary to provision power based on need.

Using some of these principles, Patel and his associates are developing data centers that are significantly more energy-efficient. The key observation is that the common practice of over-provisioning results in unnecessary redundancy and cooling. By emphasizing the pervasive use of sensors, one can develop a flexible and configurable approach driven by policies. Applying this technique to the cooling system of a data center has resulted in a 35% energy savings. Patel argued that the system also results in improved reliability, because it can quickly adapt to problems as they occur. The cooling is provided on-demand and over-provisioning, although it remains important, can be limited to a cost-efficient and pragmatic level.

In closing, Patel elaborated on some future directions for the work. He stressed the need to analyze the costs associated with the lifetime use of IT and showed how software tools can help with this analysis. He suggested combining the data from sensors with thermo-mechanical attributes of compute and storage components, to detect anomalies and predict failure. Stressing that the currency of the flat world will be joules of exergy consumed, Patel emphasized the importance of collaboration among computer scientists, mechanical engineers, and electrical engineers.

Eric Brewer, of Intel Research and U.C. Berkeley, asked if the joule is an accurate measure of total cost of ownership. He suggested that market inefficiencies are prevalent and seem to be growing. Patel acknowledged this discrepancy but postulated that in the long run sustainability concerns will come to dominate the costs. He also pointed to recent successes in the data center, where a sustainability-oriented approach has allowed him to quickly reduce costs. Mochan Shrestha noted that limiting provisioning could result in an increased error rate and wondered whether it was necessary to incorporate the value of the data into the model. Patel said that in practice this valuation is problematic but that it can be approximated with service-level agreements.

FAILURES AND LOSS

Summarized by Medha Bhadkamkar (medha@cs.fiu.edu)

- ***The RAID-6 Liberation Codes***
  James S. Plank, University of Tennessee

Plank offered an alternate RAID-6 encoding scheme that has near-optimal performance for encoding, decoding, and modifications. Plank first described the motivation, which is based on the drawbacks of the current implementation of RAID-6 systems: They are typically slow and modifications are suboptimal and inflexible. The proposed code, termed Liberation Codes, uses parity arrays, which are $w \times w$ bit matrices, where w is a prime number equal to or greater than the number of devices. The performance is compared with the Reed-Solomon coding. The evaluations show that the encoding is primarily focused on parity-based protection and single errors in RAID systems. Modification performance is overoptimal, but decoding performance is 15% of optimal. To further optimize decoding operations, a Bit Matrix scheduler for the XOR operations is proposed to reduce the Liberation decoding overhead by a factor of between 6 and 11, depending on the values of *w*. Optimal values have also been achieved for the nonprime values of *w* = {2,4}. The paper also provides a URL to the freely available source of the Liberation Coding Library.

Nitin Garg of Data Domain posited that other matrices can have bad cache performance and wondered whether Plank had compared his method with any other methods, such as the Reed-Solomon error correcting code. Is it true that evaluating cache performance is important? Plank replied that they haven't explored caching with Reed-Solomon codes. To a question about the optimal value of *k* for an 8- to 10-GB disk, Plank responded that roughly a factor of 2 for encoding, but up to 4 for decoding, was optimal.

- ***Are Disks the Dominant Contributor for Storage Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics***
  Weihang Jiang, Chongfeng Hu, and Yuanyuan Zhou, University of Illinois at Urbana-Champaign; Arkady Kanevsky, Network Appliance, Inc.

Jiang began by stating the importance of reliability and availability in storage systems. As storage systems have evolved from single hard disks to network storage systems, it is necessary to have a good understanding of the failure characteristics of disk drives. The data used in this study was obtained by analyzing failure logs for about 44 months from 39,000 commercial storage systems and about 1.8 million disks. The data was analyzed in three dimensions, with four failure types being classified based on their root cause and symptoms, the effect of design factors such as disk models and enclosures, and statistical properties. The results show that, first, whereas disk failures (29%) form a substantial part of storage system failures, failures of other components also make a substantial contribution (7% protocol failures and 60% interconnect failures). Second, after a failure, the probability of another failure of the same type is higher. Third, interconnect redundancy is an important factor. Finally, shelf enclosures play an important role in failure patterns. This study does not take into account the impact of workloads, the reason behind failures, or the consequences of different failure types.

Someone from Wayne State University asked how disk failures are defined: by data loss or by service loss? Jiang answered that problems such as scratches, vibrations, or malfunctioning of internal components are responsible for disk failures. Data loss is a consequence of disk failures.

- ***Parity Lost and Parity Regained***
  Andrew Krioukov and Lakshmi N. Bairavasundaram, University of Wisconsin, Madison; Garth R. Goodson, Kiran Srinivasan, and Randy Thelen, Network Appliance, Inc.; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

Andrew Krioukov explained that RAID systems offer numerous data protection techniques and it is unclear which technique or combination of techniques can protect against which kind of errors. The focus is primarily on parity-based protection and single errors in RAID systems. To solve this problem, a formal method based on model checking is used to analyze the design of the protection techniques. The model checker outputs a state machine that shows the state transitions that are obtained and searches the space for all possible states. It provides primitives for disk operations, such as atomic reads and writes, and for data protection, such as checksums and parity. For every analysis, exactly one error is injected. The model checker is also used to generate data loss or corruption probabilities. The results show that, for all designs, single errors can cause data loss. In addition, data scrubbing, which is used to reduce double disk failure, actually spreads corrupt data in one block to other blocks because of parity calculations. Also, data loss has a higher probability than data corruption. To address the issues uncovered, the authors also propose a protection mechanism, which uses version mirroring and combines block checksums, identity information, parity, and scrubbing.

Someone asked how one would handle a case where a misdirected write overwrites a parity block. Krioukov said that since parity blocks are protected by checksums, by a comparison of blocks on the data disk and the parity disk we know whether data has been lost, and it can be restored by reconstruction. Erik Reidel of Seagate asked about the complexity involved in representing the operations in a model. Krioukov said that building the model checker was simple, but building the framework with primitives was difficult. John Carrier of Cray, Inc., asked whether the model checker can be extended to RAID 6, especially since RAID 5 will be phased out soon. Krioukov answered that it can be used with double parity and can definitely be extended.

CPUS, COMPILERS, AND PACKETS, OH MY!

Summarized by Elie Krevat (ekrevat@cs.cmu.edu)

- ***Enhancing Storage System Availability on Multi-Core Architectures with Recovery-Conscious Scheduling***
  Sangeetha Seshadri, Georgia Institute of Technology; Lawrence Chiu, Cornel Constantinescu, Subashini Balachandran, and Clem Dickey, IBM Almaden Research Center; Ling Liu, Georgia Institute of Technology; Paul Muench, IBM Almaden Research Center

As legacy storage systems transition toward multi-core architectures and embedded storage software systems (controllers) are becoming more complex and difficult to test, Sangeetha Seshadri and her co-authors argue that it is important not to focus just on performance but also on system availability. Storage controllers have many interacting components and exhibit many different types of transient failures (these types are classified in the paper). A transient failure that occurs in one thread is typically handled by restarting and reinitializing the entire system, which also requires consistency checks. As systems grow to larger numbers of cores, systemwide recovery may not scale. However, task-level recovery has the potential for recovering a smaller subset of components in the system, with the additional challenges of determining the correct recovery semantics (dynamic and stateful), identifying recovery dependencies, and bounding the recovery process in time and resource consumption.

To improve recovery time and better scale the recovery process with system growth, the authors propose a framework for more fine-grained task-level recovery. The design goals of these recovery-conscious scheduling algorithms include creating a nonintrusive recovery framework, dynamically determining recovery actions, tracking recovery dependencies, and generally improving system availability by reducing the ripple effect of failures. Developers specify clean-up blocks (task-specific recovery procedures) and explicit dependencies between tasks, which are then refined by the system into recovery groups that include implicit dependencies. To limit the number of dependent tasks dispatched concurrently, resource pools partition the processors into smaller independent units. A recovery-conscious scheduler maps recovery groups to resource pools in a static or dynamic fashion while adhering to recoverability constraints. The scheduler bounds the time of recovery and reduces the impact of a failure in a proactive manner by limiting the number of outstanding tasks per recovery group, and in a reactive manner after a failure occurs by waiting to dispatch new tasks for a group currently undergoing recovery until after the recovery completes. The authors implement their recovery-conscious scheduler (RCS) on real industry-strength storage controllers and compare it with a standard performance-oriented scheduler (POS) that does not consider recovery dependencies. They measure the good-path performance during normal operation, and the bad-path performance under localized failure and recovery, using the z/OS Cache-Standard workload for which they identify 16 recovery groups. Experiments show that Dynamic RCS closely matches the good-path throughput of POS while improving bad-path throughput by 16.3%.

One of the participants asked how easy it is to define recovery groups, and if the developer gets it wrong, how that would affect performance. Sangeetha answered that a developer defines recovery groups explicitly based on whether a task accesses the same resource, which depends on the system, complexity of code, and other interactions. She noted that task-level recovery is an option, but system-level recovery is still needed as a safety net. Another participant asked what properties of tasks make them easier to identify than components as a reasonable boundary for recovery. The response was that techniques such as micro-reboots will reset the system, but tasks handle resources across different components. In some situations the controller can retry the operation at the task level and succeed, and based on task functionality these situations can be identified. The last question addressed how the system would scale for pool sizes greater than a single processor. The response was that the experimental setup had eight processors, a larger pool size is possible, and it's just an issue of defining the right granularity. For coarser constraints between groups, it would make sense to have a larger pool size.

- ***Improving I/O Performance of Applications through Compiler-Directed Code Restructuring***
  Mahmut Kandemir and Seung Woo Son, Pennsylvania State University; Mustafa Karakoy, Imperial College

Large-scale applications in science and engineering have grown dramatically in complexity, requiring huge computational needs and generating large amounts of data. According to Seung Woo Son and his co-authors, I/O is always a pressing problem for these data-intensive applications, but disk performance has not kept pace with the large annual growth in storage capacity density and processor speeds. One promising way to handle this problem and improve I/O performance is to reduce the number of disk accesses, achieved at different layers of the I/O subsystem by caching or restructuring the application code to maximize data reuse. Since the compiler has better knowledge of the entire application code, including data access patterns, the authors address the growing I/O problem through compiler-directed code restructuring, which can be used along with other OS- and hardware-based schemes.

The authors propose an approach to increase disk reuse in the compiler, which will hopefully also reduce the number of disk accesses by improving the chances of finding data in the cache. Their approach optimizes the entire program code rather than individual loop-nests, and they discussed file layout optimizations for adapting to parallel execution. The targeted disk system architecture is one in which file striping occurs over parallel disks, where a compiler should know which disks are accessed by certain portions of an array, either because this information is already supplied to the compiler or because it is available from an API. Inter-iteration data dependencies may not allow for code restructuring for better disk reuse, but if an ordering is legal for the particular data dependencies, then the authors make use of polyhedral algebra based on Presburger arithmetic to capture and enumerate those legal loop iterations that exhibit disk access locality. A disk map is defined to capture a particular set of disks, and a disk locality set is a set of loop iterations that access the same set of disks. Then the authors use two procedures to maximize disk reuse. First, for a given disk array, iterations in the disk locality set are executed consecutively. Second, when moving from one disk locality set to another, a disk locality set that accesses a new disk map is selected to have minimum Hamming distance from the current disk map. This second condition minimizes the number of disks whose status is changed when executing the iterations in a new disk locality set. Since real applications have other data dependencies, the authors also demonstrate how to use existing heuristics to merge or split nodes in a locality set graph (LSG) that captures these dependencies, thereby converting a nonschedulable LSG to a schedulable one. The authors also show how file layout modifications (striping info) can be changed, using profiling to detect the most suitable file layout for each file and then transforming the layout during optimization. Because good disk reuse for a single CPU does not imply that good disk reuse happens overall, the scheduling algorithm determines the global (inter-thread) usage of disks and selects disk locality sets based on a global estimate.

To evaluate these scheduling algorithms, a compiler was implemented using SUIF, and different algorithms were tested on a number of applications. The whole program-based disk reuse optimization (DRO-WP) algorithm achieved on average 23.9% better performance in I/O time than the base algorithm and 15% better performance than using conventional data locality optimization (CLO) techniques. This performance improvement occurs because the average number of times a given data block is visited is much lower with DRO-WP than with CLO (2.1 compared to 3.9). Other results show that performance gains are sensitive to the size of the cache but still substantial with higher cache sizes, and parallel thread optimization is important in maximizing overall disk reuse, especially with a large number of CPUs.

One of the participants tried to understand the limitations of this approach by asking whether there have been other optimizations besides using the polyhedral approach that were considered but did not map well. Seung answered that they use conventional solutions from other domains, which looks reasonable for now, but there is still room for more optimizations. Another participant asked whether the approach to file layout optimization, when looking over all the access patterns to find a better layout, used estimated weights for different bits of code accessing the same block or assumed the same rate of access. The response was that a ranking system is used to determine the best candidate, rating each optimization and selecting the highest value, but weights for the rate of access in different code regions were not used. Another participant, noting that in large-scale systems things change constantly, wanted to know if he would need to recompile for a changing environment to better optimize I/O. The answer to this question was that reoptimization is suggested in changing environments. To the question of how this work of limiting data reads performs in areas of HPC that don't do many data reads (e.g., internal file systems may not even cache data), the answer was that, when possible, the compiler can reduce the set of disks that are accessed at any time. The last question, whether the compiler can identify data reads and reuse explicitly or can only monitor disk reuse, elicited the response that no information on data reuse is given to the compiler.

■ ***Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems***
Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan, Carnegie Mellon University

Building cluster-based storage systems using commodity TCP/IP and Ethernet networks is attractive because of their low cost and ease of use, along with the desire to use the same routing infrastructure for LAN, SAN, and high-performance computing traffic. However, Amar Phanishayee and his co-authors argue that an important barrier to high-performance storage using these commoditized networks is the problem of TCP throughput collapse: the incast problem. Incast occurs during synchronized reads of data striped over multiple storage servers, where the system is limited by the completion time of the slowest storage node, and the concurrent flood of traffic from many servers increases past the ability of an Ethernet switch to buffer packets. Dropped packets at the switch can cause one or more TCP timeouts which impose a relatively large delay until TCP recovers, resulting in a significant degradation of throughput. The authors recreate the incast problem while performing synchronized reads on an Ethernet-based storage cluster. They fix the amount of data read from each storage server, defined as a Server Request Unit (SRU), and increase the number of servers involved in a data transfer. This experiment shows an initial improvement of throughput up to around 900 Mbps with three servers, and then a sharp order-of-magnitude collapse. Other experiments fixing the block size while scaling the number of servers produce the same collapse. Although one might expect TCP to completely utilize the bottleneck link, and a tool that measures throughput using long-lived TCP streams does not experience incast, it is perplexing that a particular setup with typical communication patterns in storage systems can cause such a significant performance loss.

The authors study the network conditions that cause TCP throughput collapse, characterize its behavior under a variety of conditions, and examine the effectiveness of TCP- and Ethernet-level solutions. To understand the reasons for throughput collapse, a distinction is made between TCP's mechanism for data-driven loss recovery, which occurs when a sender receives three duplicate acknowledgments and is relatively fast, and timeout-driven loss recovery, when no feedback is available from receiver to sender and the sender must wait until the Retransmission TimeOut (RTO) time has passed before continuing the flow, a relatively slow process. Timeouts cause throughput collapse because a server loses its packets at the switch, and without any feedback it must fall back to timeout-driven recovery. Simulations of the incast problem show that doubling the switch buffer size from 32 to 64KB also doubles the number of servers supported before a collapse. However, switches that support fast buffers are expensive. Increasing the SRU size means that servers have more data to send per data block and produce less idle time on the link; however, a larger SRU size also requires each server to do more data prefetching while the client has to allocate more memory for the complete block. The TCP variants of NewReno and SACK avoid certain timeout situations when compared to Reno, with NewReno performing best, but all variants eventually experience throughput collapse as the number of participating servers is increased. Reducing the penalty of a timeout by lowering TCP's minimum retransmission timeout period from 200 milliseconds to 200 microseconds helps significantly but is impractical and unsafe, because it requires timer support in microseconds from the operating system and may create unnecessary timeouts and retransmissions when talking to clients in the wide area network. Ethernet flow control helps only for the simplest network settings, but in more common multi-switched systems it has adverse effects on other flows, produces head-of-line blocking, and is inconsistently implemented across different switches. New standards for Data Center Ethernet are being developed to create a lossless version of Ethernet, but it is unclear when these new standards will be implemented in switches, and there are no guarantees that implementation of these standards will be uniform or that new switches will be as inexpensive as they are currently. Without any single convincing network-level solution, application-level solutions by the storage system may be more appropriate, since the storage system has the knowledge of all data flows and the control to limit situations that may cause throughput collapse.

One of the participants asked whether this problem is related to TCP using a single stream, and if a solution such as SCTP, which transports multiple message streams, would be better. Amar answered that only TCP was considered because it is used by most developers and is very simple and workable, since most machines have TCP implementations. In response to the question of whether work on an adaptive RTO would apply, Amar said that RTO is already adaptive and is based on the round-trip time estimation. Another participant asked about the queuing discipline implemented at the switch. Amar said that drop tail and random drops were used, but these didn't provide a solution. Another participant, remarking that the problem with TCP flows is that they are bursty and stay open for a long time, asked whether explicitly causing TCP to close its congestion window was used. The answer was that disabling TCP slow start in experiments did not help. To the suggestion that other TCP variants that avoid loss altogether be used, Amar responded that with the TCP variants that were tried, RED (which drops packets early) and ECN (which notifies the server to back off) were not successful in preventing incast. Two participants asked about application-level solutions, such as introducing random delay at the servers, and the response was that indeed this was one of the solutions that might help, that staggering should help to limit overflow at the switch buffer, but that very early experiments did not demonstrate much improvement.

WHERE DID WE GO WRONG?

Summarized by James Hendricks (James.Hendricks@cs.cmu.edu)

■ ***Portably Solving File TOCTTOU Races with Hardness Amplification***

Dan Tsafrir, IBM T.J. Watson Research Center; Tomer Hertz, Microsoft Research; David Wagner, University of California, Berkeley; Dilma Da Silva, IBM T.J. Watson Research Center

Awarded Best Paper!

Photo by Ethan Miller

Dan Tsafrir started by explaining the time-of-check-to-time-of-use (TOCTTOU) race problem. Suppose some root-privileged script deletes files in /tmp that are not accessed for a while. The script would (1) check the access time of each file F and (2) delete F if it had not been accessed recently. This approach, however, may allow an attacker to trick the script into deleting any file because there is a window of vulnerability between the check operation (examining F's access time) and the use operation (deleting F). For example, an attacker may make a directory /tmp/etc and file /tmp/etc/passwd, then symlink /etc to /tmp/etc at the right moment. Check will decide to delete /tmp/etc/passwd, but use will delete /etc/passwd because /tmp/etc will be pointed to /etc during the window of vulnerability. (Search the Web for "TOCTTOU symlink" for real-world vulnerabilities.)

TOCTTOU vulnerabilities occur between any check-use pair of system calls that involve a name of a file; in the example here it's lstat(F) and unlink(F). Thus, there are many variants of the problem, often providing attackers with the ability to obtain permanent root access. One proposed solution is *hardness amplification* (Dean and Hu). The idea is to check the condition multiple times, reducing the probability of a TOCTTOU race. Unfortunately, a maze of symbolic links makes TOCTTOU attacks much more feasible (Borisov et al.) because traversing symbolic links is slow, easily defeating such defenses and many similar proposals. The authors propose a generic check-use mechanism that emulates the kernel's file-path resolution procedure in user mode. This approach allows programmers to safely execute most check-use operations without suffering from the associated TOCTTOU problems, effectively binding the check-use pair into an atomic transaction. The fine-grained control over the path resolution process allows programmers to express policies such as forbidding symbolic links along the path or verifying that a given user is allowed to operate on a given file. The solution is portable and works for existing systems, in contrast to prior proposals which changed the kernel or the API.

Bill Bolosky from Microsoft Research noted that the race exists because the check-use mechanism is not atomic, that transactions remove this race, and that Windows Vista has transactions. The speaker responded that the solution in the paper is portable. Another questioner asked about the cost of doing the check in userspace. The speaker responded that some performance is lost. There is a tradeoff between efficiency and safety, and the proposed mechanism takes 3–6 times longer to complete than the naive insecure alternative.

- ***EIO: Error Handling Is Occasionally Correct***
  Haryadi S. Gunawi, Cindy Rubio-González, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Ben Liblit, University of Wisconsin, Madison

Haryadi Gunawi started by stating that errors are often improperly propagated. The authors considered 34 error codes (e.g., EIO, ENOMEM) on 51 file systems (all in linux/fs/*) and three storage drivers (SCSI, IDE, software RAID) and found that 1153 of 9022 function calls do not save the propagated error codes. More complex file systems have more propagation violations, and writes are worse than reads. Propagation errors are common, not just corner cases. Common problems occur when the error code goes unchecked (e.g., `err = func()` is called, but `err` is not propagated), unsaved (e.g., `func()` is called, ignoring any returned errors), or overwritten (e.g., `err = func1(); err = func2()` is called, discarding the error code from `func1`).

The authors built a tool to map the call graph, demonstrating which error calls are incorrectly propagated. The call graphs are impressive and the reader is encouraged to peruse them in the online proceedings. Coda correctly propagates all errors (the audience applauded efforts by the Coda team); several systems incorrectly propagate more than a quarter of all errors. The authors concluded the talk and the paper with entertaining responses from developers (e.g., "Should we pass any errors back?" and "Just ignore errors at this point. There is nothing we can do except try to keep going.").

Dave Chinner of SGI noted that some of the XFS faults have been fixed. SGI has recently implemented a tool to ensure that errors that should be checked are checked. He would be interested in a more recent run. Another questioner asked whether the tool would be released; developers often hear anecdotes of problems but are given no way to correct them. The speaker responded that the tool will be released (it will be located at http://www.cs.wisc.edu/adsl/Publications/eio-fast08/readme.html). Keith Smith of NetApp asked whether they had any experience trying this technique in other parts of the kernel; the speaker responded that he is beginning to look into other parts of the kernel.

- ***An Analysis of Data Corruption in the Storage Stack***
  Lakshmi N. Bairavasundaram, University of Wisconsin, Madison; Garth Goodson, Network Appliance, Inc.; Bianca Schroeder, University of Toronto; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

  Awarded Best Student Paper!

Photo by Ethan Miller

Lakshmi Bairavasundaram started by saying that files get corrupted and that corruptions are often correlated. Unfortunately, our understanding of how data is corrupted is mostly anecdotal. The authors analyzed over 1.5 million disks in thousands of NetApp systems over 41 months. There are many types of corruption, such as basic bit corruption, lost writes (writes ignored by disk), misdirected writes (blocks sent to the wrong physical location), and torn writes (in which only part of the block is written). NetApp applies various techniques, including checksums and disk scrubbing, to detect the effects of these faults, such as checksum mismatches, parity inconsistencies, and identity discrepancies. Enterprise disks have 10% as many faults as nearline disks, and bit corruption and torn writes are more common than lost writes or misdirected writes. Different models age differently (some fail early, some fail late, and some are less affected).

Nearline drives have checksum mismatches more often. But when enterprise drives have any checksum mismatches, they tend to get several mismatches. Checksum mismatches are often for consecutive or nearby blocks, with high temporal locality. Furthermore, they are not independent in a disk array. In one drive model, a particular logical block number was much more likely to exhibit faults, and certain ranges of blocks were somewhat more likely to develop faults than others. Thus, staggering RAID stripes for each disk may be a good idea. Given the variety of faults, it is wise to make efforts to detect, prevent, and recover from faults. Preventing data loss may require double parity and vigilant scrubbing.

Mary Baker from HP Labs asked whether the numbers were broken down by firmware revision in addition to model; the speaker responded that this is a good point but that they were not. Rik Farrow asked whether, given that hard drives reorganize data on the disk, the speaker could say anything about the fact that he found correlations with block numbers. The speaker replied that the correlations were found using logical block numbers. Another questioner suggested that error correlations could be due to the NetApp boxes, given that all tests were run on NetApp systems. The speaker said that was not likely, since he found very different types of errors for different disk models. Another questioner asked whether the type of errors correlates to the type of end user, but the speaker said this was not studied. Bruce Worthington of Microsoft asked whether part of the correlation could be due to overwriting critical blocks repeatedly and whether it would make sense to move master blocks around. The speaker responded that the errors seem to be more due to firmware and software, so overwrite frequency was not a likely culprit.

BUFFERS, POWER, AND BOTTLENECKS

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

- ***BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage***
  Hyojun Kim and Seongjun Ahn, Software Laboratory of Samsung Electronics, Korea

Hyojun Kim proposed adding a RAM-based cache and management system to flash devices in order to improve random write performance. NAND-based flash memory has grown to become the primary storage solution for mobile devices owing to its good overall performance and low power consumption. However, the medium suffers from low random write performance, which may hamper its growth in the future.

Existing filesystems perform write operations that are ill-suited to flash memory because of hardware limitations that force I/O operations to be aligned at a coarse granularity. In addition, overwrite operations require first erasing the target area, which incurs additional cost. By adding a RAM buffer, similar to those that exist in conventional disk drives, write operations can be coalesced and performed more efficiently. The Block Padding Least Recently Used (BPLRU) algorithm was introduced to manage this buffer. It operates by merging adjacent requests such that they can be written at the block size of the flash unit and by prioritizing these lower-cost write operations. The system was evaluated with trace-based simulations, a subset of which were then verified on a prototype directly.

Brent Welch of Panasas wondered about the buffer's operation under power failure. The current system does not tolerate this scenario, but Kim identified the problem as one that may be solved in the future. The potential for increased latency was also a concern, as it was not formally evaluated.

- ***Write Off-Loading: Practical Power Management for Enterprise Storage***
  Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron, Microsoft Research Ltd.

Dushyanth Narayanan made the case that significant power savings can be achieved in data centers by spinning down idle disks. He also demonstrated that further savings are possible when writes to idle disks are redirected to already active disks. The results, which were based on a trace of measurements of real workloads, stand in contrast to previous benchmark-based findings that suggested such an approach to be impractical.

Narayanan described an approach that allows incremental deployment by working with existing storage and file systems. By adding a management module to the storage stack of each volume, drives that idle for some period of time can be spun down. In addition, future writes to such a disk can be redirected to another disk that is already active, thus avoiding the overheads associated with spinning up idle disks, and allowing fewer disks to remain in operation at any given time. This latter technique was referred to as write off-loading.

In servicing these off-loaded write requests, a log structure associated with the remapping operation is stored on the active disk at a known location. In addition, a record of the remapped block is stored in the memory of the original host. Future reads may then be redirected to the appropriate disk, which will be activated if needed. These remapping operations are temporary, which ensures that the underlying storage structure is not degraded. The system tolerates failure by reestablishing the block locations after an unclean shutdown. Narayanan also showed that the increased latency associated with an operation that activates an idle disk was significant but that this occurrence was rare in practice.

The audience responded actively, allowing Narayanan to elaborate on several topics. He described the load-based heuristic for selecting an off-loading target and the transparency benefits to ensuring that remapped blocks are short-lived. He also asserted that the benefits of write off-loading would be present even on a system that had consolidated its data onto fewer disks.

- ***Avoiding the Disk Bottleneck in the Data Domain Deduplication File System***
  Benjamin Zhu, Data Domain, Inc.; Kai Li, Data Domain, Inc., and Princeton University; Hugo Patterson, Data Domain, Inc.

Data Domain's system for enterprise data backup and network-based disaster recovery was described by Benjamin

Zhu. He showed how the company's file system provides a high degree of data deduplication and off-site backup, with good performance. The system was evaluated with data gathered from real customers, which showed a data compression rate up to 30 times, with throughput meeting or exceeding the system's 100 MB/s target.

After providing some background, Zhu detailed the differences between primary storage and backup. Whereas the former focuses on latency, seek times, and throughput, the latter is concerned with large batched writes and space efficiency. In the case of a remote backup, bandwidth over the WAN link is also a concern. The Data Domain File System targets these problems with a combination of compression and deduplication based on content hash.

In a large data set, detecting duplicates can be costly. The size of the hash table can quickly exceed the available RAM and, when flushed to disk, the table shows no temporal or spatial locality. To address this challenge, a Bloom filter is used to determine whether a particular segment is a duplicate. Since a negative result from the Bloom filter is only probabilistically correct, such a hypothesis must be confirmed by consulting an on-disk structure. To help regain locality, segments written by the same stream are placed together on-disk in containers. Once duplicate data is merged in storage, it is compressed to further save space.

After the presentation, a lengthy exchange ensued regarding the performance of the system at some boundary conditions. At least one attendee was concerned that a long series of nonduplicate data could overwhelm the system's ability to efficiently flush metadata. The debate was deferred until later before any consensus was reached. Others wondered about the potential for hash collision. Zhu initially said that such a collision was rare enough to be of no concern; he later clarified, explaining that a mechanism to test for collisions existed but was disabled by default.

COMPLIANCE AND PROVISIONING

Summarized by Chris Frost (frost@cs.ucla.edu)

- ***Towards Tamper-evident Storage on Patterned Media***
  Pieter H. Hartel, Leon Abelmann, and Mohammed G. Khatib, University of Twente, The Netherlands

Everyone wants to prevent data tampering, and Write-Once Read-Many (WORM) devices may be of help. Compared to disks and flash, they have high access times and low throughput. Hartel suggested that an alternative approach to tamper-resistant storage is tamper-evident storage, where data may be modified, but any modification will be detected. Common techniques include calculating the hash of data and writing this to a notebook and giving it to a notary public. The difficulty with these approaches lies in managing these hashes and keeping them consistent with their tracking of the data-hash pairs. This talk proposed a hardware device based on patterned media that can store both Write-Many Read-Many and WORM data, effectively providing Selectively Eventually Read-Only (SERO) storage.

Hartel et al. show how, in theory, a device could support magnetic read and write operations as well as electrical read and write operations. An electrical write would permanently change the location's magnetic properties, but electrical reads and writes are significantly slower than the magnetic equivalents, so one could make data tamper-evident by storing only a secure hash electrically. Specifically, they propose storing each hash bit in two bits with a parity of one (the Manchester encoding) of electrical storage. To modify data an attacker would need to either find data with an equal hash or modify the hash value. But an attacker could only set unset bits in the hash, and this would change the bit's parity. They also presented a filesystem design for SERO storage, in which fragmentation becomes a serious concern as more of the disk becomes read-only.

Peter Honeyman asked whether they have simulated operation timings. Hartel answered that they have not, but that they hope patterned media will support both archival and online storage. Bill Bolosky asked what would stop an attacker from blanking the entire SERO device and rewriting it with modifications, to work around the tamper-evidence guards. The authors replied that one could, but this would take some time, and perhaps the device could record the fact.

- ***SWEEPER: An Efficient Disaster Recovery Point Identification Mechanism***
  Akshat Verma, IBM India Research; Kaladhar Voruganti, Network Appliance; Ramani Routray, IBM Almaden Research; Rohit Jain, Yahoo! India

Akshat Verma presented their work on quickly finding a backup snapshot from before a latent error occurred. Current methods for quickly finding such a backup are ad hoc; their system systematizes the location process. SWEEPER logs system events that, with a goal recovery time and recovery point objective, help speed up good backup identification. SWEEPER's balanced search strategy calculates probabilities that certain events are correlated with the specified error and uses its event log, along with a binary search, to locate good backups. Example events include misconfiguration, virus activity, hardware warnings and errors, and applications logs.

An audience member asked about the sensitivity of their benchmark to event weights. Verma replied that SWEEPER's informed search can be way off but that this is acceptable because the binary search will still find a good backup.

- ***Using Utility to Provision Storage Systems***
  John D. Strunk, Carnegie Mellon University; Eno Thereska, Microsoft Research, Cambridge, UK; Christos Faloutsos and Gregory R. Ganger, Carnegie Mellon University

In provisioning a storage system (e.g., for OLTP or scientific or archival purposes), trade-offs are unavoidable. For example, increasing data protection can harm performance or increase purchase cost. Whereas the existing practice is to consult an area expert, John Strunk spoke on how utility functions can convey the cost-benefit structure to an automated provisioning tool. Users are then able to make appropriate trade-offs among various system metrics.

Strunk et al. use utility functions, functions from a set of metrics (e.g., revenue, availability, data value, power usage, or purchase cost) to a utility value (e.g., dollars), to characterize a particular point in the purchase space. To find a desirable point in this (large) space they use a genetic algorithm to refine a configuration population over many generations. Strunk then illustrated the value of this approach through three case studies, including scenarios with a limited budget and where system cost can affect the long-term solution.

Peter Honeyman asked why linear programming was not used instead of a genetic algorithm. Strunk answered that linear programming's constraints on objective function form rules out many real-world utility functions. Honeyman also asked whether one can maximize multiple objectives; Strunk replied that you would convert these to one utility. Another audience member asked whether they had looked at a method for generating good utility functions, noting that Strunk's seemed simplistic. Strunk said they have, that the paper has more examples, and that this is also an area where they are doing further work. One person asked whether this approach can determine whether it is better to upgrade an existing system or migrate to a new system. Strunk answered that they can do this, but that it is the second part of his thesis. Two audience members asked whether Strunk's approach supported varying input values as a function of time. Strunk answered that their system focuses only on static provisioning. The final questioner asked whether not finding the most optimal solution is a problem. Strunk replied that in the real world one often only gets in the ballpark, and that this approach already does at least as well as today's ad hoc approaches.