conference reports

THANKS TO OUR SUMMARIZERS

FAST '07 70
Andrew Leung
Michael Mesnier
Brandon Philips
Raja Sambasivan
Avishay Traeger
Charles Weddle

LSF '07..... 84
Brandon Philips

FAST '07: 5th USENIX Conference on File and Storage Technologies

San Jose, CA
February 13–16, 2007

INVITED TALK

A Crash Course on Some Recent Bug Finding Tricks

Dawson Engler, Stanford University
Summarized by Brandon Philips (brandon@ifup.org)

Storage systems have a simple and important contract to keep: Given user data, they must save that data to disk without loss or corruption even in the face of system crashes. Dawson Engler and his students at Stanford have created eXplode, a systematic approach to finding bugs in storage systems, to help root out the bugs that can break this contract.

eXplode systematically explores all the possible choices that can be made at each choice point in the code to make low-probability events, or corner cases, just as probable as the main running path. And it does this exploration on a real running system with minimal modifications.

This system has the advantage of being conceptually simple and very effective. Bugs were found in every major Linux file system, including an fsync bug that can cause data corruption on ext2. This bug can be produced by doing the following: Create a new file, B, which recycles an indirect block from a recently truncated file, A, then call fsync on file B and crash the system before file A's truncate gets to disk. There is now inconsistent data on disk, and when e2fsck tries to fix the inconsistency it corrupts file B's data. A discussion of the bug has been started on the linux-fsdevel mailing list.

EXE (EXecution generated Executions) is another useful testing tool and was also briefly discussed. A developer using the tool would annotate the application to mark unrestrained input data. In the case of file systems, the unrestrained input data is a disk to mount. The annotated code is then run through exe-cc, which instruments the code, and then is compiled using gcc.

The added instrumentation will track all constraints on the input data to discover inputs that can cause termination by a call to exit(), crash, assertion failure, or error. The inputs that can lead the code to terminate are recorded and used to create an "input of death" or, in the case of a file system, a "disk of death" that can crash the system and uncover exploitable bugs.

Input sanitation on mounts is becoming more important as USB flash drives become pervasive and nonroot users get the ability to mount drives.

MEASURE THRICE

Summarized by Avishay Traeger (atraeger@cs.sunysb.edu)

■ ***Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?***

Bianca Schroeder and Garth A. Gibson, Carnegie Mellon University

Awarded Best Paper!

Bianca Schroeder began by stating that understanding disk failure frequencies has become increasingly important as server clusters become larger. Disk replacement data shows that what system administrators see in the real world is far different from what we statistically predict based on manufacturer specifications. Better knowledge about the statistical properties of storage failure processes, such as the distribution of time between failures, can empower researchers and designers to develop new, more reliable and available storage systems.

The authors draw their conclusions from seven supercomputing and ISP data sets, which include more than 100,000 drives. Some factors for why disks are replaced more frequently than predicted may be that administrators and disk manufacturers may disagree on the definition of a disk failure and that real-world operating conditions do not match those used by the manufacturers in their accelerated stress tests.

Some of the interesting observations are that the MTTF (Mean Time to Failure) is always much lower than the observed time to disk replacement, that SATA is not necessarily less reliable than FC and SCSI disks, and that, contrary to popular belief, hard drive replacement rates do not enter steady state after the first year of operation, but in fact steadily increase over time. In addition, early onset of wear-out has a stronger impact on replacement than does infant mortality. The authors also show that the common assumptions that times between failures follow an exponential distribution and that failures are independent are not correct. In addition to analyzing the data, they are creating a public failure data repository and hope that their results will help build better systems—for example, by predicting the probability of a RAID failure.

With regard to RAID failures, where a second drive failed during reconstruction, one questioner asked whether the data includes information about failures in specific arrays. The reply was that it does not, but the data that was used should provide a conservative estimate. To the question of whether the authors had a hypothesis about the correlation of disk replacements, their reply was that the disks are in the same physical environment and are probably under similar loads. Another question involved the accuracy of the data, given that the authors used disk replacements in their metrics, rather than actual failures. The authors said that even if they assumed that half of the replacements were unnecessary, the failure rate is still twice as much as the MTTF indicates. Someone asked how disk batches affected the data, and if it could account for all of the results, which would indicate that the data should be discarded. Since customers do not have data regarding bad disk batches, the authors said that this is difficult to estimate, but it is unlikely that it had much effect. Another person noted that since bad batches exist in the real world, the data should not be discarded.

■ ***Failure Trends in a Large Disk Drive Population***

Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso, Google Inc.

Although magnetic media are used to store most of the world's information, there is little published work on the failure patterns of disk drives and the key factors that affect their lifetime. Eduardo Pinheiro and the authors presented failure statistics from Google and analyzed the correlation between failures and several parameters generally believed to impact longevity. This analysis is made possible by a new highly parallel health data collection and analysis infrastructure, along with the ample amount of data provided by the authors' computing deployment. The hope is that this analysis will help predict failures so that administrators can act preemptively, help diagnose problems, and improve fault-tolerant techniques.

The authors found that there was no consistent correlation between higher temperature drives (for the available data, between 20 and 50 degrees Celsius) or drives at higher utilization levels (normalized per drive model) with failure rates. This indicated that other effects may be more prominent in affecting disk drive reliability. They confirm that some of the SMART parameters, such as scan errors, are well-correlated with higher failure probabilities. However, failure prediction models based on SMART parameters alone are likely to be limited in their accuracy, since many drives failed with no SMART errors.

Randal Burns (Johns Hopkins University) asked how this research has affected disk replacement policies at Google. The reply was they have not achieved high enough accuracy yet to base purchasing decisions on the data. Someone asked about the kind of accuracy one could get by modeling the SMART data. It was explained that this would result in many false positives and not enough good predictions, but it depends on the modeling techniques. In response to whether it was worth replacing disks as soon as an error was seen, the reply was that it would not be cost-effective. Someone later noted that this is true for Google since they have very good replication, but it is not true for the common user. Rik Farrow (*;login:* editor) asked whether multiple SMART errors are a better indicator for disk

failure, and the presenters admitted that this is in fact the case. Someone inquired as to whether the data that was collected included disk access patterns. Although this was examined, no significant differences were found. Ying Wang asked whether the type of disk failure was taken into consideration (i.e., bad blocks, head crashes, etc.). Eduardo replied that they did not include such data in their analysis, and they used disk replacements to model failures. Noting the correlation between SMART errors and failures, another questioner asked why this can't be used for prediction, but it was explained that although the drives will be more likely to fail after error, the rates are not so much more likely that a prediction is possible. More SMART data may help form predictions. Finally, the question of whether the correlation between disk failures and variation in temperatures was investigated came up. Although it was, there was no strong correlation.

■ ***A Five-Year Study of File-System Metadata***

Nitin Agrawal, University of Wisconsin, Madison; William J. Bolosky, John R. Douceur, and Jacob R. Lorch, Microsoft Research

Nitin Agrawal explained that metadata from over 60,000 Windows PC file systems at Microsoft collected over five years was used to study how various file characteristics changed over time. This data can prove useful for designers of file systems and related software, testing hypotheses, driving simulations, and validating benchmarks. The authors also provided a generative, probabilistic model for how directory trees are created, in terms of the depth of the namespace tree and the distribution of subdirectories. The authors plan to make their data set publicly available.

Some of their observations: (1) Both the mean file size and the number of files have increased. (2) A few filename extensions account for a large percentage of files and storage and have not changed much with time. (3) Less filesystem content is created or modified locally over time. (4) Directory size distribution has not changed significantly, although directory size is steadily increasing. (5) The fraction of documents in the namespace subtree meant for user documents and settings has increased in every year of the study. (6) Although filesystem capacity has increased dramatically, filesystem fullness has only slightly decreased. (7) Large files (e.g., binaries, movies, database files) are contributing to an increasing fraction of filesystem usage, but most files are still 4KB or smaller. (8) The median file age is between 80 and 160 days, and this has not changed with time.

Erez Zadok (Stony Brook University) asked whether there were trends in how people were organizing their files. The reply was that most files were being stored in the Windows, Documents and Settings, and Program Files directories. Another question was about the types of systems and workloads that were analyzed. Agrawal said that almost all were Windows PCs and were mostly from developers. Geoff Kuenning (Harvey Mudd College) asked how they considered machines where capacity was added, or the situation when a file system was migrated to a new machine. The study considered these to be new file systems. There was an inquiry about the presenter mentioning that most files were less than 4KB but also saying that the median size was 4KB. Agrawal joked that, in this case, most means half, and the audience laughed. In response to whether differences between developer and nondeveloper distributions were considered, the reply was that they considered temporal trends and not workload types, but the data is available for future analysis. Eduardo Pinheiro (Google) inquired about the reason for 4KB being the median file size, wondering whether this had something to do with block sizes or application behavior. The presenter was unsure of the cause. Finally, John Wilkes (HP Labs) asked about the amount of unique data present in the file system, since all probably had similar installs. While admitting that this data may be useful to analyze, they said they had not done so.

WHO PUT THEIR NETWORK IN MY STORAGE?

Summarized by Michael Mesnier (michael.mesnier@intel.com)

■ ***Proportional-Share Scheduling for Distributed Storage Systems***

Yin Wang, University of Michigan; Arif Merchant, HP Laboratories

Yin Wang began by saying that the advantage of distributed storage is the ability to scale out using cheap commodity hardware (storage bricks). HP's Federated Array of Bricks (FAB) is one example of such a distributed architecture. However, the drawback is often unavoidable resource contention, which is complicated by the fact that data is typically distributed across multiple storage bricks and accessed by clients using different I/O patterns.

The goal of this work is to provide proportional-share scheduling of storage resources based on a client's "weight," which can be assigned by an administrator. No previous work in this area has addressed multiple schedulers with multiple resources (storage bricks). Of course, if a scheduler is simply placed at every storage coordinator or back-end storage device, no one scheduler will ever see all of the I/O requests. For example, a scheduler at a storage brick will not see requests to other storage bricks, and a scheduler at a coordinator will not see requests to other coordinators. Thus it would be difficult to determine a client's total service across all resources.

One naive solution is to broadcast all I/O requests to all bricks or coordinators, but this of course introduces significant network overhead. However, the authors show that only the service cost of each I/O needs to be broadcasted to each coordinator. Such a broadcast includes a "delay value" that allows a storage coordinator to delay I/O requests from a particular client in order to provide propor-

tional-share service across clients. In their architecture, coordinators maintain FCFS queues and can therefore easily incorporate the broadcasted delay values.

The authors present a new proportional-service scheduling framework suitable for use in a distributed storage system that uses such a delay-broadcast approach. Their approach is an extension of SFQ(D), called Distributed Start-time Fair Queuing (DSFQ). Two approaches are evaluated: Total-DSFQ and Hybrid-DSFQ. The primary difference is that Hybrid-DSFQ guarantees a minimum amount of service to each client. In their evaluation, the authors showed that Total-DSFQ works in the single-brick case, across multiple bricks, and across multiple bricks with dependencies among the I/O requests. They also showed that Hybrid-DSFQ, unlike Total-DSFQ, guarantees a minimum share for each client stream for fluctuating workloads.

David Black (EMC) asked whether there was any negative result with respect to disk striping; he believed that total service was at odds with striping. Arif Merchant said that the degree of proportional sharing can be adjusted to work with striping. Wenguang Wang (Apple) then asked what would happen if the client issued one stream with small random I/O and another with sequential. Yin said that that you can still maintain total proportional-service sharing.

■ ***Argon: Performance Insulation for Shared Storage Servers***

Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger, Carnegie Mellon University

Matthew Wachs stated that benefits of shared storage include the simplicity of a single infrastructure and the ability to better load-balance workloads across storage servers. However, these benefits come at the cost of interference among different workloads. When workloads share storage resources without any regard to efficient I/O scheduling, storage efficiency can plummet. Whereas processor and network sharing is well understood, it is unclear how to best share storage resources such as disk and cache. Moreover, it is unclear what a "time slice" even means for a resource such as a cache.

The goal of Argon is to provide better performance insulation. In the ideal case, sharing a storage server among *n* processes will result in each process receiving 1/*n* of its stand-alone performance. Because this ideal may not always be achievable, Argon uses an "R-value" to force a bound on lost efficiency. Once an R-value, such as 0.9, is set, each of the processes will then have performance within that fraction of the ideal. In general, typical disk scheduling policies do not grant sequential workloads long enough blocks of time, resulting in too much interference from seeks. Therefore, one goal of Argon is to amortize the cost of each seek by having larger request sizes for sequential workloads (using prefetching or write coalescing). For a given R-value, Argon automatically determines at system startup the optimal request size for a given storage server.

Seek-cost amortization solves only part of the performance insulation problem. The other part is cache pollution. To address this, Argon statically partitions cache space among *n* competing workloads, such that each workload is given an amount of reserved cache space. The amount each workload is given is determined through simulation: Block-level traces are replayed through a cache simulator in order to determine how a workload's performance improves with increased cache allocation. Again, the goal is to achieve within an R-value of the ideal performance by balancing the cache allocations for each workload.

Wenguang Wang (Apple) asked about the cache replacement policy, believing something smarter than LRU could be used. Matthew agreed, noting that although LRU was used for the experiments, one could use different policies for different workloads. Matthew further noted that for certain combinations of workloads, more intelligent policies (e.g., ARC) may obviate the need to statically partition the cache. Another attendee asked whether Argon yields the remainder of a time slice to scheduled workloads. Matthew replied by saying that Argon does not currently implement yielding. The challenge is determining the "right" time to yield, as application think time may make it appear that a request stream is idle, when in fact more I/O is soon to be issued. Yin Wang (University of Michigan) asked whether a trade-off exists between the length of the time slice (request size) and the fairness. Matthew said no, as fairness can also be achieved with larger time slices. More strictly, fairness can be achieved with any length time slice so long as the same length is given to each workload. The purpose of longer time slices is to increase efficiency, not change fairness. An attendee from IBM suggested that better write scheduling (e.g., request coalescing) could improve performance, and Matthew agreed. Another attendee asked what would happen if the cache were not large enough to store prefetched data for a large number of sequential workloads. Matthew said that the I/O accesses would degrade into random with a shared cache and that Argon does nothing to change this fact. Finally, an attendee from NetApp mentioned that a large variance in response time might be unacceptable for certain applications. Matthew agreed, but he also noted that a low mean response time is often more important than the variance for many applications. Matthew further noted that applications that need low variance in response times might need to have their own systems.

■ ***Strong Accountability for Network Storage***

Aydan R. Yumerefendi and Jeffrey S. Chase, Duke University

For network storage to be trustworthy, Adan Yumerefendi declaimed that strong accountability is required for both clients and servers. Servers must be able to track client accesses and guarantee nonrepudiation. Clients must be able to verify that their actions against a server have been committed and verify that a server is faithful.

The authors presented CATS, a rudimentary network storage service with strong accountability properties. CATS takes a "trust but verify" approach. The properties of their threat model include authenticity and undeniability, freshness and consistency, and completeness and inclusion. Confidentiality is also desired but is outside the scope of this work. The mechanisms used by CATS to ensure strong accountability include digital signatures, client action history, broadcasts of digital signatures of server memory state, challenges and proofs, and auditing.

CATS is an object storage service and a state storage toolkit. Each stored object is annotated with action records that reflect all state changes to the object. Versioning preserves multiple versions of each object, and version stamps allow clients and servers to agree on the ordering of updates. Such stamps also make request replay/reordering detectable. Servers must commit to their state and broadcast a digital signature of their internal memory state to the clients. Thus, a server can deny service but never subvert the system. Moreover, servers can make provable statements about their internal state. Freshness prevents a server from reverting writes, as each accepted write is represented in the digest of a server's internal memory state, so a committed request cannot be removed without detection by the clients.

The CATS storage service presented by the authors is just one example of a strongly accountable service that can be built by using the CATS state storage toolkit. The authors' evaluation shows that the cost of such a service can be high but that the approach is practical when strong accountability is required.

One attendee asked whether clients can exchange digests directly to perform verification. Aydan answered yes, noting that a common framework could be used to publish digests. An attendee from HP labs asked how the systems scales to large numbers of objects, specifically, whether one can keep files from disappearing. Aydan said that a bound on the probability of the file/object's existence can be given, but it depends on the age of the object.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Brandon Philips (brandon@ifup.org)

■ *Secure, Archival Storage with POTSHARDS*

Mark Storer introduced POTSHARDS, a system to securely store archival data. The system attempts to find a better solution to long-term encrypted storage. It uses secret sharing across a number of RAID archives and ensures that no one archive gets enough shards to reconstruct the secret. It also uses approximate pointers to ensure that an attacker has to have access to the data on all archives in order to reconstruct the data set.

■ *SeFS: Unleashing the Power of Full-Text Search on File Systems*

Stergios Anastasiadis proposed SeFS, a file system that is designed to facilitate full-text search. The need for this new file system was defended by the weaknesses in existing search technologies such as electronic journal databases and Web searches. Journal database entries are immutable once they are added to the database and Web search engines index data infrequently, but on most file systems files are modified frequently, making both approaches unacceptable for filesystem search.

■ *On the Scalability of Storage Sub-System Back-end Network*

Yan Li presented progress on research concerning how many disks a Fibre Channel Switched Bunch of Disks (FC SBOD) can efficiently support. The workload will be simulated using the Storage Performance Council SPC-1 benchmark. The initial findings show that a 2Gbps FC SBOD is saturated by 48 disks under RAID5 and 53 disks under RAID6 with a stripe size of 16KB. Future work includes testing 4Gbps versus dual 2Gbps FC and the scalability of FC SBOD with a cache system.

■ *Layout-aware Exhaustive Search*

Aravindan Raghuveer presented the motivation and plan for a layout-aware exhaustive search algorithm. The work is motivated by Jim Gray's keynote at FAST '05, where he projected that future hard disks will take a day to read 10TB of data sequentially and 5 months with random access and 8KB sectors. To get the fastest search possible an exhaustive search algorithm will need two properties: It must use physical layout information to compute a optimal sequential search, and it must have the ability to suspend and resume a search to provide service to real-time requests. Two pieces of metadata will need to be maintained to achieve these goals: location metadata on data object placement and state metadata on a suspended search.

■ *Scaling Security for Big, Parallel File Systems*

Andrew Leung proposed a new protocol, Maat, to scale I/O security to petabyte-scale parallel file systems used for high-performance computing applications. Current systems rely on shared-key cryptography, which can be a weakness in a large system. Maat uses extended capabilities, automatic revocation, and secure delegations. Extended capabilities can authorize I/O for a number of clients and files, which reduces the number of capabilities in the system. The small number of capabilities allows asymmetric cryptography to be used efficiently. Each capability has an automatic revocation timeout and can be extended by using a small, efficient extension token. An implementation is underway on top of the Ceph parallel file system.

■ *CompulsiveFS: Making NVRAM Suitable for Extremely Reliable Storage*

Kevin Green presented CompulsiveFS, a file system that stores persistent metadata directly to an erasure coded log in NVRAM instead of a RAM cache. The file system performs incremental erasure-encoding and signature computations over the contents of the log to create fault tolerance. The current prototype log is an order of magnitude faster than page-protected caches for small writes of 10–50 bytes. CompulsiveFS is in the early stages of research, design, and implementation.

■ *Performance Evaluation of RAID6 Systems*

Yan Li presented a plan for a three-piece study on the performance characteristics of RAID6 under the Storage Performance Council-1 benchmark. The study will simulate the storage using SimRAID, which models and simulates the RAID controller, cache, fibre-channel bus, and disks and has been shown to have a maximum inaccuracy of 5%. The study looks at RAID6 performance under fault-free mode, degraded mode, and recovery mode. Of particular interest is finding the ideal mix of handling requests versus rebuilding a disk.

■ *GANESHA, a Multi-Usage with Large Cache NFSv4 Server*

Philippe Deniel presented work on GANESHA, a user-space NFSv4 server with a large cache and support for a number of backend file systems. GANESHA has a filesystem abstraction layer (FSAL) that makes writing plug-ins for different backing file systems easy. Currently, there is support for HPSS and POSIX backends. A number of interesting backends are under development; these include an NFSv4 client that will allow GANESHA to be a proxy, an SNMP backend that will allow MIBS to be browsed, and the LDAP backend that will allow browsing of trees. Announcements to SourceForge and freshmeat are forthcoming.

■ *FlexiCache: A Flexible Interface for Customizing Linux File System Buffer Cache Replacement Policies*

Pavan Konanki presented the initial work for FlexiCache, an interface in the Linux kernel to accommodate different buffer cache replacement algorithms. The motivation for this work is to test new replacement algorithms, such as ARC, PCC and LIRS, that have been shown to perform better under certain access patterns. Also, this API would accelerate the testing and development of new buffer cache replacement algorithms. The key issue is trying to design the system to support many replacement policies while keeping the cache mechanics hidden. The performance implications of this added generality are also unknown.

■ *Diamonds Are Forever, Files Are Not*

Surendar Chandra talked about storage systems that use an "importance number" to decide how to manage a data store automatically. The experiments were motivated by a storage server for video-recorded lectures where some of the data may be less important than new data coming in and can be removed. To make the system successful, administrators must be able to specify accurate object lifetimes; therefore providing usage feedback is important. Currently a system is being built to prototype this idea.

■ *RBF: A New Storage Structure for Space-Efficient Queries for Multidimensional Metadata in OSS*

Yu Hua presented RBF, an r-tree with a bloom filter at each node, a structure that allows for efficient point and range queries. Point queries ask whether an object is in a data set and a range query grabs the set of objects that match a query. The application of this is to make efficient object-based storage devices that can do point/range operations on object metadata. Currently, a real 10TB storage system has been implemented using a partial implementation of the RBF structure.

■ *Storage Performance Isolation: An Investigation of Contemporary I/O Schedulers*

Sarala Arunagiri presented research on the performance isolation characteristics of a number of modern I/O schedulers. This research is motivated by the quality of service guarantees that large consolidated shared storage must make. The findings suggest that many I/O schedulers do not provide performance isolation in all circumstances.

INVITED TALK

■ *Trends in Managing Data at the Petabyte Scale*

Steve Kleiman, Senior VP and CTO, Network Appliance
Summarized by Michael Mesnier (michael.mesnier@intel.com)

In the first three-quarters of 2006, approximately 900 PB of storage was shipped worldwide by storage systems vendors, including Dell, EMC, Hitachi, HP, IBM, and Network Appliance. Of this total, Network Appliance shipped approximately 200 PB and is currently at a 100 PB/quarter run rate.

The big challenge today is keeping pace with such growth. A 50–100% yearly increase in storage capacity is not uncommon. Most of this growth is from unstructured data (e.g., contracts, letters, memos) and semi-structured data (e.g., email), but structured data (db) is also growing. Managing this growth introduces hidden burdens (costs).

A common strategy is to overprovision, resulting in low disk utilizations (with 25% or less being typical). Of course, CIOs do not want to take chances when it comes to safely storing company and/or customer data. Today's legal burdens (e.g., regulatory compliance) and social burdens (e.g., losing data is bad press) underscore this point. In general, overprovisioning stems from the "build-out" manner in which most storage infrastructures are managed today (i.e., within application-centric "silos"), including

space for an application's primary storage, disaster recovery, testing and development, backup, and archive. Although silos provide good QoS, they can result in different processes for managing data, and they require too many experts.

The key to reducing hidden costs is to separate data from its physical containers and to use virtual copies of data whenever possible (e.g., snapshots, clones, and data mirrors). This can help reduce the physical storage requirements and consolidate storage infrastructure. In turn, this will reduce the number of processes and experts. Of course, a new management paradigm is required to deal with virtual copies. The approach taken by Network Appliance is to consolidate all data and copies associated with a particular application into a "data set." Data sets can have properties such as security, regulatory compliance, QoS, and namespace management. Indeed, future systems will see unified environments with user-specified properties on data sets. As such, properties can remain constant while the storage infrastructure adapts to advances in storage technology.

In summary, much of this is already starting to happen. "It's good to be in storage!" Steve proclaimed. There is another decade of interesting change (megatrends) ahead of us.

Rik Farrow (USENIX) asked whether there will be a reduction in the amount of storage because of virtual copies, and a similar question was posed by Chris Lumb (Data Domain). Steve said yes, but he also pointed out that it will be easier to create copies. Rik then asked if file systems will become more interesting. Steve said yes, as they will be forced to use the new abstractions for storage. Garth Gibson (Carnegie Mellon) made a comment in support of the Aperi open-source storage management framework. David Black (EMC) asked how we can prevent users from turning the QoS dials "all the way to the right." Steve said that, in practice, users will be presented with options that they will have to pay for (e.g., bronze, silver, and gold). Julian Satran asked how we can move forward to content management, as users are not interested in managing storage. Steve replied that although he did not talk about application integration, he expects storage to get tighter with the application (e.g., data sets administered through Oracle). An attendee from Berkeley asked how support will work for a single management infrastructure. Steve said that virtualization will make it difficult, but he expects that most of the deployments will occur in a "two worlds" scenario where applications can share the same physical infrastructure but still be managed separately.

THE LATEST VERSION

Summarized by Raja Sambasivan (rajas@andrew.cmu.edu)

■ *Design and Implementation of Verifiable Audit Trails for a Versioning File System*

Zachary N.J. Peterson, Randal Burns, Giuseppe Atensiese, and Stephen Bono, Johns Hopkins University

Zach Peterson presented a system capable of creating, managing, and verifying digital audit trails for versioning file systems. The digital audit system he presented involves three components: the file system, an authenticator, and an escrow site. The file system generates new versions of files and, when an audit trail is desired for a particular file version, commits an authenticator of that file version to an escrow site. The authenticator stored by the escrow site is a MAC that includes the authenticator of the previous version of the file and the data contained in the current version. An independent auditor can verify the contents of a given version of a file by first requesting both the version data and the previous version's authenticator and then comparing the auditor-constructed MAC based on this data to the MAC stored on the escrow site.

During his talk, Peterson noted that a central challenge the authors faced lay in finding a way to limit the amount of I/O necessary when computing authenticators. Computing an HMAC for a file, for example, requires that all of the file's data first be read into memory. To address this problem, the authors chose to use an XOR MAC as the authenticator. XOR MACs allow incremental authentication and thus allow the cost of authentication to scale with the size of changes to the data, not the size of the file.

The authors implemented their digital audit system in the ext3cow filesystem. Evaluation was performed using two micro-benchmarks and a trace-driven study, all of which showed significant gains for using an XOR MAC based on SHA-1 versus an HMAC based on SHA-1. Most of these gains were a result of the XOR MAC needing to perform less I/O than the HMAC. Most interestingly, Peterson pointed out that XOR MACs outperform HMACS especially well when the workload seen is dominated by small appends. These workloads can best take advantage of the incremental nature of the XOR MAC. Peterson then showed that write activity is dominated by appends via an analysis of his trace data.

During the Q&A period, Bill Bolosky from Microsoft asked whether the authors had considered using Merkle trees instead of the XOR MAC for their authenticators. Peterson responded by stating that they had considered Merkle trees, but had decided to use the XOR MAC instead because it is more efficient for data that changes via small incremental updates. David Black from EMC then asked whether the XOR MAC reveals information about what was authenticated and whether such transparency matters. Peterson's response was that it is the underlying MAC that

determines if any information is revealed. In this case the underlying MAC, SHA-1, does not reveal any information. Finally, Eduardo Pinheiro from Google wanted to know the size of the trace data used. Peterson stated that the trace data size was 4.2 gigabytes.

■ ***Architectures for Controller Based CDP***

Guy Laden, Paula Ta-Shma, Eitan Yaffe, Michael Factor, and Shachar Fienblit, IBM Haifa Research Laboratory

The ability to roll back to any previous storage state is clearly very useful and is exactly the functionality that continuous data protection (CDP) techniques provide. In this talk, Paula Ta-Shma compared four different storage architectures and provided analytic equations for reasoning about their cost when applied to different types of workloads. Finally, a trace-based study was used to validate the equations and compare the architectures on real workloads.

Paula Ta-Shma started by stating that the cost of using CDP is the number of user data device I/Os per write request in the common case (when no data is being reverted). This cost depends on four variables: the CDP granularity (specifically, the granularity of updates preserved), the workload (specifically, the temporal distance distribution of writes), the controller write cache (specifically, the size and replacement policy), and the CDP architecture in use.

Next, she described four different architectures for CDP: Logging, SlipStream, SplitDownStream, and Checkpointing. The Logging architecture is the simplest—the entire history of writes is stored in a log. Only one user data I/O per write request is incurred. However, although this architecture works well for write-dominated workloads, it does not allow for good read performance.

To remedy the read performance problem of the Logging architecture, the next two architectures split the version data into two volumes—a directly addressable "current store" that holds the current data and a hidden, mapped "history store" that holds all historical data, including the current version. The first of these architectures, SplitStream, splits write data above the cache; one copy of the write data is sent to each volume and thus a maximum of two user data I/Os and a minimum of zero user data I/Os are incurred per write request. Conversely, the SplitDownStream architecture splits write data underneath the cache. This architecture allows cache pages to be shared across current and historical volumes, thereby conserving memory resources. Compared to the Logging architecture, both the SlipStream and SlipDownStream architectures achieve better read performance, but they incur more I/Os and use more resources.

The Checkpointing architecture is similar to that of SplitDownStream, except that the history store only holds previous versions of write data. When a write is evicted from cache, the previous version of that data is copied from the current store to the history store. This architecture requires a maximum of three user data I/Os and a minimum of one.

Next, Paula compared the CDP cost of the various architectures when applied to a trace of the SPC-1 benchmark, which exhibits large temporal write distances, and the Cello99 trace workload, which exhibits smaller temporal write distances. For SPC-1, the results showed that SplitDownStream costs less than Checkpointing for smaller CDP granularities (i.e., less than 30 minutes). The same trend was observed for Cello99, except the cutoff point occurred near 5 minutes instead of 30 minutes. Additionally, the cost of Checkpointing declined to that of Logging at very coarse granularities.

At the end of the talk, an attendee asked whether the cost of SlipStream and SlipDownStream was one, not two, user data I/Os per write, since the I/Os to the current store and to the history store occur in parallel. Keith Smith asked whether different CDP architectures will need different metadata structures and thus differ in the cost of accessing metadata. Paula responded that in her CDP implementation, the underlying metadata structures for different architectures are the same, but they could be different.

■ ***Jumbo Store: Providing Efficient Incremental Upload and Versioning for a Utility Rendering Service***

Kave Eshgi, Mark Lillibridge, Lawrence Wilcock, Guillaume Belrose, and Rycharde Hawkes, HP Laboratories

Many providers would like to provide batch services to customers, in which clients send some data to the provider and the provider performs some large computation on the data (e.g., finite element analysis, data mining) and then sends the results back. However, providing batch services is difficult because the links used to transfer data from the customer to the provider (e.g., over the Internet via ADSL) tend to be very slow. In this talk, Kave Eshgi addresses this problem by describing a new storage system called Jumbo Store, which uses Hash-Based Directed Acyclic Graphs (HDAGs) to provide incremental upload of filesystem snapshots from a Jumbo Store client to a Jumbo Store server. The authors claim that incremental upload is a solution to the transfer problem since, in many cases, new client jobs use data that is only slightly different from previous jobs. Eshgi noted that Jumbo Store had been experimentally evaluated by incorporating it into the HP Labs prototype utility rendering service located in Palo Alto, CA, and used by animators in England to create a number of high-quality animated shorts.

To provide some intuition about HDAGs, Eshgi noted that HDAGs are a generalization of Merkle trees. However, he was adamant that HDAGs not be called Merkle trees, as DAGs, unlike trees, can contain nodes with multiple parents. Also, unlike Merkle trees, nonleaf nodes can contain data in an HDAG.

An HDAG is a directed acyclic graph (DAG) whose nodes refer to other nodes by their hash rather than their location in memory. Each node in a HDAG is comprised of two fields: the pointer field, which is a possibly empty array of hash pointers, and the data field, which is an application-defined byte array. Jumbo Store encodes the directory structure into an HDAG by representing each directory as an HDAG node whose data field contains that directory's metadata and whose hash pointers point to the directory's members. Conversely, a single file is represented as a two-level HDAG; the first-level node's data field contains the metadata for the file and its pointer field points to a node containing the file contents. To avoid having to send the entire file contents every time a small portion is changed, the file contents are broken into chunks via a technique called content-based chunking.

Eshgi next described the utility rendering service (URS) in which Jumbo Store was used. The URS is a batch utility service that performs the calculations required to render a 3D movie. Animators in the U.K. would use the URS client to submit rendering jobs to the URS in Palo Alto. To evaluate Jumbo Store, a subset of the data uploaded by the animators was re-uploaded using rsync (with the compress option turned on) and the difference in actual data transferred was noted. The authors found that Jumbo Store, on average, transferred half as many bytes as rsync.

During the Q&A period, an attendee asked whether the chunk sizes used to split up the file contents were static. Eshgi responded affirmatively, stating that a static chunk size of 4KB was used. Another attendee wanted to know whether Jumbo Store was subject to large latencies when uploading information. Eshgi answered that latency was large, but that latency was a relatively inconsequential metric given the usage scenario of Jumbo Store within the URS. Specifically, the animators would often leave or work on other things as soon as they were convinced that the incremental upload had started. Finally, Keith Smith asked whether the feedback from the animators using the system had been positive. Eshgi responded that the animators liked the system.

SCALABLE SYSTEMS

Summarized by Avishay Traeger (atraeger@cs.sunysb.edu)

■ *Data ONTAP GX: A Scalable Storage Cluster*

Michael Eisler, Peter Corbett, Michael Kazar, and Daniel S. Nydick, Network Appliance; J. Christopher Wagner, Ironport Systems, Inc.

Peter Corbett began by describing Data ONTAP GX as a scalable clustered network file server composed of a number of cooperating filers. The storage of a large number of filers can be presented as a single shared storage pool. The system's key features are scalability (by allowing for the easy addition of filers to the cluster), location transparency of data within the cluster, an extended namespace that can span multiple filers, increased resiliency in the face of failures, and simplified load and capacity balancing.

The system exports both NFS and CIFS protocols to clients via virtual interfaces (VIFs). Requests are initially processed by the networking blade (N-Blade), which terminates incoming NFS and CIFS connections and maintains protocol-specific state. The requests are translated into SpinNP RPCs, which are transmitted over a cluster fabric to the server responsible for the target volume, where a volume is a filesystem subtree and volumes are grouped into aggregates. SpinNP calls are processed by the data blade (D-Blade) on the target server. Two cluster-wide databases are used to route requests and responses. The volume location database (VLDB) tracks the corresponding aggregate for each volume, as well as the D-Blade currently responsible for the aggregate. The VIF manager database tracks which N-Blade is currently hosting each virtual interface, so that D-Blades can send callbacks to clients.

Data ONTAP GX is the first system to achieve one million operations/s on the SPEC SFS benchmark. In addition, it scales linearly on all benchmarks up to 24 nodes. The product is already deployed at customer sites and provides a powerful set of features that go well beyond what a stand-alone file server offers.

Gregory Touretsky (Intel) asked whether performance was measured on single or multiple D-Blades, and the reply was that benchmarks were run across multiple volumes and multiple D-Blades. Someone asked what happens to performance when they go beyond 24 nodes, and the audience laughed. The reply was that this is what is currently being shipped, and they are working on expanding the system further. There was a question about what was used to benchmark performance via the CIFS interface, and if any results were available. The reply was that there is no standard CIFS benchmark, and any results that they have are not publicly available. Another questioner wondered why write throughput was less than half of read throughput. The answer was that it is probably the result of read-ahead.

■ *//TRACE: Parallel Trace Replay with Approximate Causal Events*

Michael P. Mesnier, Intel Research with Carnegie Mellon University; Matthew Wachs, Raja R. Sambasivan, Julio Lopez, James Hendricks, Gregory R. Ganger, and David O'Hallaron, Carnegie Mellon University

Michael Mesnier stated that the goal of //TRACE is to extract traces of parallel applications and replay them in such a way that the I/O behavior is true to the original workload. This is done by discovering internode data dependencies and inter-I/O compute times. Replaying the trace as fast as possible is one classic option, but performance is poor because it assumes an I/O bottleneck, there is no idle

time, and dependencies are ignored. The other classic replaying option is to use the original timing, but with this technique we would not see any changes in performance when replaying on different hardware.

//TRACE is portable and treats the application and storage system as a black box. To find the dependencies, the workload is run multiple times, each time adding delays to the I/O stream of a node. I/O dependencies can be found when nodes block on the I/O that is currently being throttled. This will also find computation times. It then annotates the per-node traces with the dependencies. These annotations allow a parallel replayer to closely mimic the behavior of a traced application across a variety of storage systems. The number of runs and the rate at which I/O is throttled determine how many data dependencies can be discovered, as well as the time necessary to collect the traces. (One can trade off running time and replay accuracy.) Once //TRACE has collected this information, it can adjust with the speed of the storage system while enforcing dependencies. This technique works well for parallel applications with deterministic I/O dependencies.

The causality engine that //TRACE utilizes is implemented as an LD_PRELOAD library, allowing it to capture file-level traces. This allows users to properly evaluate different file and storage systems. Compared to other replay mechanisms, //TRACE offers significant gains in replay accuracy. Overall, the average replay error for the three parallel applications evaluated is below 10% when throttling every node and delaying every I/O. Trading off replay errors with greater time in collecting the traces was explored.

One question was asked about how //TRACE orders events within a node. Mesnier explained that it is actually the threads that are being throttled, and so this is not an issue. Another point raised was that most HPC workloads today are performing nondeterministic I/O, where this technique will not work. The reply was that there are many workloads of both types, and this work targets the deterministic applications. A follow-up point was that many HPC centers have clusters that are being shared by several applications, so even a deterministic application will act in a nondeterministic fashion. Mesnier responded that they are targeting dedicated clusters, which are often used for larger applications. Another question involved how one could deal with bottlenecks within the applications. The answer was that one could perform static analysis on the traces to see if there are too many barriers, for example, or possibly visualize the traces. In response to whether they had tried replaying the traces on systems other than those where the trace was collected, the reply was that this is what they had in fact done in their evaluation.

CACHE PRIZES

Summarized by Raja Sambasivan (rajas@andrew.cmu.edu)

■ *Karma: Know-It-All Replacement for a Multilevel Cache*

Gala Yadgar, Technion; Michael Factor, IBM Haifa Research Laboratories; Asaaf Schuster, Technion

Gala Yadgar introduced Karma, a system for optimizing cache replacement in systems with multiple levels of caching. Yadgar noted that multilevel cache hierarchies introduce three major problems in cache replacement. First, locality information is hidden from lower-level caches by the upper-level caches. Second, cache space is wasted because blocks are often duplicated at multiple cache levels. Finally, contextual information about blocks (e.g., the file to which they belong, the application that issued the I/O request, etc.) is also often hidden from the lower-level caches. Karma addresses all of these problems in concert. First, hints from applications are used in all cache levels to divide disk blocks into ranges based on their expected access pattern and access frequency; each range is allocated its own cache partition (which may span multiple cache levels) and replacement policy. The data blocks contained in each partition are managed by Karma using the READ, READ-SAVE, and DEMOTE operations. READ forces the lower-level cache to delete a block whenever it is read by an upper-level cache. Conversely, READ-SAVE allows a lower-level cache to keep a block read by an upper-level cache. Finally, DEMOTE sends a block evicted from an upper-level cache back to the next lower-level cache. Overall, by using application hints and partitioning the cache, Karma is able to use the optimal replacement policy for each access pattern seen. As a result, it compares favorably to all other cache replacement techniques (e.g., LRU, ARC, MultiQ).

There are two key design decisions in Karma. The first lies in how it partitions the workload it sees into ranges. The second revolves around the mechanism it uses to allocate a cache partition for each range. Workloads are partitioned into ranges based on application hints. Blocks in a given range have similar access frequencies and are accessed in a similar fashion. Application-level hints are propagated to lower-level caches by attaching a range identifier to each cache block. Karma allocates a cache partition for a given range in such a way as to maximize the "normalized marginal gain" for that range. The marginal gain for an access trace is the increase in hit rate that will be seen by this trace if the cache size increases by a single block. For example, for sequential accesses, the marginal gain is always zero; conversely, for looping accesses, the marginal gain is constant until the entire loop fits in cache and is zero afterward. Ranges with higher normalized gains are allocated cache partitions in higher cache levels, since it is likely that blocks in these ranges will be accessed more frequently.

During the Q&A session Jason Flinn from the University of Michigan asked how Karma dealt with access patterns that changed too quickly for Karma to optimize for them. Yadgar explained that such access patterns are not handled by Karma, but their effects are somewhat minimized because Karma does not yet perform any prefetching. John Wilkes from HP Labs then asked whether the authors had thought about using Karma in a multiple host setup. As an example, Wilkes noted that several changes had to be made to DEMOTE to adapt it to work with multiple hosts. Yadgar responded by stating that this was future work and that, for the multiple-host case, the authors would have to look into replacement policies that take into account the fact that blocks are not necessarily discarded when they leave a given host.

■ ***AMP: Adaptive Multi-Stream Prefetching in a Shared Cache***

Binny S. Gill and Luis Angel D. Bathen, IBM Almaden Research Center

In this very entertaining talk, Binny Gill provided an analysis of sequential prefetching algorithms for the case where an LRU cache houses prefetched data for multiple concurrent sequential streams. He first separated current sequential prefetching algorithms into four different classes based on two criteria: whether the prefetching algorithm chooses the prefetch size in a fixed manner or in an adaptive manner and whether the prefetching algorithm prefetches in a synchronous or asynchronous manner. He then showed that, of these classes, Adaptive Asynchronous (AA) prefetching algorithms show the most promise for sequential prefetching in terms of minimizing cache pollution and wasted prefetches. Next, a formal analysis showing how to best adaptively pick the parameters p (the prefetch distance) and g (the distance at which a prefetch is triggered) for AA prefetching algorithms was shown. Finally, Binny described AMP, an implementation of an AA algorithm, and experimentally showed its superiority to other types of prefetching algorithms for sequential workloads.

During the talk, Binny described his taxonomy of different prefetching algorithms by using an animation showing customers reaching for and eating donuts on a table while a waiter strove to "prefetch" more donuts according to the semantics of the various prefetching algorithm classes. His taxonomy classifies prefetching algorithms into four different classes: Fixed Synchronous (FS), Fixed Asynchronous (FA), Adaptive Synchronous (AS), and Adaptive Asynchronous (AA). The first term in this nomenclature refers to whether the prefetching algorithm adaptively chooses the prefetch size. The second term refers to whether prefetching is carried out synchronously or asynchronously. Binny noted that although AA prefetching algorithms show the most promise for sequential streams, they have rarely been implemented in practice.

Next, Binny quickly described a formal analysis showing how best to adaptively choose the prefetch size, p, and prefetch trigger distance, g, for each independent stream when using an AA prefetching algorithm. This information was then used to describe the Adaptive Multi-Prefetching (AMP) algorithm. This algorithm adapts the prefetch size by decreasing the value of p whenever a prefetched page reaches the end of the LRU list unaccessed. To minimize wasted prefetches, such unaccessed prefetched pages are also moved back to the head of the LRU list and marked as "old" when they fall off the end of the LRU list. Conversely, the value of p is incremented whenever the last page read within a given read I/O hits in cache and the cache block hit is not "old." The size by which p is incremented is the size of the read operation in pages. The prefetch trigger distance, g, is incremented whenever a prefetch returns only to find that a read request is already waiting for some page in the prefetch set. The value of g is decremented when p is decremented.

Finally, Binny showed that the AMP algorithm easily outperforms the other classes of prefetching algorithms on several different workloads. These workloads included a sequential stream workload, a workload with many short sequential sequences, the SPC-1 Read workload, and the SPC-2 Video on Demand workload.

During the Q&A session, John Wilkes from HP Labs asked whether the authors had tried to use AMP on mixed workloads. Binny responded that they hadn't explored this space yet, but that the SPC-1 Read workload, on which AMP performed well, is not a strictly sequential workload.

■ ***Nache: Design and Implementation of a Caching Proxy for NFSv4***

Ajay Gulati, Rice University; Manoj Naik and Renu Tewari, IBM Almaden Research Center

Sharing data across wide area networks (WANs) is becoming very common; however, a common problem with WAN file sharing is high latency. A common solution used to reduce latency in WANs involves installing a caching proxy close to the client, but many protocols available for use by the caching proxy and the server, such as CIFS and NFSv2/v3, are optimized for local area networks (LANs), not WANs. As a result, these protocols tend to be "chatty" and result in suboptimal performance. In this talk, Ajay Gulati described why NFSv4 is a good choice for the protocol that should be used between caching proxies and servers and then proceeded to describe the design and implementation of such a NFSv4 caching proxy (Nache).

Gulati stated that NFSv4 is useful in a caching proxy setup because it provides read/write delegations and compound requests. A write delegation for a file issued to a client by an NFSv4 server gives the client the authority to modify that file locally without interacting with the server until the delegation is revoked. Similarly, a read delegation for a file issued to a client allows the client to read the file without continually checking cache consistency. For cases

where sharing of files among clients is uncommon, delegations can greatly improve performance. Compound requests allow multiple related requests to be batched into a single request; such requests are suited for WANs as they generate less overall network traffic and per-command round-trip delays than the case in which each request contained in the compound request is issued separately.

Ajay then described Nache, a caching proxy for NFSv4 that sits between local clients and a remote server. Nache, in some sense, relies on receiving delegations from the server for the files accessed by its clients. If it does receive these delegations, it can proceed to serve the clients locally without communicating with the server.

Nache is implemented via cascaded mounts. Nache mounts the data exported by the remote server and then re-exports this mount to clients. This means that when a client request must be forwarded from the Nache to the remote server, the request must first be translated from an NFSv4 request to a VFS call and then back to an NFSv4 request. This translation works without issue in most cases; however, some stateful NFSv4 requests (e.g., OPEN, CLOSE, LOCK, REQUEST) require special handling.

Ajay then proceeded to provide an evaluation of both Nache and delegation performance in NFSv4. He showed that delegations can greatly reduce the number of operations at the remote server, but that the cost of issuing a delegation is high. Hence, delegations are useful as long as the server does not have to revoke them often (owing to conflicting accesses, etc.). The authors evaluated Nache by using the filebench benchmark, which was used to approximate both a Web-based workload and an OLTP workload, and by executing a software build on the clients. The results showed that as long as more than one client is used, the fraction of total operations generated by clients sent to the remote server was substantially reduced. Finally, the authors measured response time over the WAN when executing the software build. They found that the server response time decreased as the number of clients used increased.

One attendee asked what would happen if the remote server were to revoke the delegations issued to Nache. Ajay responded that Nache would then stop acting as a proxy and start acting as a simple passthrough.

BEYOND THE MACHINE ROOM

Summarized by Andrew Leung (aleung@soe.ucsc.edu)

■ *TFS: A Transparent File System for Contributory Storage*

James Cipar, Mark D. Corner, and Emery D. Berger, University of Massachusetts, Amherst

Awarded Best Paper!

James Cipar discussed how contributory applications such as Folding@home and Freenet utilize a user's unused local disk space to contribute to a common distributed system. This disk utilization intrudes on users' personal free space as well as impacting normal application performance. Namely, as used disk space increases, the file system's ability to make ideal block allocation decisions decreases. This leads to disk layout fragmentation, which has a very negative impact on disk access times. James continued by saying that for users to want to use contributory applications, these applications must not consume too much disk space.

To address this issue the authors presented the Transparent File System (TFS), an on-disk file system that allows users to contribute disk space with minimal performance impact. James outlined the goals of TFS: to make contributory data transparent, meaning their presence has no impact on filesystem performance or capacity; to allow contributory data to be overwritten when space for user data is needed; and to make such data compatible with legacy contributory applications. To facilitate overwriting of contributory data, TFS uses five block allocation states to specify whether data is contributory, contributory and overwritten by user data, or overwritten contributory data that has been deleted. James evaluates the contributions of TFS with respect to a watermarking approach and using fixed contribution space. The two key metrics are the amount of storage capacity contributed and bandwidth. Their figures show TFS is able to contribute more space, as well as maintaining higher bandwidth. To examine local filesystem performance, the Andrew Benchmark is used against different amounts of data contribution. James presents the TFS run time for several benchmark phases, each of which is better than or comparable to ext2 with or without any contributory application running.

Brent Callaghan from Apple asked whether TFS could be used as a Web browser cache. James replied that in fact they had tried to use TFS as a browser cache but that because browser cache size is generally small the effects are negligible. Another question related to files that are open for extended periods of time. Since TFS does not overwrite contributory data from an open file it is conceivable that a contributory application could cause fragmentation. James confirmed that this is the case but that perhaps some effort should be made to ensure that contributory applications do not hold files open too long. Binny Gill from IBM Research asked whether ext2 had to be modified. James said that indeed ext2 was modified and that porting TFS to another file system would require the source code. A second question was whether movie data from sources such as BitTorrent is affected by TFS. James answered that BitTorrent data is user data, rather than contributory data, and therefore TFS does not affect it; rather, BitTorrent consumes other resources, such as bandwidth during uploads. Finally, James provided a link to more TFS information and source code: http://prisms.cs.umass.edu/tcsm/.

■ ***Cobalt: Separating Content Distribution from Authorization in Distributed File Systems***

Kaushik Veeraraghaven, Andrew Myrick, and Jason Flinn, University of Michigan

Kaushik Veeraraghavan presented a solution for secure mobile content access. In addition to accessing digital content from personal devices, Kaushik said that users often wish to access content from other devices to share media with friends or family. He described how these ad hoc clients, which are devices such as MP3 players or laptops that a user does not own or rarely uses, make access to digital data difficult. The current data distribution and authorization model makes accessing data from ad hoc clients difficult because the ad hoc client must be used to explicitly locate and fetch remote content after searching for it over a slow, wide-area link. Kaushik described how digital rights management adds additional complexity by forcing users to authenticate themselves at each ad hoc device, opening the door for possible abuse, and often requires the user to jump through hoops to play back the content. What makes protected content special is that the goals of users and content providers are largely orthogonal. Users desire easy and pervasive access to their data, whereas providers do not want data leaked to any unauthorized device or user.

To address this, Kaushik described a shift in paradigm where people, rather than clients, are authenticated. Their solution, Cobalt, relies on a Cobalt token, such as a cell phone or PDA, which is carried by the user, allowing authentication to be based on proximity. Kaushik said the goal of Cobalt is to improve usability, privacy, and content protection. This relies on trusting the Cobalt token and ad hoc media players, each of which has a Trust Platform Module (TPM) chip that allows content providers to verify the integrity of the Cobalt token. Kaushik described their current implementation on the Blue File System. The Cobalt token manages content acquisition, where the provider forwards encrypted content to either the token or a more powerful helper computer. Playing the content requires the token to identify all media players in range, select a specific media, and send either the data from the token or the IP address of the helper computer to the media player.

Evaluation of Cobalt aims at understanding the overhead of content acquisition and content playback and any new applications that Cobalt may enable. The content acquisition has a fixed 10-second overhead that scales linearly with file size. Kaushik attributes most of this cost to the establishment of a secure connection. Associating a media player with the token and specifying content to share requires about 12 seconds. Finally, Kaushik describes how Cobalt can be used to adaptively merge many users' playlists into a single playlist, enjoyable by all. Each user specifies a query and only songs that appear in all query results are played.

Peter Honeyman from the University of Michigan asked about clarifying whether it is the TPM or the media player that is trusted, since the media player can still steal unencrypted content. Kaushik responded that the media player must also be trusted and that ideally DRM is enforced all the way until it reaches the user's ear but that this obviously cannot be done. Brent Welch from Panasas asked if Cobalt would make it possible to steal media from TiVo, to which Kaushik replied that TiVo does not have a TPM chip, which makes it untrusted. Daniel Ellard from Network Appliance asked about how content can be retrieved should the Cobalt token be lost. Kaushik said content providers could deauthorize the token or one could encrypt the content key and resynchronize with a new token. Finally, Craig Everhart from Network Appliance wondered how Cobalt defined media players in range and whether it would be possible to accidentally access a neighbor's media player. Kaushik responded that it is indeed possible but that simply registering tokens with specific media players could alleviate that problem.

MAKING THE RAID

Summarized by Charles Weddle (weddle@cs.fsu.edu)

■ ***PARAID: A Gear-Shifting Power-Aware RAID***

Charles Weddle, Mathew Oldham, Jin Qian, and An-I Andy Wang, Florida State University; Peter Reiher, University of California, Los Angeles; Geoff Kuenning, Harvey Mudd College

Charles Weddle began his talk by discussing the increasing concern of energy consumption in server-class machines. Storage systems account for a significant part of the energy consumption in servers. In fact, storage systems account for 24% of the power usage in Web servers and 27% of the electricity cost for data centers. The authors hypothesize that it is possible to reduce energy consumption in RAID devices without degrading performance, while maintaining reliability. The three main challenges in this problem are finding opportunities to save energy in active servers, preserving peak performance, and maintaining reliability. The existing work mostly trades performance for energy savings directly, for example by varying the speeds of disks. The authors took advantage of three observations to help provide a solution to this problem. First, conventional RAID overprovisions resources: A conventional RAID keeps all disks spinning even under light load. Second, unused storage exists on storage servers because of overprovisioning. This unused storage can be used for energy savings. Third, workloads have a cyclic fluctuation. Infrequent disk power transitions during periods of light load can save energy.

The authors present the power-aware RAID (PARAID) as a solution to this problem. PARAID introduces skewed striping that allows PARAID to power off disks during periods of light load. Skewed striping replicates blocks in unused

storage, allowing disks to be powered off. Skewed striping creates sets of disks of varying sizes, where each disk set is thought of as a gear. Skewed striping allows PARAID to match performance to workload by switching into different gears. PARAID preserves peak performance by operating in the highest gear, with all disks in the array, when the system is under peak load. PARAID maintains reliability by reusing existing RAID levels, for example RAID level 5. The authors implemented a PARAID prototype in Linux 2.6.5 for evaluation. The PARAID components reside between the conventional RAID device driver and the disk device driver. The evaluation was performed by replaying a Web trace and a Cello99 trace and running the Postmark benchmark. For the Web trace experiments the PARAID device, using RAID level 0, was compared to a conventional RAID level 0 device and was found to reduce energy consumption by up to 34%. For the Cello99 experiments the PARAID device, using RAID level 5, was compared to a conventional RAID level 5 device, yielding up to 13% energy savings. PARAID is ongoing work and the authors would like to test PARAID under more workloads. Also, the authors would like to put PARAID into a production environment for live testing.

Bill Bolosky from Microsoft asked why the latency CDF graphs for the Web trace experiments have sustained peaks. Charles responded that this was caused by a small number of Web requests never finishing owing to the way the Web trace playback program handles certain errors. Keith Smith from Network Appliance asked whether RAID level 4 would make disk power transitions easier. Charles responded that it might help alleviate the overhead of cascading parity updates. Carl Waldspurger from VMware asked how the optimal gear configuration is chosen. Charles explained that gear optimization is ongoing work and that iterating over gear configurations using simulation might help with this problem.

■ ***REO: A Generic RAID Engine and Optimizer***

Deepak Kenchammana-Hosekote, IBM Almaden Research Center; Dingshan He, Microsoft; James Lee Hafner, IBM Almaden Research Center

Deepak Kenchammana began his talk by discussing the need for a variety of RAID codes. No single RAID code satisfies all aspects of data storage in terms of storage efficiency, reliability, and performance. Also, as data sets grow, using the same RAID code is not practical because disk failures grow with capacity. A greater variety of RAID codes are needed to provide high data reliability using less reliable disks. This is challenging because it is expensive to support several RAID codes. The current mindset is that deploying new RAID codes is expensive and risky. The authors present the RAID Engine Optimizer (REO) as a solution to this problem. REO can handle any XOR-based RAID codes, including N-way mirroring. REO automatically handles all RAID-related errors and leverages dynamic-state-like current cache pages. REO offers competitive performance, yielding ~8% speedup for SPC-1-like workloads.

REO comprises a set of routines invoked by cache on read (on miss), write (flush), rebuild, and migrate. REO routines deal with all necessary RAID translations and executions. REO extends the idea that it is efficient to simultaneously flush dirty pages in cache that belong to the same stripe by defining a W-neighborhood for the victim page. The W-neighborhood is defined as the set of all pages, clean or dirty, in the data cache that are in a 2W + 1 stripe window centered on the victim's stripe. REO is made up of a RAID engine and an execution engine. The RAID engine determines the best strategy for what is to be done and the execution engine figures out how it gets done. Input into the execution engine from the RAID engine is an I/O plan that contains the set of blocks to be read, a set of blocks to be XOR-ed, and a set of blocks to be written. The read strategy for fault-free read is straightforward: Just read it from disk. The challenge with read is the reconstruct case. The write strategy involves identifying all affected parity elements for dirty elements to be written. For every affected parity element there are two update strategies: Parity Compute (PC) and Parity Increment (PI). The execution engine takes the I/O plan and acquires stripe locks, allocates cache pages, submits reads in the read set, executes XORs in the XOR set, and finally submits the writes in the write set. If an error is detected during the execution of an I/O plan, the execution engine aborts the I/O plan and resubmits the I/O plan to the RAID engine. The RAID engine contains an I/O plan optimizer that is responsible for picking the least-cost read or write strategy. Some metrics used for optimization are the number of I/O commands (IOC) submitted to the storage devices and the number of XOR operations. REO was evaluated through trace-driven simulations. The authors built the data cache and REO components for evaluation; Disksim was used to simulate disk I/O. The optimization objective for evaluation was to minimize disk I/O (IOC). Two measures were used in evaluation: total access time and total memory bus usage. REO showed a 10% improvement in disk access time for a fault-free RAID 5 layout and a 7% improvement for a RAID 5 layout with one disk failure.

The first questioner asked what REO would do under page pressure (thrashing). Deepak answered that REO would do nothing worse than any other RAID firmware would do. One way to reduce the additional pages needed to complete in-progress writes is to reduce the window size. If that does not help then you are really stuck with operating at a point where things will complete, albeit slowly. To the second question, whether the authors considered having REO look more broadly into the file cache, Deepak answered no. In response to whether there was any chance of seeing the source code, Deepak stated that he is sure that IBM has plans for it to be made public.

■ ***PRO: A Popularity-Based Multi-Threaded Reconstruction Optimization for RAID-Structured Storage Systems***

Lei Tian and Dan Feng, Huazhong University of Science and Technology; Hong Jiang, University of Nebraska—Lincoln; Ke Zhou, Lingfang Zeng, Jianxi Chen, and Zhikun Wang, Huazhong University of Science and Technology and Wuhan National Laboratory for Optoelectronics; Zhenlei Song, Huazhong University of Science and Technology

Hong Jiang began his talk by discussing the importance of data recovery. Disk failures have become more common in RAID-structured storage systems. The improvement in disk capacity has far outpaced improvements in disk bandwidth, lengthening the overall RAID recovery time. Also, disk drive reliability has improved slowly, resulting in a very high overall failure rate in a large-scale RAID storage system. Disk-oriented reconstruction (DOR) is one of the existing I/O parallelism-based recovery mechanisms. DOR follows a sequential order of stripes in reconstruction, regardless of user access patterns. Workload access patterns need to be considered because 80% of the accesses are directed to 20% of the data, according to Pareto's Principle, and 10% of the files accessed on a Web server typically account for 90% of the server requests. The authors present a popularity-based multi-threaded reconstruction optimization (PRO) that takes advantage of data popularity to improve reconstruction performance. PRO divides data units on the spare disks into hot zones. Each hot zone has a reconstruction thread. The priority of each thread is dynamically adjusted according to the current popularity of its hot zone. PRO keeps track of the user accesses and adjusts the popularity of each hot zone accordingly. PRO selects the reconstruction thread with the highest priority and allocates a time slice to it. When a thread's time slice runs out, PRO assigns a time slice to the next highest priority thread. The process repeats until all of the data units have been rebuilt. Priority-based scheduling is used so that the reconstruction regions are always the hottest regions. Time-slicing is used to exploit the I/O bandwidth of hard disks and access locality.

PRO was compared to DOR in the evaluation because DOR is arguably the most effective among the existing reconstruction algorithms. The evaluation of PRO examined reconstruction performance by measuring user response time, reconstruction time, and algorithm complexity. PRO was integrated into the original DOR approach implemented in the RAIDframe software to validate and evaluate PRO. The evaluation was performed by replaying three different Web traces that consisted of read-only Web search activity. It was found that PRO consistently outperformed DOR in reconstruction and user response time by up to 44.7% and 23.9%, respectively. PRO's effectiveness relies on the existence of popularity and locality in the workload as well as the intensity of the workload. PRO also uses extra memory for each thread descriptor. The computation overhead of PRO is $O(n)$, although if a priority queue is used in the PRO algorithm the computation overhead can be reduced to $O(\log n)$. The entire PRO implementation in the RAIDFrame software only added 686 lines of code. Work on PRO is ongoing. Future work includes optimizing the time slice, scheduling strategies, and hot zone length. Currently, PRO is being ported into the Linux software RAID. Finally, the authors plan on further investigating use of access patterns to help predict user accesses and of filesystem semantic knowledge to explore accurate reconstruction.

The first questioner asked about the average rate of recovery for PRO. Hong answered that the average reconstruction time is several hundred seconds in the experimental setup. The second questioner asked how well PRO reconstruction compares to DOR reconstruction under no workload. Hong commented that when there is no workload the reconstruction performance for PRO and DOR is the same. In response to a question about write overhead, Hong stated that his research team is actively looking into this. The last question involved the sensitivity of the results to the number of threads and to the time slice. Hong explained that the impact of the number of threads and time slice is negligible in the current experimental configuration. However, a more elaborate sensitivity study is underway in the project.