This issue reports on the First Symposium on Networked Systems Design and Implementation (NSDI '04) and on the 3rd USENIX Conference on File and Storage Technologies (FAST '04).

For NSDI '04: Our thanks to Amin Vadhat, who shepherded the following summarizers:

Magdalena Balazinska
Laura Grit
Vinay M. Igure
Suchita Kaundin
Chip Killian
Ramakrishna Kotla
Xun Luo
Vinay Mallikarjun
Piyush Shivam
Chunqiang Tang
Praveen Yalagandula
Aydan Yumerefendi

For FAST '04: Thanks to Ismail Ari and his summarizers:

Michael Abd-el-Malek
Nitin Agrawal
Akshat Aranya
Dean Hildebrand
Andrew Klosterman
Xun Luo
Kiran-Kumar Muniswamy-Reddy
Steve Schlosser
Shafeeq Sinnamohideen
Deepa Tuteja
Wenguang Wang
Lan Xue
Aydan Yumerefendi

Note: The reports on BSDCon '03, held in San Mateo, California, September 8–12, 2003, can be found at *http://www.usenix.org/events/bsdcon03/confrpts.pdf*

3rd USENIX Conference on File and Storage Technologies (FAST '04)

March 31–April 3, 2004
San Francisco, California

TECHNICAL SESSIONS

RELIABILITY & AVAILABILITY

Summarized by Kiran-Kumar Muniswamy-Reddy

Best Paper award

Row-Diagonal Parity for Double Disk Failure Correction

Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar, Network Appliance, Inc.

Peter Corbett described a new algorithm, Row-Diagonal Parity (RDP), for protection against double failures and described its application to RAID.

The RDP algorithm uses a simple parity scheme based on EX-OR operations. Each data block belongs to one row-parity set and one diagonal parity set. In a simple RDP array, there are p + 1 disks. The stripes across the array consist of one block from each disk. In each stripe, one block holds diagonal parity, one block holds row parity, and p – 1 blocks hold data. Every row parity block has an even parity of the data blocks in the row, excluding the diagonal parity block. Every diagonal parity block has an even parity of the data and row parity blocks in the same diagonal. In case of double disk failure, each diagonal misses one disk, and there are two diagonal parity sets that miss only one block. Once we recover one block, we can recover a complete row. Using this, we recover another diagonal, and so on.

In the Q&A session, someone asked why weren't triple (or more) failures considered. The speaker said that double failures are the more common case and that this could be extended for triple failures. Another asked whether the algorithm could really be added to an existing RAID, as the paper claims. The speaker replied that it could be added to an existing RAID.

Improving Storage System Availability with D-GRAID

Muthian Sivathanu, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

This paper won one of the two Best Student Paper award.

The talk was given by the primary author, Muthian Sivathanu. Normal RAID works on a simple failure model. The basic premise of this work is that if D or fewer disks fail, RAID or D-GRAID continues to operate with some degraded performance. The disk array becomes completely unavailable once more than D disks fail. D-GRAID degrades gracefully and also recovers quickly in the event of a disk failure.

For graceful degradation, D-GRAID uses two techniques: selective metadata replication and fault-isolated data placement. In selective metadata replication, the naming and system metadata structures of the file system are highly replicated. Thus, failures of a few disks do not render the entire array unavailable. In fault-isolated data placement, semantically related blocks are placed within the array's unit of fault-containment. For example, all the blocks of a file are placed within a disk. This way, semantically meaningful data units are available under failure. Since fault-isolated data placement reduces the parallelism inherent in RAID, they make copies of blocks of "hot" files across the drives of the system.

MEASUREMENT, MODELING, AND MANAGEMENT

POLUS: GROWING STORAGE QOS MANAGEMENT BEYOND A "FOUR-YEAR-OLD KID"

Sandeep Uttamchandani, Kaladhar Voruganti, John Palmer, and David Pease, IBM Almaden Research Center; Sudarshan Srinivasan, University of Illinios at Urbana-Champaign

Summarized by Nitin Agrawal

Sandeep Uttamchandani described the Polus framework, which addresses the QoS goal transformation problem in the context of policy-based storage management.

In spite of prior research and standardization, the problem of mapping high-level QoS goals to low-level storage device actions still yields complex and error-prone processes. Polus generates this mapping by using a combination of rule-of-thumb specifications, a reasoning engine, and a learning engine. Currently, policies are specified as a collection of rules in <event, condition, action> format, and the management module simply invokes the appropriate rule. The problems with this approach are complexity (the level of detail required in generating the rules) and brittleness with respect to changing system configurations and workloads.

Polus performs the balancing act by requiring only high-level specifications from the administrator and using machine learning and pre-packaged procedures for the rest. The paradigm essentially consists of three parts: taking input from toolkit user, quantifying the relations using learning functions, and base strategies.

The work illustrates how Polus can provide storage management guidance to a simulated Storage Area Network file system. A quantitative comparison of Polus with rule-based systems is done for different system states. A future direction of work is to handle incomplete and even incorrect specifications.

During the Q&A, someone pointed out that the decision of when to perform prefetching, which was described as a candidate for a Polus-based learning framework, would be good only if it is taken quickly. Sandeep's response was that they used the prefetching example only as an illustration. Managing a real system using the Polus paradigm is currently under implementation. In summary, the paper proposes Polus as a conceptual stepping stone in the direction of "non-rule"-based approaches for automated system management.

In normal RAID, recovery is slow, as all the blocks in a disk must be recovered. D-GRAID makes recovery fast by recovering only the "live" file system data, i.e., data that is currently in use by processes.

BUTTRESS: A TOOLKIT FOR FLEXIBLE AND HIGH FIDELITY I/O BENCHMARKING

Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, and Ram Swaminathan, Hewlett-Packard Laboratories

Summarized by Steve Schlosser

Mustafa Uysal presented Buttress, a flexible, highly accurate I/O generation tool. Buttress can be configured both as a synthetic workload generator and as a trace replayer. Reproducing very heavy I/O workloads with high fidelity requires a great deal of care, especially to correctly handle bursts of traffic, which exist in many workloads. The authors contend that getting this right requires I/Os to be issued within 100 microseconds of their intended arrival time. Another goal of Buttress is to be as portable as possible and not to require modifications to the host operating system. With this goal in mind, Buttress is implemented as a user-level program, working with standard pthreads on both Linux and HP-UX. It requires a multiprocessor machine to produce the desired I/O loads of up to 100,000 I/Os per second.

In order to generate a workload with very high fidelity, Buttress has to deal with many issues. First, Buttress has to minimize latency to shared data structures by reducing contention at high load. It does this by minimizing the number of lock operations, minimizing critical section time, and using bypass locking. Second, Buttress must minimize the impact of unpredictable OS behavior owing to unpredictable system-call latencies and preemption due to interrupts. It must deal with multiprocessor clock skew by recalibrating the clock when a thread switches CPUs. To handle timing variations on thread wakeup, threads are allowed to pre-spin before they are to issue events. Lastly, it uses a single low-priority spinning thread to keep track of time.

To evaluate Buttress, they compared its performance to two simple trace replay methods. The first uses existing OS syscalls like Select and Sleep to handle timing, and the second uses a dedicated thread to spin and keep track of time. Using a range of workload traces, they showed that Buttress issues the majority of requests within 100 microseconds of their intended issue time and handles bursts better than the simple schemes.

One questioner in the Q&A session wondered if Buttress verified data that was written to the disks using checksums. Mustafa answered that in this methodology the data that is actually stored is not important, just the I/O requests themselves, and that Buttress does not write any real data to the disks. Another asked whether there is value in extending Buttress to operate at the file system level rather than the block level. Another asked if using realtime extensions to Linux or HP-UX would help the simple replay methods that just use system calls. Mustafa said they tried that

with realtime HP-UX but that the results still weren't as good as with Buttress. Another asked about the intuition behind why such accuracy is necessary. Mustafa said there were two factors: naive approaches end up issuing I/Os out of order, and handling bursts correctly requires higher accuracy than one would think.

Designing for Disasters

Kimberley Keeton, Cipriano Santos, Dirk Beyer, Jeffery Chase, and John Wilkes, Hewlett-Packard Laboratories

Summarized by Steve Schlosser

Keeton presented a solver to automatically design basic dependability solutions for large storage systems based on a customer's workload, recovery requirements, and the penalties of outages in terms of dollars lost. Using these parameters, and a few assumptions about the types of recovery mechanisms that are available, the system formulates a mixed integer program and uses an existing solver, CPLEX, to find a solution.

The system uses simple models of the available hardware (i.e., hosts, storage area networks, and disk arrays) and of the available data protection mechanisms (i.e., remote mirroring, tape backup). Remote mirrors can use synchronous or asynchronous updates, with or without batching. Tape backup solutions can be kept locally or off-site. A system can use the secondary copy either for failover or to reconstruct the primary copy. Lastly, the secondary copy can be kept hot or unconfigured, and can be dedicated or shared.

Given all of these alternative configurations and the customer's requirements, the goal of the system is to find the best configuration to meet the customer's needs.

The key is to express the user's dependability requirements in terms of monetary penalties (dollars per hour). First is the data-outage penalty rate, which is the penalty until the system comes back up. Second is the data-loss penalty rate, which is how much recently written data is lost when the system goes down. Given these penalties and the customer's workload requirements, the system can find the optimal solution.

They evaluated the system using the Cello 2002 file system trace from HP Labs, looking at the average bandwidth multiplier for bursts of activity and at the batch update rate. They used a number of different scenarios ranging from a university IT department backing up student directories to a large financial institution with regulatory requirements for backup and availability. These scenarios ranged from cases in which data loss was not so important to those for which data loss would be a business catastrophe. The examples illustrated the choices of dependability solutions that were appropriate for those scenarios.

During Q&A, a questioner wanted to know if the models took into account the solution's performance requirements as well as dependability requirements. Keeton said that they only consider update rates as an element of the solver, but that other issues of performance are an area of future work. She also mentioned that outlay costs are annualized and depreciated over three years with one failure per year. Another questioner mentioned that financial penalty is not linear and should be asymptotic. Keeton agreed that this is an important extension to this work. The last questioner asked if this method can be used to extend existing systems. Keeton said, yes, the system can do this.

KEYNOTE ADDRESS

Scaling File Service Up and Out

Garth Gibson, Panasas, Inc., and Carnegie Mellon University

Summarized by Dean Hildebrand

Garth Gibson discussed the need for improvements in file access and several current methods for doing so. He believes that quality expectation drives technology, with the highest level of quality being demanded by the Tri-labs and the oil and gas industry, requiring throughput of 1GB/s and 1.2GB/s, respectively. For technology to improve and breach the chasm, the need for such performance must be demanded by the masses. This includes the need for both file and file aggregate bandwidth improvements.

Current improvements started with the evolution from monolithic supercomputers to Linux clusters, giving a cost improvement from $100 million/Tflop to $1 million/Tflop. Garth then discussed several phases of file access improvements that have occurred since this development:

- Scaling up the NAS File Server
  To eliminate the single file-server bottleneck, the file system is partitioned among several file servers. Limitations: Cannot increase the aggregate throughput of a single file/directory.
- Partitioning is a manual process and quickly becomes non-optimal. Backup consistency issues.
- Scaling out phase 1: client data cache. Exploits client data cache, i.e., NFSv4 delegations. Limitations: Workloads can exceed client cache size.
- Scaling out phase 2: forwarding servers. Binds many file servers into a single system image. Forwards requests from accessed file server to the data's home file server. File system still partitioned, with each file

server providing global data access. Limitations: Data not directly connected to accessed file server must travel through two file servers.

- Scaling out phase 3: any server will do. All file servers have access to all storage (not indirectly through another file server). Limitations: Throughput for a single file is not increased.
- Scaling out pPhase 4a: asymmetric out-of-band. Clients obtain file layout map from a metadata server and access storage devices directly. Limited by network instead of CPU. Examples: SanFS, EMC High Road, SGI CXFS, Panasas, etc.
- Scaling out phase 4b: symmetric out-of-band. Client cluster is file server. Examples: RedHat GFS, IBM GPFS, Sun QFS, etc.

The description of file service scaling improvements concluded with a statement that Phase 4 is the most advanced architecture currently available. Panasas has combined the architecture of Phase 4a with object-based storage, an evolutionary improvement to standard SCSI. Each object is a container of related data, offloading most data path work from server to an intelligent device. Garth explained that the features of object storage he likes the most are metadata encapsulation, extensible attributes, security at device through digital signatures, and a smaller file layout map allowing more caching while providing performance and scalability for a variety of workloads.

Garth concluded with a statement that the most important aspect in improving throughput is utilizing out-of-band file access. Therefore, a pNFS (Parallel NFS) initiative has started to provide a single standard framework for data access, whether it is based upon blocks, objects, or files. This effort will continue to reduce file access cost by sharing client support among all vendors, enabling the file service to scale up and out for use by the general population.

GRABBAG

Diamond: A Storage Architecture for Early Discard in Interactive Search

Larry Huston, Rajiv Wickremesinghe, Intel Research Pittsburgh; Rahul Sukthankar, M. Satyanarayanan, Gregory R. Ganger, and Anastassia Ailamaki, Carnegie Mellon University; Erik Riedel, Seagate Research

Summarized by Aydan Yumerefendi

Larry Huston presented Diamond, an attempt to provide efficient searches over large volumes of data. The key insight behind the system is to move some of the search functionality to the storage servers and to discard irrelevant data as soon as possible. By distributing the search over a number of servers, Diamond speeds up query execution. It also limits the network traffic by sending only relevant data.

Diamond uses Active Storage to execute client-specified queries wrapped in "searchlets." Each searchlet consists of configuration code and a number of predicate filters, which are applied in sequence and determine the useful data. The system also dynamically manages the load among the storage servers and the client machine by using two different algorithms: CPU partitioning delegates work based on the CPU power of each machine, and queue backpressure considers each system stage as a queue and maintains a certain load on each stage.

The implementation of Diamond runs on RedHat Linux 9. Tests of the system use SnapFind, a custom application for performing searches in digital images. The evaluation results show that Diamond performs well and correctly balances the load among all participants. An audience member pointed out that searchlets do not maintain state and, as a result, they limit the system expressiveness. Larry answered the question by saying that lack of state allows for better load distribution, yet gives the system enough expressive power.

More information about this project can be found at *http://info.pittsburgh.intel-research.net/project/diamond/*.

MEMS-Based Storage Devices and Standard Disk Interfaces: A Square Peg in a Round Hole?

Steven W. Schlosser and Gregory R. Ganger, Carnegie Mellon University

Summarized by Andrew Klosterman

Steven Schlosser introduced a novel storage device, the MEMStore, a magnetic storage device that uses X- and Y-axis motion to access desired data. He explained how the settling time of the seeks is dependent on the maximum of settle time in both the X- and Y-axis, and that the X-axis settling time dominates.

In evaluating this new storage technology, Mr. Schlosser compared the interface of current storage technologies (logical block addressing) with various alternatives for the MEMStore that might exploit its unique characteristics. He examined roles (uses of MEMStores in systems) and policies (ways to change systems to specifically use MEMStores). Two tests were used to evaluate the fitness of MEMStores for a given role or policy: a specificity test and a merit test. The specificity test evaluated the fitness of a MEMStore for a particular role or policy. The merit test evaluated whether a change in the abstraction for use of the storage device was warranted.

After presenting some results of tests from his paper, Mr. Schlosser concluded by mentioning that MEMStores can be used as fast disks in systems with programmers using the familiar LBN addressing scheme.

During the Q&A session, Dan Ellard of Harvard asked whether or not the

"busiest disk" from the EMC trace saw the most I/Os or had the worst seeks, because (Dan claimed) sometimes lots of small I/Os tend to be sequential. Mr. Schlosser did not have an answer to the question, but posited that by replacing that disk with a MEMStore, the trace replay improved, thus validating the use of MEMStores in that role. Another question was raised as to the actual performance of the MEMStore research prototypes, but Mr. Schlosser had to inform the audience that no instances of the device he modeled yet exist as fully implemented storage systems.

Additional information may be obtained from *http://www.pdl.cmu.edu/MEMS*.

A Performance Comparison of NFS and iSCSI for IP-Networked Storage

Peter Radkov, Prashant Shenoy, University of Massachusetts; Li Yin, University of California, Berkeley; Pawan Goyal and Prasenjit Sarkar, IBM Almaden Research Center

Summarized by Wenguang Wang and Lan Xue

This paper turned out to be the most controversial paper of the conference. Prasenjit Sarkar presented a performance comparison of two IP-networked storage protocols that allow remote data access. The basic idea is to measure how performance differs when clients access remote data via file system versus IP-based storage area networking (SAN). NFS and iSCSI were used as specific instantiations of file- and block-level access protocols in this experiment.

Using combinations of micro- and macro-benchmarks, this paper provides a thorough comparison between NFS versions 2, 3, and 4 and iSCSI. Individual file and directory operations and overall application performance are measured, with both data-intensive workloads and metadata-intensive workloads, under various scenarios. Also, a decent amount of work has been done to identify the bottleneck of NFS protocol by capturing and analyzing the network message overhead.

The experiment results show that for a single-client environment, NFS and iSCSI have under comparable performance data-intensive workloads, while iSCSI outperforms NFS by a factor of two for metadata-intensive workloads. This work also identifies lack of metadata caching and update aggregation as the two major reasons for NFS degradation.

However, the fairness of the performance comparison between NFS and iSCSI was questioned. The major concerns and comments raised by the audience can be summarized as follows: It's not an apple-to-apple comparison because block-level protocol accesses data directly from storage, which is straightforward and bypasses file system layers overhead, while NFS provides more complex functionality which incurs extra overhead as a consequence. In addition, NFS is being improved and optimized, and the performance of the NFS prototype can differ greatly from an optimized implementation of NFS. Thus, it's still a question whether the results and conclusions of this experiment can be generalized to every NFS implementation.

Optimizing Block Access

A Versatile and User-Oriented Versioning File System

Kiran-Kumar Muniswamy-Reddy, Charles P. Wright, Andrew Himmer, and Erez Zadok, Stony Brook University

Summarized by Michael Abd-el-Malek

Kiran-Kumar Muniswamy-Reddy described Versionfs, a versioning file system that differs from earlier versioning file systems such as Elephant and CVFS in that existing applications do not have to be modified in order to access previous versions. Additionally, multiple versioning policies provide flexibility, and current-version access is optimized.

Whole-file versioning is used, as opposed to block-level versioning, as that is viewed as more user-friendly. Versions are created on close, mmap, or metadata operations (e.g., chmod). Two types of per-file versioning policies are provided: retention (how many versions to keep – number, total space) and storage (how to store versions – full, compress, or sparse). Copy-on-change is used, which is different from copy-on-write because the new data is compared to the old data and a version is created only if they are different. Existing (unmodified) applications can work with versioned files using a preloaded library that overrides the various file system syscalls.

Using the Am-utils benchmarks, the authors show little (1–4%) time overhead when using their versioning file system. Approximately 20% more storage is required for the old versions. However, the Postmark benchmark runs two times slower.

Q: Wenguang Wang. Each directory has more files than before. What's the performance penalty?
A: In a directory with many small files (such as Postmark), we found a difference in chopping down the directory in system time. It depends on the lower-level file system, but, yes, we did find a difference.
Q: Ed Gould. Hard links were not versioned, why? If I have two symbolic links to a file, will operations on each of these symbolic links create a different version?
A: Yes.
Q: Daniel Ellard. You used open/close to create new versions. What about NFSv3 that does not have open/close? Alternatively, if you have more than one thread, how do you create different files?
A: Haven't done this yet; it's a direction for future work.

Q: When two processes with different files are open at the same time, how many versions are created?
A: One on every close; we keep a counter every time a file is opened.
Q: Brent Calaghan. Insertion/deletion of data into the middle of files, how does that work with sparse data mode, where all following blocks are "pushed down" and sparse mode won't be as useful?
A: Yes, that is an issue. There is no way for us to tell if data is pushed down.

Q: Val Henson. Did you have LD_PRELOAD set to anything else before you started this project? Because this is not transparent to the user, since they have to set LD_PRELOAD.
A: Yes that's true, the user has to do this.

Tracefs: A File System to Trace Them All

Akshat Aranya, Charles P. Wright, and Erez Zadok, Stony Brook University

Summarized by Deepa Tuteja

Akshat Aranya described the details of Tracefs, a stackable and portable file system developed for capturing file system traces. The motivation of this work is to evaluate file system performance and help in its development. Security applications, such as monitoring activities, can also be built based on this work. The background study or related work quoted is in the BSD, Sprite, and Roselli's FS tracing studies. The work is based on the design goals of flexibility, performance, convenience, security, privacy, and portability.

Architecture of the system is such that the Tracefs (consisting of a number of tracers) sits in between the VFS layer and the file system to be traced. The main components of the tracer are input filters, assembly drivers, output filters, and output drivers. The input filters determine what to trace based on the user input. The job of the assembly drivers is to convert the traced operations and its parameters into a traced stream format. The output filters perform a series of transformations like encryption/compression. The output is written to a stable storage by the output drivers. The stable storage can be a file or a socket and can be specified by the user. The traces generated are in binary format, which helps in saving space and in parsing.

Anonymization is required to deal with the concerns of security and privacy. But at the same time, some correlation is required for the traces. Here symmetric key encryption methodology is used for anonymization.

Evaluation of the system was done on different sets of input and output filters. The configuration of input filters used were:

- full – to trace all file system operations
- medium – tracing only 40–50% of the operations
- light – tracing approx. 10% of the operations

The various output filters used were none, compression, checksum, encryption, and all.

The Am-utils and the Postmark benchmarks, respectively CPU-intensive and I/O-intensive, were used to test the system. The system did not show much variation for the elapsed time when using different output filters on the Am-utils benchmark. For the Postmark benchmark, there is an overhead over ext3 for elapsed time and the system time.

Using compression filters reduces the size of the traces. The compression ratio achieved is in the range of 8–21. Postmark generates traces 2.5 times faster than Am-utils, which explains the difference in overheads. Also, the file size increases because of the encryption padding required for anonymization.

To conclude, Tracefs gives a systematic approach to tracing and is a general solution that can be applied to various situations. It can be configured and is extensible in the sense that any output or assembly filters can be used with it. It is mainly helpful in file system development and security applications.

Additional information is available at *http://www.fsl.cs.sunysb.edu*.

HyLog: A High-Performance Approach to Managing Disk Layout

Wenguang Wang, Yanping Zhao, and Rick Bunt, University of Saskatchewan

Summarized by Shafeeq Sinnamohideen

The objective of this work is to improve disk I/O performance for servers with multiple concurrent users. Most reads are absorbed by in-memory buffer caches, so writes dominate the disk workload. Write performance is determined by disk performance (seek time and bandwidth), strategy (overwrite or log-structuring), and scheduling. Overwrite overwrites a block's previous contents with its new contents, possibly requiring a seek from the last write. LFS (Log-structured File System) aggregates small writes together and writes them in one log segment, making use of the disk's sequential write bandwidth. LFS, however, was generally believed to perform poorly for random update workloads because of the overhead of cleaning partially used segments (those with data that has been superseded by more recent writes). The authors observe that advances in disk technology, particularly the increasing ratio of sequential bandwidth to positioning time, lead LFS to outperform Overwrite on modern disks, except when disk space utilization is high.

To overcome this, the authors propose HyLog, a hybrid scheme that offers the benefits of both LFS and Overwrite. They observe that most writes go to a

small number of hot pages, while most of the overhead in LFS comes from cleaning cold pages. Thus, HyLog separates the disk into an LFS portion for hot data and an overwrite portion for cold data and directs writes to the appropriate section based on the observed write frequency. On standard benchmarks, such as TPC-C, HyLog picks nearly the optimal hot page percentage, and thus performs similarly to the best of both LFS (at low utilization) and Overwrite (at high utilization)

Bill Morris from Sun asked why a random write, sequential read workload was not evaluated. The response was that it is not common, but represents a worst case.

OPTIMIZING BLOCK ACCESS

ATROPOS: A DISK ARRAY VOLUME MANAGER FOR ORCHESTRATED USE OF DISKS

Jiri Schindler, Steven W. Schlosser, Minglong Shao, Anastassia Ailamaki, and Gregory R. Ganger, Carnegie Mellon University

Summarized by Andrew Klosterman

Jiri Schindler presented a means for extending the logical volume abstraction of disk arrays to accommodate efficient access to two-dimensional data structures. Through an example, he showed how traditional data layouts lead to efficient (streaming sequential) access in only one dimension. With a modification to the data layout, using what he called "quadrangles," more efficient access can be obtained. With quadrangles, column-major access can be performed on entire disk tracks at one time and row-major access is semi-sequential by exploiting disk head, or track, switch times as the disk spins (eliminating rotational latency).

Graphical examples of the quadrangle data layout scheme were presented, along with a graph showing the increased efficiency for accessing two-dimensional data with quadrangles. Experiments showed the performance gains resulting from the more efficient data layout on TPC trace replays. Mr. Schindler concluded by pointing out that by exploiting the physical characteristics of disks and maintaining the LBN programming model, efficient access to two-dimensional data structures was possible.

During the Q&A session, Wenguang Wang of the University of Saskatchewan asked whether switching disk heads instead of disk tracks would improve accesses with quadrangles. Mr. Schindler's response was that the use of head switches or track switches depended on the manner in which the disk sequenced its logical block numbers. Val Henson of Sun Microsystems, asked how the disk and array parameters could be extracted such that quadrangles could be exploited. Mr. Schindler indicated that experimental extraction (e.g., at format time) suffices to obtain the necessary information.

Additional information may be obtained from *http://www.pdl.cmu.edu*, the Parallel Data Laboratory at Carnegie Mellon University.

C-MINER: MINING BLOCK CORRELATIONS IN STORAGE SYSTEMS

Zhenmin Li, Zhifeng Chen, Sudarshan M. Srinivasan, and Yuanyuan Zhou, University of Illinois at Urbana-Champaign

Summarized by Deepa Tuteja

Zhenmin Li described the use of block correlations for improving the effectiveness of storage systems. There have been previous approaches for exploiting file correlations, but these did not scale well enough to be used for block correlations.

There are three possible approaches to obtaining block correlations: the white box approach, used by NASD, the gray box approach, used by SDS, and the black box approach, used by probability graphs and C-Miner. The probability-graph approach works well for finding file correlations, but cannot be effectively used for block correlations due to a scalability problem and multi-block correlation problem. C-Miner gives a practical black box approach that uses data mining techniques. It is based on the "frequent sequence mining" algorithm called CloSpan. The main observation is that the correlated blocks are accessed together in a short period of time. The frequency sub-sequences are a good indication of block correlations.

For an evaluation of the system, a comparison was done among the following approaches: no prefetching, sequence prefetching, correlation-directed prefetching (CDP), and CDP with disk layout. CDP decreases the miss ratio by 24% and reduces average response time by 25%. Another test done to determine the stability of block correlations shows that they are very stable and effective for a long time. There are, however, reasonable time and space overheads associated with it due to data mining.

C-Miner can improve the performance of a storage system and involves reasonable overheads. It can also be used to solve other problems such as network traffic monitoring and intrusion detection.

Additional information is available at *http://carmen.cs.uiuc.edu*.

WORK-IN-PROGRESS REPORTS

Summarized by Andrew Klosterman

SELF-*

Andrew Klosterman, Carnegie Mellon University

The Self-* project is building a brick-based distributed-object store from the ground up with management in mind. Current figures place storage administrative load at about one administrator per 5–10 TB. This figure makes manage-

ment a large part of storage total cost of ownership.

The system is built around a human organizational analogy, with individual storage bricks being self-optimizing "workers," while a hierarchy of "supervisors" decide how to spread data and work across the worker nodes. In the process of building the system, the team plans to get real management experience in order to more effectively target the problem. More information: *http://www.pdl.cmu.edu/SelfStar*.

Spensa: An Adaptive Distributed File System

Douglas Santry, University of Cambridge

The Spensa system is a brick-based cluster system in which all bricks are peers. Each brick in the system will run the Xen hyper-visor, to allow bricks to both store data and do processing. The cluster will thus be able to run databases, genetic searches, Web servers, etc., in virtual machines inside the cluster.

The Xen hyper-visor allows live migration of virtual machines, which permits a variety of optimization decisions. For example, migration can be used to load-balance or to bring data close to the corresponding computation. In addition, they envision that the system will be able to snapshot virtual machines, allowing the system to restart a previous snapshot if a virtual machine crashes.

MRAMFS: A File System for Non-Volatile RAM Using Inode Compression

Nate Edel, University of California, Santa Cruz

Magnetic RAM (or MRAM) is a new type of non-volatile RAM. The project studies ways to include MRAM in file systems. Because MRAM is very expensive compared to disk space, they use compression to conserve space usage. The system puts small files in MRAM in order to improve performance. They compress inode data using Gamma compression and file data on a block-by-block basis. Gamma compression takes a 128-byte inode down to around 15–20 bytes.

They have a prototype implementation that only compresses inode data, but will eventually implement data compression as well. Evaluation shows that the current prototype performs comparably to ext2 running in a RAM disk. This work is the starting research for another project called the Linking File System.

AVFS: An On-Access Anti-Virus File System

Charles Wright, Stony Brook University

AVFS is a virus-scanning file system, which uses a stackable file system approach to layer virus checking onto an existing file system. Most current virus scanners operate on points such as close or exec. AVFS scans for virus signatures on every read and write.

The system operates in two modes. In immediate mode, when a virus is written the block is discarded; when it is read the file is quarantined. However, this allows a virus to evade detection by writing itself in several different sessions. To address this problem the system also has a forensic mode, in which it versions files on the first writes, and does not return an error until a virus signature is detected in the file. The system uses a variant of the ClamAV OSS scanner, which uses automation for pattern finding. However, ClamAV is slow in the number of patterns that are searched. Thus, the AVFS team modified ClamAV into Oyster, which is faster than the original scanner.

Limiting Liability in a Federally Compliant File System

Zachary Peterson, Johns Hopkins University

Congress has enacted a variety of data maintenance acts and regulations to enhance corporate accountability. Most of these acts require that storage keep an audit trail by keeping versions of data. However, companies want to limit their liability within compliance to these acts.

This work focuses on how to securely delete data in a file system that keeps such an audit trail. Current secure deletion technology either requires multiple overwrites to remove magnetic signatures, or keeping data encrypted and securely deleting the key. Both of these techniques are difficult to perform in a versioning file system.

This work proposes keeping data encrypted using a small (128-bit) key kept in the file inode. This allows the user to securely delete the file by securely deleting this key. The current progress in the project is a modified ext3 file system that performs versioning (but does not implement this secure deletion technology). The versioning ext3 file system can be found at *http://www.ext3cow.com*.

Modeling Stateful Network File System Protocols Using Hidden Markov Models

R.J. Honicky, UC Berkeley and Network Appliance

One difficulty in building synthetic file system workloads is that operation orders is very important. For example, read and write operations to a file must be performed after an open to that file.

This work proposes the use of hidden Markov models (HMMs) to model file system workloads. HMMs have been used extensively in fields such as speech recognition. The group uses a vanilla HMM learning algorithm to generate an HMM for a trace, along with some methods to try to avoid local minima.

They have performed some initial work at validating the results. The operation counts, and the operation pairing counts, are similar to the original trace,

and hand inspection seems to show the traces to be similar.

FileBench

Patrick McDougall, Sun Microsystems

The team asserts that file system benchmarks are used mainly in two ways. They are used by vendors to classify and characterize products, and they are used by designers to set design goals. For both of these tasks the benchmarks must represent applications and not simply I/O.

For example, in cases the group has encountered, a database ran much more slowly than corresponding benchmarks would have suggested. They determined that it was due to slow performance in a logging thread, which gated the performance of the whole system. None of the existing benchmarks revealed this critical dependency.

They are working on a new file system benchmark that incorporates a variety of improvements. For example, they hope to make it modular, to include file system aging, to make it scalable by throughput, users, data set, and clients.

There are a variety of techniques for benchmarks, each with a set of drawbacks. Micro-benchmarks have limited coverage, trace replay can lose dependencies, and model-based approaches can lose important details. They plan to make a model of I/O generation and dependencies for individual threads, to attempt to avoid these problems.

File Attribute-Based Predictions and Optimizations

Daniel Ellard, Harvard

This group's studies have found that there is a strong association between create time attributes of a file and the operations performed on the file during its lifetime. They have implemented a system that builds a model of these associations. It is small and can be put into the kernel.

They are currently exploring ways in which these associations can be used to improve file layout, caching, and replication policies. They also speculate that this information could be used to make global tuning decisions and identify common idioms (such as lock files).

More information on the work can be found at *http://www.eecs.harvard.edu/sos* or *http://www.pdl.cmu.edu*.

TBBT: A Trace-Based File System Benchmarking Tool

Ningning Zhu, Stony Brook University

This work focuses on building an NFS trace playback tool. It is very hard to capture the important attributes of a real workload in a model, which makes trace replay important in a variety of situations. A variety of traces are available to the research community, and the number will continue to increase. This makes a replayer an important tool.

The playback tool involves a variety of challenges. For example, creating the initial file system image is difficult, as is file system aging, concurrency, error handling, and disk and CPU perturbation by the tool itself.

The replayer should be available to the community in two to three months.

Reliable Parallel File Transfer

Lu Zhao, Bowling Green State University

A variety of methods exist to do file transfer, such as FTP, HTTP, bitTorrrent, grid FTP, and parallel FTP. This group hopes to build a protocol that captures both reliability and parallelism.

In the proposed protocol, clients request file info from a main server. This main server knows the location of other storage nodes containing pieces of the data. This node tells these other nodes to send data to the client. The data is protected with checksums; if a checksum fails, the client can re-request data from another server.

Can Replicas Converge Across Network Partitions

Brent Byuonghoon Kang, University of California, Berkeley

Many systems use optimistic replication, where updates go to a single replica and are lazily propagated to other replicas. Previous work uses a version vector with an entry for each node; when combined, the vector takes the highest values of the previous vectors, plus an update for the combining itself.

However, with a network partition, the vectors can diverge. Instead of using a version vector, this work uses an approach called a summary hash history, which they claim allows for convergence on network partition.

Collaborative Buffer Caches in Data Centers

Zhifeng Chen, University of Illinois at Urbana-Champaign

Many data centers contain heterogeneous systems with low latency connections and a large amount of cache in a variety of locations. This creates a problem in that caches often will contain the same data. This group proposes content-aware caching, where each cache has some knowledge of the other caches in the system and can use this knowledge to improve its caching policies. This knowledge can be obtained through message passing or by prediction of cache behavior.

Deep Store

Lawrence You, University of California, Santa Cruz

One cost problem in storage systems is the maintenance of the growing volumes of archival data. This group proposes a new archival storage system to ease the cost and difficulty of maintaining archival information. For example, they propose the removal of redundancy

using inter- and intra-file compression. They also hope to manage content; data lives and dies with applications and systems. Performance is also a challenge, since many users require near-line access to archival data.

CACHING & SCHEDULING

CAR: Clock with Adaptive Replacement

Sorav Bansal, Stanford University; Dharmendra S. Modha, IBM Almaden Research Center

Summarized by Xun Luo

Dharmendra presented CAR, an enhanced CLOCK-caching algorithm, which keeps conformation with CLOCK on the primitives and outperforms CLOCK across a wide range of cache sizes and workloads.

The idea of CAR is to maintain two clocks; one of them contains pages of "recency" while the other contains pages of "frequency." New pages are first inserted in the former and graduate to the latter upon passing a certain test of long-term utility. The researchers did extensive trace-driven tests on the CAR algorithm and observed that CAR is comparable to ARC and substantially outperformed LRU and CLOCK.

Circus: Opportunistic Block Reordering for Scalable Content Servers

Stergios V. Anastasiadis, Rajiv G. Wickremesinghe, and Jeffrey S. Chase, Duke University

Summarized by Shafeeq Sinnamohideen

The goal of this work is to improve the scalability of Internet content servers that perform whole-file transfers, such as those serving images or non-streaming video and music. In these cases, the client needs to receive the entire file before it can make progress. In common with other Internet services, the majority of accesses are to a small number of common files. Many clients may simultaneously be fetching the same file although they may have started at different times.

The traditional approach is to serve each transfer individually and let each client progress at its own rate. If the requested file is too large for the server's cache, however, the multiple request streams will thrash in the cache. In the worst case, the access pattern to the disk degenerates toward random.

The proposed solution is to use the client's memory as a reorder buffer and transfer each block read from the disk to each client that needs it in the order it is read from the disk. Since scheduling the optimal sequence of transfers is NP-hard, the following heuristic is used: First, blocks are read to satisfy the most demanding (fastest, furthest ahead) client. Other clients are sequentially served blocks from the cache as fast as they can be transferred. If a client falls far behind, it is moved ahead to the most recent block, leaving a gap of missing blocks. When a client reaches the end of the file, it sequentially reads any missing blocks.

The system was evaluated with a synthetic workload of requests arriving in a Poission distribution, uniformly distributed across files with an average size of 512MB. The server and network were configured so that the bottleneck to sequential transfers would be the server's disk. Both identical and varied client network links rates were considered. With identical links, the network becomes the bottleneck, and with varied links, the disk is the bottleneck, but at much higher performance than with sequential transfers. Circus is insensitive to changes in file size, unlike sequential, but degenerates to sequential if file sharing is low. The conclusion is that content servers don't benefit from sharing if the files they are serving are large, and block reordering helps if there is sharing, up to a 10x speedup at the sweet spot.

Erez Zadok from Stony Brook University questioned how clients that perform multiple fetches or partial fetches from mirrors affect Circus. Mirroring reduces sharing, and thus the benefit, but parallel fetches do not. Erik Reidel from Seagate Research wanted to know how more sophisticated sharing models such as 90/10 or zipf would affect results. The response is that the results are applicable to the shared portion of the workload. Yitzhak Birk from Technion questioned whether the use of digital fountain techniques wouldn't be a better solution to the problem. While fundamentally different, it shifts the cost into client CPU usage and disk capacity, which may be cheaper than improving disk seek time.

A Framework for Building Unobtrusive Disk Maintenance Applications

Eno Thereska, Jiri Schindler, John Bucy, Brandon Salmon, Christopher R. Lumb, and Gregory R. Ganger, Carnegie Mellon University

Summarized by Michael Abd-el-Malek

This was one of the two Best Student Paper awards. The talk described a mechanism for running low-priority applications that perform disk maintenance (e.g., backup, cleaning, cache writeback, etc.). Storage systems can use idle time or "wasted" disk head rotation time in order to handle disk requests for the background applications, without impacting foreground applications.

Idle-time detection is conceptually simple, and so the presenter concentrated more on how to make use of "wasted" disk head rotation time. For example, if you have two foreground disk requests that are spaced apart in their disk layout, then you can process a disk request that occurs in the middle of the rotation for free. This constitutes the freeblock scheduling subsystem.

An API is provided that lets an application register its intent to read/write blocks in the system in the background.

An asynchronous calling model is used – an application callback is called when the freeblock subsystem is about to handle a disk request. This model is useful because it gives freedom to the application to lazily allocate memory.

Evaluation using TCP-C, Postmark, and a synthetic benchmark demonstrates that free disk bandwidth is available, even if no idle time is detected (e.g., in the case of Postmark). Foreground performance is not impacted.

More information is available at *http://www.pdl.cmu.edu/*.

MOBILE STORAGE

Integrating Portable and Distributed Storage

Niraj Tolia, Jan Harkes, and M. Satyanarayanan, Carnegie Mellon University; Michael Kozuch, Intel Research

Summarized by Akshat Aranya

Niraj Tolia presented "lookaside caching," a technique that harnesses the convenience and speed of portable storage devices, such as USB memory keychains, to enhance performance and availability of distributed file systems.

Niraj started his talk by asking whether portable storage devices can be used in more meaningful ways or if they are just glorified floppy disks. He showed that portable devices and distributed file systems have certain complementary properties. For instance, portable devices provide high performance and availability, whereas distributed file systems provide robustness, consistency, and high capacity.

The basic idea behind lookaside caching is to use a portable device as a fast cache; a file server is still the authoritative source for a file. Whenever a file is opened, its 20-byte SHA-1 hash on the portable device is compared with that stored on the server. If the hashes match, then the file is provided from the portable store; otherwise it is fetched from the server. The benchmark results show significant performance improvement for slow links even when a small number of files are served by the cache.

The work was questioned for contrived benchmarks: Why would anyone carry around the Linux kernel source code on a portable device for compilation? Also, a member of the audience suggested work on policies to decide what to put in the lookaside cache.

Segank: A Distributed Mobile Storage System

Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Junwen Lai, Yilei Shao, Chi Zhang, Elisha Ziskind, and Randolph Y. Wang, Princeton University; Arvind Krishnamurthy, Yale University

Summarized by Akshat Aranya

Sumeet described a system to manage data and mobile devices in a heterogeneous environment. The Segank system provides a uniform namespace and location independence of data without complete replication. The major emphasis of Segank is on awareness of network characteristics so as to minimize use of weak links.

Segank guarantees consistency by maintaining an invalidation log that maintains the timestamp for each update. The most recent invalidation log is maintained in a portable device that the writer carries. The log is propagated lazily to invalidate stale copies of the data. This approach decouples data propagation from log propagation.

Segank uses a multicast protocol, called Segankast, for location and access of replicated data. This provides network-aware reading by heuristically building a multicast tree. Data is shared using snapshots. This provides a trade-off between freshness and performance.

The presentation generated plenty of interest from the audience, who raised questions about concurrency and compared the system with alternative approaches like VNC. There were also some concerns about background propagation of data in insecure mobile environments, which, the speaker pointed out, were orthogonal to the problems addressed by the system.