FAST '05: 4th USENIX Conference on File and Storage Technologies

San Francisco, CA
December 13–16, 2005

KEYNOTE ADDRESS

Greetings from a File-System User

Jim Gray, Distinguished Engineer, Microsoft Bay Area Research Center
Summarized by Stefan Büttcher

Jim Gray's message was that we have arrived at an era of infinite storage. He argued that in today's storage systems, I/O bandwidth and seek latency are limiting factors. In order to keep a single CPU core busy, 100 hard drives are needed; for future 10-terabyte hard drives, it will take 1.3 days to read all data sequentially and five months to read them randomly.

File systems are becoming so large that we need database systems in order to be able to use them effectively: how do we find data in a file system containing 30 million files of 1GB each? Integrating a database into the file system and combining the hierarchical structure with a content-based addressing mechanism would help.

According to Jim, we are heading towards a backup-free world, because the file systems are getting so large that it would take too long to restore the contents. Since we have more storage space than we need, we might as well keep many versions of the data (snapshot file systems) instead of backups.

As a side blow in the direction of Garth Gibson, one of the inventors of RAID, Jim pointed out that RAID-5 is the wrong tradeoff, as it sacrifices bandwidth for more efficient space utilization. In the Q&A, Garth, of course, disagreed, noting that for many people storage space efficiency is still very important.

FILE SYSTEMS SEMANTICS

Summarized by Vijayan Prabhakaran

A Logic of File Systems

Muthian Sivathanu, Google Inc.; Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Somesh Jha, University of Wisconsin, Madison

Muthian began the talk by discussing how important it is to ensure the correctness of file systems. The current approaches, such as stress testing and manual exploration, are inadequate and error-prone. A logic of file systems is a formal framework for reasoning about file systems. It focuses on file system interaction with disk and targets file system design rather than implementation.

Then Muthian gave some background on file systems, describing metadata and data consistencies. The key challenge in reasoning is the asynchrony which arises in file systems due to buffering and delayed writes. There are three basic entities in the model: containers, pointers, and generations. A file system is a collection of containers linked through pointers. A container is a placeholder of data, and generation is defined as an instance of a container between reuse and free. Muthian then explained the concept of containers and generations through an example.

Concepts such as beliefs, actions, and ordering operators (e.g., before, after, and precedes) were explained. The proof system followed by the logic is based on event sequence substitution. Muthian gave examples of basic postulates, for example, "If container A points to B in memory, a write of A will result in the disk inheriting the belief."

Three case studies that are described in detail in the paper were briefly explained by Muthian. The first case study verifies the data integrity under various file system mechanisms, such as soft updates and journaling. The second case study examines a performance bug in ext3. The last case study looks at the non-rollback property under journaling. Other case studies detailed in the paper deal with generation pointers and semantic disks.

Providing Tunable Consistency for a Parallel File Store

Murali Vilayannur, Partho Nath, and Anand Sivasubramaniam, Pennsylvania State University

Parallel file systems distribute portions of a file across diff servers. With multiple data servers and client-side caches, consistency becomes an important issue. Current configurations provide either much weaker consistency (e.g., PVFS) or much stronger consistency (e.g., Lustre). However, the applications running on a parallel file system know better about their concurrency/consistency needs than does the file system.

The approach taken in CAPFS is to export the mechanisms and leave the policy to the applications, which provides tunable granularity. CAPFS uses content-addressable data stores and optimistic concurrency control mechanisms to provide serialization.

The architecture consists of two server components: hash servers, which are the metadata servers, and content addressable servers (CAS), which are the data servers. Hash servers provide a NFSv4-like interface. The CAS servers are multi-threaded servers. Murali then described how writes are handled. Whenever a write is issued it goes to the hash server first, which computes the hash, and then the write goes to the CAS. During commit, the old hash of the data is compared with the new hash. If they match, the commit succeeds. Write serialization is achieved this way. The system is verified with a 20-node experimental setup.

SENSOR STORAGE

Summarized by Shafeeq Sinnamohideens

- ***MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices***

  Demtrios Zeinalipour-Yazti, University of Cyprus; Song Lin, Vana Kalogeraki, Dimitrios Gunopulos, and Walid A. Najjar, University of California, Riverside

  Dimitrios Gunopulos first described a sensor network developed for, among other applications, a U.C. Riverside Conservation Biology project to monitor soil organisms. While the network has a large number of sensors, sensing several parameters over a long period of time, only a few time points or parameters are interesting. The sensor nodes are based on the RISE platform, have local flash RAM as their main storage, and limited network and power resources. Since nodes in their system only transmit data in response to queries, each node must store and index the data it collects in its local flash memory. The problem is that while existing storage and indexing structures are well suited to the properties of RAM and hard disks, flash memory has a few unique properties of its own: it can only be erased a whole block (several pages) at a time; a page can only be written into an already erased block; and a page physically wears out after being written 10,000–100,000 times.

  Using structures meant for other media will result in poor performance as a result of having to read and rewrite an entire block whenever any of its contents change, as well as wearing out some pages more rapidly than others. The goal of their proposed structure (MicroHash) is to efficiently support value-based and time-based queries for single data points or ranges while maximizing the lifetime of the flash memory.

  MicroHash contains data records that are both hashed into buckets and indexed. It uses four types of pages: data pages that store data records, index pages that store indices to the data, directory pages containing information on hash buckets, and a root page that stores properties for the structure. Pages are always written to flash in a circular order to provide wear-leveling. In normal operation, as the sensor generates data records, it inserts them into a data page. When the data page is full, it is written to flash in the next available position. The corresponding index record is updated and the index page written to the next position, if necessary. As writing proceeds, the oldest page will be overwritten when there are no more free pages. Because the index is always updated after data is written, an index page is never deleted until the data it indexed is also deleted. If a particular hash bucket contains too great a proportion of indexed records, a repartitioning step will split it into two less-full buckets.

  Searching by time is simple, since all pages are written in chronological order. Searching by value requires first hashing the value to select a dir page. The dir page will point to the most recent index page for that value. The index page will either point to a data page with the data or to another index page that can be followed.

  Jason Flinn asked how the number of directory buckets ever shrinks. The answer is that when splitting produces two new buckets, the old bucket is eventually reclaimed by the normal overwriting process.

- ***Adaptive Data Placement for Wide-Area Sensing Services***

  Suman Nath, Microsoft Research; Phillip B. Gibbons, Intel Research Pittsburgh; Srinivasan Seshan, Carnegie Mellon University

  These sensor nodes differ in scale from those in the previous talk; they are assumed to have more computing power and may be distributed anywhere in the Internet. Query-issuing clients may also be anywhere in the Internet. In addition to sensor nodes, the system may include other infrastructure nodes, which can also replicate data, perform data aggregation, and process queries. The system aims to assign functions to nodes automatically, in order to optimize efficiency, robustness, and performance across the entire system. Additionally, the IrisNet infrastructure may be supporting several different sensor networks, with different access patterns which may change over time.

  The IrisNet Data Placement (IDP) algorithm attempts to determine, for a given network hierarchy and node capabilities, the data placement that optimizes query latency, query traffic, and update traffic. It is a distributed algorithm that runs on each node, using only local knowledge to approximate the globally optimal solution, while rapidly responding to flash crowds. Each node builds a workload graph representing all data objects necessary for its queries, with edges weighted by the traffic across that edge. The algorithm must select fragments (subgraphs) to partition and allocate to each node. The optimal solution is $O(n^3)$, which is too slow to be used. By only considering subtrees, an approximate solution can be found in $O(n)$. By contrast, all better-performing algorithms require global knowledge, and no distributed algorithms perform as well.

  After partitioning, IDP must choose where to locate each fragment of the workload. It does this using two heuristics. One attempts to cluster data objects together. This reduces the number of nodes involved, but requires consideration of whether nearby machines can handle the extra load. The other places fragments as close to

the data source or sink as possible. This reduces traffic and latency, but may involve additional nodes. Repartitioning or replication is performed when load on a node exceeds a set threshold. When replicas are available, a query can select either a random replica or the nearest one. If the nearest is selected, it may become persistently overloaded, whereas selecting a random one will cause all replicas to have an equally light load. As a compromise, IDP selects a replica randomly, but with weighted distribution, so nearby replicas are selected more often.

Christopher Hooper asked whether energy consumption was considered and whether IrisNet could take advantage of heterogeneous power availability. The answer was that power had not been considered, but could be considered one element of a node's capacity.

FAULT HANDLING

Summarized by Kevin Greenan

- ***Ursa Minor: Versatile Cluster-based Storage***

  Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie, Carnegie Mellon University

  Awarded Best Paper!

  John Strunk presented the work on versatile cluster-based storage at CMU's Parallel Data Lab. To get the audience into the right state of mind, Strunk presented a quick brain-teaser which illustrated the fact that a single storage system generally stores different data sets, with different requirements. Unfortunately, all of the data in the system may share the same fault model and encoding scheme.

  Even though cluster-based systems provide more cost-effectiveness and scalability than today's monolithic approaches, these systems do not necessarily provide versatility. Ursa Minor attempts to solve the challenge of versatility in cluster-based storage.

  Essentially, Ursa Minor is a cluster-based storage system which supports multiple timing models, fault models, and encoding schemes among multiple data sets in a single system. In addition, changes to the distribution of data can be made online, thus configuration choices are adaptive. The architecture of Ursa Minor is quite simple and provides object access similar to NASD architecture and the emerging OSD standard. Basically, clients are required to consult an object manager for metadata requests and I/O request authorization. Versatility is accomplished using a protocol family that supports consistent access to data in the storage system. Each member in a protocol family is defined by three parameters: timing model, fault model, and encoding scheme. Online data distribution changes are made using back-pointers from the new data locations to the old data locations. The object manager can then revoke access to the old locations, forcing the client to request the new location of the data.

  In the end, we find there is a lot to gain from defining specialized configurations for different workloads in a cluster-based storage system, especially when the workloads are running at the same time.

  A great many questions came up during the Q&A. One member of the audience asked how an object is re-encoded upon distribution change. A distribution coordinator works its way through the object by actively re-encoding in the background, ensuring that newly written data does not get overwritten. A few of the questions were directly related to the encoding schemes used in Ursa Minor. Currently, the user is responsible for choosing which information dispersal encoding is used for a given data set. Lastly, Strunk was asked whether failures were assumed during migration, and he answered they are not.

- ***Zodiac: Efficient Impact Analysis for Storage Area Networks***

  Aameek Singh, Georgia Institute of Technology; Madhukar Korupolu and Kaladhar Voruganti, IBM Almaden Research Center

  Aameek Singh presented work on impact analysis, starting with a photograph of a woman pulling her hair out, which was strategically placed to symbolize the frustration involved in storage management. The work focuses on the change-analysis problem. The Zodiac framework is provided to help system administrators determine the impact of changes to a SAN (storage area network) before actually making the change. This framework integrates proactive change analysis with policy-based management; thus, the impact of an administrator's action is assessed with respect to a set of user-defined policies. In the context of impact analysis, policies can be thought of as best practices.

  Singh briefly explained the four primary components of Zodiac: SAN-state for incremental operation within a single analysis session; optimization structures for efficient policy evaluation; a process engine for impact evaluation; and a visualization engine, which acts as the output interface to the user. The main SAN data structure is represented by a graph of entities connected by network links; thus graph traversals are required when policies are added or evaluated. In order to make policy evaluation more efficient, the authors exploit policy classification, caching at every node in the SAN graph, and aggregation. All of these optimizations allow for a

reduced graph traversal space when evaluating policies.

Singh showed that the three policy evaluation optimizations significantly decrease the latency of policy evaluations and allow for more scalable evaluations as the size of the SAN increases. Overall, this work provides an efficient framework that provides what-if analysis under a policy-based infrastructure.

A member of the audience asked whether this framework could be used for root-cause analysis. Singh replied that this work did not focus on finding a root cause, but such a tool used in conjunction with their framework would be very helpful.

- ***Journal-Guided Resynchronization for Software RAID***

  Timothy E. Denehy, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

  Timothy Denehy presented an approach to software RAID resynchronization after a system crash, which was done with some of the other folks at the University of Wisconsin, Madison. Denehy pointed out that it is hard to maintain consistency at the RAID layer, since a very long window of vulnerability may exist during updates. Failures that occur within this window of vulnerability may leave a stripe within a RAID array in an inconsistent state. The authors show that this window of vulnerability can be removed through the use of a write-ahead log, which may result in poor performance.

  Instead of relying on a write-ahead log or offline scanners, the authors propose a solution that leverages the functionality of a client-journaling file system, such as ext3. Denehy gave a quick overview of ext3 with respect to transactions and journaling. A new mode of operation, declared mode, is added to the underlying file system. This new mode of operation, which is similar to ext3's ordered mode, requires the write record for each data block to reside in the journal before issuing the actual write. Unlike ordered mode, this results in a record of outstanding writes, which can be used to restore consistency upon a system crash.

  In addition to the new mode of operation, an interface is created, which allows the file system to communicate inconsistencies to the RAID layer. This interface between the file system and software RAID is very straightforward. The file system can tag a block with a synchronize flag, which results in a verify read request at the software RAID layer. At the RAID layer, the corresponding stripe is read and checked. If the parity is inconsistent, a new parity for the stripe is computed and written.

  The process of file system recovery and RAID resynchronization is done using the new functionality. After a system crash, the file system can scan the journal and communicate possible inconsistencies to the RAID layer. These possible inconsistencies are handled at the RAID layer using the verify read request.

  Denehy further justifies the effectiveness of declared mode by comparing it to ordered and data-journaling mode on a set of benchmarks, which shows that declared mode generally outperforms data-journaling mode and incurs very little overhead with respect to ordered mode. Another important side effect of this new form of RAID resynchronization is the reduction of the window of vulnerability from 254 to 0.21 seconds.

CACHING

Summarized by Ali R. Butt

- ***DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Localities***

  Song Jiang, Los Alamos National Laboratory; Xiaoning Ding, Feng Chen, Enhua Tan, and Xiaodong Zhang, Ohio State University

  Song explained that the motivation for their work is the increasing gap between disk and processor speed. Hard disks remain a performance bottleneck when accessing data at high speeds. He explained that the main reason for this bottleneck is the lack of sequential accesses to the disk, which, among other things, causes expensive disk-head movements. Although the application accesses are more sequential than random, the filtering effect of the buffer cache results in the accesses to disk becoming randomized. To address this issue, Song presented a new buffer cache-replacement algorithm, DULO, which uses both temporal and spatial patterns. The main goal of the paper is to increase the sequential accesses that are issued to disk.

  Song explained that there are two schemes that are employed to improve the number of sequential accesses: namely, disk request scheduling and file prefetching. The buffer cache sits on top of the file prefetcher and I/O scheduler. The cache filters the requests from the application to the lower layers, and has the potential problem of filtering the patterns, which in turn makes the pattern more random. Hence, the buffer cache has the ability to shape the disk requests. Since other schemes only consider temporal locality of the blocks being accessed, they may result in a smaller number of blocks being read from the disk, but these blocks may have poor sequential properties, resulting in more disk head movements.

In the Q&A someone asked whether the authors have compared DULO to LIRS (an algorithm also proposed by the same authors). Song responded that he has not yet done that but is considering extending DULO to a more general scheme that can accommodate any cache-replacement algorithm rather than only LRU. Kai Shen from Rochester University inquired how DULO compares to the optimal case in this situation. Song replied that it is hard to define "optimal" in this scenario, due to the complexity of the components involved.

- ***Second-Tier Cache Management Using Write Hints***

  Xuhui Li, Ashraf Aboulnaga, Kenneth Salem, University of Waterloo; Aamer Sachedina, IBM Toronto Lab; Shaobo Gao, University of Waterloo

  Examples of second-tier cache management, the topic of the paper presented by Kenneth Salem, are the file server acting as the lower cache, and the client as the top cache, or, more interestingly, the lower cache as the storage server and the top cache as a database system. An important thing to note in this context is that database systems that handle OLTP workloads perform a lot of write requests.

  There are two difficulties that are faced in two-tier cache management. One is that of cache inclusion, i.e., a page is stored in both the caches, which essentially wastes space. Therefore the challenge is to maintain exclusivity between the two caches. The second challenge is that the second-tier cache exhibits poor temporal locality. Kenneth pointed out that other people have looked at various schemes to manage two-tier caches by employing hierarchy-aware schemes, interpreting storage data, using explicit notifications between caches, and providing hints to the higher tiers. The approach presented in this paper is based on hint-based schemes that require only simple changes to the first-tier cache management. The main focus of the work is on write requests so as to improve the hit ratio of the second-tier cache.

  In the Q&A session, Song Jiang of LANL inquired about the effect of the first-tier cache management algorithm on the second tier, and he pointed out that the interaction may adversely affect the cache performance. Kenneth agreed that this was possible, which is why they are interested in evaluating the scheme for more applications. Prashant Pandey of IBM Research said that in the current scheme the second tier is expected to interpret the hints on its own and asked whether there will be any benefit in telling the storage exactly what to do. Kenneth replied that they have made an effort to keep the hints open for interpretation by the second tier, but it would be interesting to see if the second tier can simply use the hints as classification. However, this aspect remains part of their future work. Another questioner asked about the distinction between write hints and eviction hints. Kenneth replied that they currently only interpret the hints at the second tier, but possibly could introduce two additional bits in the hints to ask the second tier for direct eviction.

- ***WOW: Wise Ordering for Writes—Combining Spatial and Temporal Locality in Non-Volatile Caches***

  Binny S. Gill and Dharmendra S. Modha, IBM Almaden Research Center

  Binny presented an innovative idea that aims at improving the performance of writes to hard disks. He pointed out that the writes have often been ignored in caching research, which mainly focuses on improving performance of reads. The presentation started with a brief history of caching's important part in improving the I/O time of disks. But although read caches have significantly improved the performance of disks, there are six times more writes in terms of disk seeks. He also pointed out that write caches are typically 1/16th the size of read caches. Hence, improvement in write time can have a significant impact on the overall I/O performance, but the small size of write caches requires careful planning in order to get any benefit from them.

  Binny then presented WOW, which uses reordering of writes in the NVRAM write cache to reduce the disk seeks associated with writes. The order in which writes are destaged to disk is critical. WOW aims to use the smallest amount of disk time for writes and to use most of the time to service read requests. For this purpose, it utilizes both temporal and spatial locality of the writes. To create spatial locality, WOW uses reordering.

  The WOW algorithm is produced via an innovative marriage of the CSCAN and CLOCK algorithms, and has the good qualities of both. Basically, WOW uses CLOCK bits for temporal locality information, and weights of CSCAN to give the spatial order information. WOW keeps the sorted order of CSCAN and temporal bits of CLOCK to give both spatial and temporal locality information. The evaluation of the scheme shows that WOW indeed provides improved throughput and response time.

SECURITY

Summarized by Aameek Singh

- ***Secure Deletion for a Versioning File System***

  Zachary Peterson, Randal Burns, Joe Herring, Adam Stubblefield, and Aviel D. Rubin, The Johns Hopkins University

  Due to increasing federal regulations and other business requirements, versioning file systems are being deployed rapidly. These file

systems maintain multiple versions of the data and can be used to restore to an earlier version. For space efficiency, different versions can share data blocks. This paper makes two contributions: (1) secure deletion of a file, implying that a deleted file cannot be retrieved by any forensic techniques; (2) authenticated encryption, ensuring that data has not been corrupted between a disk write and its corresponding read.

Some of the earlier approaches—repeated overwriting, encrypting, and deleting the key—require more storage or need data blocks to be contiguous. This paper's approach minimizes the amount of secure overwriting and eliminates the need for contiguity. The main idea is to use a keyed transform to create a short stub representing the data blocks with the additional property that deleting the stub by secure overwriting automatically deletes the data.

The authos also presented techniques that are better optimized for deleting an entire version chain. The techniques have been implemented in ext3cow versioning file system.

- ***TOCTTOU Vulnerabilities in UNIX-Style File Systems: An Anatomical Study***

  Jinpeng Wei and Calton Pu, Georgia Institute of Technology

  Time-of-check-to-time-of-use (TOCTTOU) vulnerabilities occur in UNIX-style file systems when applications perform two non-atomic steps—first establish an invariant about the state of the file system and then perform an operation assuming the invariant to hold. For example, Sendmail first establishes that the mailbox is not a symbolic link (a malicious user's attempt to corrupt an important system file) and then writes to the mailbox. Between these two steps, a malicious user can modify the file system so that the invariant does not hold, but the application does not check for it in the second step.

  The paper attempts to define a formal model, called CUU, that can be used to identify such TOCTTOU vulnerabilities. For this, they identify pairs of operations that establish invariance and then operate on it: for example, <stat, open>. Such pairs, called TOCTTOU pairs, can then lead to potential attacks.

  The paper identified 224 such pairs in various utility programs such as Sendmail, vi, and RPM. They also checked the feasibility of attacks on vi, which shows that such attacks can have nearly a 50% success rate for large files.

- ***A Security Model for Full-Text File System Search in Multi-User Environments***

  Stefan Büttcher and Charles L.A. Clarke, University of Waterloo

  With increased interest in desktop search, there are many tools available now from companies such as Google, Microsoft, Apple, and Yahoo. However, a multi-user environment presents new and interesting challenges. Keeping a separate index for each user in the system is inefficient, since many files are actually accessed by multiple users and thus a single file system change would need to be pushed into each index.

  A second approach, that of keeping a single index and postprocessing search, requiring that files that a user should not see are removed, suffers from a subtle problem: since query ranking uses statistics that in a single index case would be systemwide, carefully formed queries can leak out potentially critical information.

  As a solution, the paper proposes GCL, a structured query language developed in the 1990s by one of the authors which evaluates on-the-fly query ranking using security primitives, ensuring that no file or its influence on statistics is revealed through the results. One of the shortcomings is the memory caching of security properties of each file, which is 32 bytes for each inode.

  The system shows good performance and is available at http://www.wumpus-search.org.

MULTI-FAULT TOLERANCE

Summarized by Florentina Popovici and Timothy Denehy

- ***Matrix Methods for Lost Data Reconstruction in Erasure Codes***

  James Lee Hafner, Veera Deenadhayalan and K.K. Rao, IBM Almaden Research Center; John A. Tomlin, Yahoo! Research

  Jim Hafner addressed two general problems pertaining to erasure codes, with the ultimate goal of recovering lost data whenever it is information-theoretically possible. First, can the system recover from uncorrelated errors and, if so, how? Second, how can the system efficiently recover partial strip data? To solve these problems, the author presented the following theorem: for any linear erasure code and a set of sector failures, there exists a simple mechanism that identifies which sectors cannot be recovered and provides formulas for the reconstruction of those sectors that can be recovered. Jim presented their method, based on matrix theory and pseudo-inverses, which completely solves the first problem and provides the formulas for solving the second problem.

  He also presented a hybrid approach which uses the matrix methods along with the code-specific recursive reconstruction methods to improve efficiency. Finally, he demonstrated their methodology for recovering lost array sectors with a TCL/Tk application.

  The first questioner asked if the ordering of sector recovery matters? Jim responded that if the sec-

tors are lost simultaneously, the ordering of recovery does not matter. Garth Gibson asked how often an additional sector can be lost and recovered under existing erasure codes. In his experience, Jim estimated that a third lost sector could be recovered about 50% of the time.

■ ***STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures***

Cheng Huang, Microsoft Research; Lihao Xu, Wayne State University

Cheng Huang asked how to ensure both reliability and performance for storage systems. Some of the characteristics of such systems are that they are built from less reliable components in order to achieve large capacity, and that they may also be geographically distributed.

Reliability is achieved by redundancy. Usually the codes used are (n,k) threshold codes. n is the number of nodes where the shares of the data are distributed, and k represents the minimum number of shares that need to be gathered to reconstitute the original data. Most systems use MDS schemes, which allow for the recovery of $r = n - k$ nodes, where r is called the reliability degree of an (n,k) scheme.

But all practical schemes use Reed Solomon schemes as MDS, and they are slow, so the question is whether there are other, better-performing schemes. The alternatives are MDS array codes such as XOR (rr = 1) and EVENODD (r = 2). There is a generalized EVENODD algorithm that recovers from three failures, but the authors wanted to reduce its decoding complexity further and so propose a new algorithm, called STAR.

Cheng exemplified the recovery schemes for the EVENODD and STAR algorithms and showed how the algorithms recover from failures. The extended EVENODD algorithm uses diagonal parities with slopes of one and two. STAR, however, uses diagonals with slopes of one and negative one. Cheng showed how this geometric symmetry used by STAR leads to faster decoding.

■ ***WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems***

James Lee Hafner, IBM Almaden Research Center

Jim Hafner started by discussing why there is a need for another erasure code. The focus is on distributed storage systems and distributed RAID with more vulnerable components, and there is a need for another performance metric.

The proposal is a vertical code, with properties of symmetry, balance, and localization. Symmetry allows for easy implementation and natural load balancing. Localization means that I/Os do not involve the entire stripe. There is also more sequentiality from longer I/Os. The array size can be varied with fixed parity in-degree (number of inputs). Furthermore, the data-out degree is constant and equal to the fault tolerance.

The focus of this work is on codes with 50% efficiency. One of the features is variability of fault tolerance. The fault tolerance level can be changed by adding or subtracting an element without remapping or readdressing existing blocks. The disadvantage is that there is only 50% efficiency.

Ed Gould asked Jim to estimate how much fault tolerance is needed for a level of reconstructability of 90%. Jim answered that it depends on the components, as different batches of components from manufacturers have different errors. Also, it depends on the configuration and the combination of independent versus dependent domains.

WORK-IN-PROGRESS REPORTS

Summarized by Matthew Wachs

■ ***Controlling File System Write Ordering***

Nathan Burnett, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

The order in which writes occur to a file system must be controlled, or it may not be possible to restore the file system to a consistent state after a crash. While operating systems use techniques such as write-ahead logging to manipulate file system structures safely, it is difficult for an application to do the same for data consistency within its own files, because the operating system does not expose primitives for write ordering to the application. Nathan Burnett described two conventional ways that applications can ensure that commits to stable storage occur in the desired order: direct I/O and fsync(). Direct I/O allows applications to write directly to the raw storage device, avoiding all caches; however, it is slow (because of synchronous writes) and not portable (it is not universally available and APIs are not consistent). fsync() is portable, but slow as well. The fast alternative, ignoring write ordering, cannot ensure recoverability after a crash. Burnett suggested that the OS export an interface allowing the application to describe ordering constraints for writes. Not all writes may need to be ordered; taking advantage of this might yield better performance.

He proposed two methods for expressing ordering constraints: a barrier() system call which (globally) prohibits reordering of writes across the call; and asynchronous graphs, which express the constraints using an implicit graph data structure. In conventional applications, calls to fsync() could easily be replaced by calls to barrier(). The graph approach requires more extensive modifications:

write() returns an identifier for each call, and future invocations of write() can be passed a list of identifiers corresponding to writes (if any) that must occur first.

The authors have simulated both techniques and shown that the graph approach reduces the number of writes and the number of non-sequential writes in a TPC-B-like workload over both synchronous writes and the barrier approach, because it allows the coalescing, in some cases, of hundreds of small writes into one large write. They are now implementing the techniques in FreeBSD 5.4.

- ***Rethink the Sync!***

  Edmund Nightingale, Kaushik Veeraraghavan, Peter Chen, and Jason Flinn, University of Michigan

  Another way of thinking about durability is at a much higher level: transactions must not become visible externally until they have been committed to stable storage, but the application issuing them may continue executing while the transactions are still in volatile memory. When writes ultimately occur, they are performed in the order issued, maintaining the proper write ordering. Jason Flinn presented this approach as "external synchrony" or "visible synchrony." The authors are implementing this type of durability in the Linux kernel using mechanisms from their Speculator project (which addresses speculative execution in a distributed file system). Synchronous operations are performed asynchronously, and each operation is a transaction in the ext3 file system with data journaling.

  Synchronous I/Os taint the calling process with an annotation prohibiting external output (such as to the screen or network) until all preceding I/Os are complete. If processes engage in IPC, taint annotations are inherited as appropriate. Because progress is being made on I/O in the background, the latency during which external output is withheld while pending commits finish is expected to be short enough not to be noticed by a human. Postmark results using visible synchrony show that performance is within 6% of an asynchronous implementation.

- ***Amino: Extending ACID Semantics to the File System***

  Charles Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok, Stony Brook University

  Applications such as mail servers and text editors often need to enforce transactional semantics on file manipulations: atomicity, consistency, isolation, and durability (known as ACID). While databases provide ACID semantics, there is no standardized interface to databases, which limits portability for applications that might use them. The availability of ACID at the file system would simplify error handling (transactions could simply be aborted), enhance security (time-of-check-to-time-of-use security vulnerabilities could be avoided by serializing concurrent accesses), and ensure durability.

  Gopalan Sivathanu presented the idea of providing support for arbitrary transactions in the file system as a first-class service. To make this possible, the operating system itself must also support transactions at layers such as the cache. The authors have created a prototype file system, Amino, that provides begin, commit, and abort calls alongside the standard POSIX interface. Legacy applications automatically have each system call wrapped in a transaction; enhanced applications can wrap begin and commit calls around arbitrary sequences of POSIX I/O calls and computational activity. Back-end storage and transactional primitives are provided by Berkeley DB. The prototype is implemented in user level through a ptrace monitor, allowing existing applications to run unmodified and avoiding fundamental modifications to the OS.

- ***PASS: Provenance-Aware Storage System***

  Margo Seltzer, David Holland, Kiran-Kumar Muniswamy-Reddy, Uri Braun, Jonathan Ledlie, Harvard University

  Provenance is metadata about the history of an object. For instance, if an application reads files A and B, then later writes file C, the provenance of file C includes files A and B, the application itself, and other environmental information that may have been used to derive C. Kiran-Kumar Muniswamy-Reddy explained that provenance is useful to scientists in understanding how results were arrived at, to homeland security applications in determining the information used to suggest a possible threat, and to business compliance systems in tweaking policies for information life-cycle management. He believes that the operating system and file system should be in charge of tracking provenance, because all data flows through them. Provenance should be a first-class entity which is automatically annotatable, indexable, and queryable; the authors are designing a storage system that meets these goals.

  Muniswamy-Reddy highlighted several research questions: first, how provenance should be stored so that it is indexable and queryable; second, what the proper security model for provenance should be (does access to a file imply access to its provenance?); and third, how it can be sent over "the wire." A prototypical implementation added only 2% overhead for a Linux kernel build. More information can be found at http://www.eecs.harvard.edu/syrah/pass.

- ***Logistical Storage***

  Surya Pathak, Alan Tackett, and Kevin McCord, Vanderbilt University

  Scientific computing, especially for efforts such as high energy

physics, often requires sharing large data sets among collaborators around the world (for instance, some projects generate 3TB per day, 1PB per year). Surya Pathak introduced L-Store (Logistical Storage), a framework to address this need using software agent technology and the Internet Backplane Protocol. The software agents provide automated resource discovery and fault tolerance. The scalability of the authors' distributed approach has allowed them to achieve 10Gb/sec sustained reads and writes to distributed storage using a RAID-5 encoding on moderate hardware.

- ***A Unifying Approach to the Exploitation of File Semantics in Distributed File Systems***

  Philipp Hahn and Carl von Ossietzky, University of Oldenburg

  Many distributed file systems exist, but few are widely used in practice. One reason for this may be the fact that they are often specialized for particular types of environments or applications. File systems that have seen widespread adoption because of their generality may suffer from "compromise" designs that optimize for average performance and excel at nothing. Philipp Hahn suggested that it would be ideal to have a universal abstraction for a distributed file system that allows for per-file optimizations and special cases and permits requirements to change over time. Various dimensions of configurability include concurrency, latency, availability, and consistency; the anticipated fault mode, access frequency, and access pattern; and the caching, versioning, encryption, and compression strategies employed.

  Benefits from his work might include being able to bypass locking for backups, to avoid strong consistency in disconnected operation, to suppress replicas for temporary files, and to use different replica placement strategies for different files. He seeks to achieve this flexibility by creating a framework for a distributed file system with pluggable modules that allows the user to control all of these options up to administrator-configured limits, and falls back to a default configuration when none is specified. Hahn anticipates that self-tuning may relieve some of the burden of configuration.

- ***A Centralized Failure Handler for File Systems***

  Vijayan Prabhakaran, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

  Commodity file systems have "broken" failure handling, because they assume that disks fail in a fail-stop manner. Moreover, failure handling is complex, because the code that actually performs I/O on behalf of applications is diffused throughout the system (for instance, journaling code or the sync daemon), and thus the code that must handle failures is distributed throughout the system. Vijayan Prabhakaran believes that this results in illogically inconsistent policies, because the reactions to the same error depend upon which component received it: some may propagate while others retry. It is also difficult to separate policies from mechanisms in this regime; and each component is subject to bugs.

  The authors' proposed solution is a centralized failure handler, which addresses each of these shortcomings; it also relieves programmers of the need to add error-handling code to each new function, because the global handler already takes care of errors. Prabhakaran pointed out three issues with this approach: semantic information about a particular I/O needs to be available at the handler so it can respond to errors appropriately; the handler has parts that must be specialized to a particular file system while other parts are generic across file systems; and I/O paths are time-critical, requiring the common completion path to be separated from the error case.

- ***Storage Benchmarking for HPC***

  Mike Mesnier, James Hendricks, Raja R. Sambasivan, Matthew Wachs, and Gregory Ganger, Carnegie Mellon University; Garth Gibson, Carnegie Mellon University and Panasas

  High-performance computing (HPC) applications are one important class of programs that use storage systems, but it is difficult to simulate their access patterns with existing benchmarks. In particular, the coordination and data dependencies between multiple compute nodes that are accessing storage may need to be modeled in a benchmark to capture the true nature of HPC workloads. Mike Mesnier discussed the idea of explicitly capturing this coordination in an existing workflow-specification language; specifications could then be used by a distributed workload simulator to synthetically generate multi-client accesses similar to those of a given HPC application.

  The modeling language might include data sources and sinks with flows between them passing through compute nodes that perform transformations on the data. At the same time, the language must also incorporate I/O characteristics such as read/write ratio, request size, and randomness for the simulator to follow. The authors plan to select and extend an appropriate workflow modeling environment and to begin a repository of specifications expressed in this language by providing reference specifications—for example, HPC codes—and then soliciting the contributions of domain experts from different fields such as computational chemistry, bioinformatics, and so on.

- ***POSIX I/O Extensions for HPC***

  Brent Welch, Panasas

  Just as it is important to benchmark HPC applications, so, too, is it fruitful to optimize for them at

each level of a storage system. The POSIX semantics for I/O, which are intended for single-node access, are not ideal for environments using collective I/O and clustered compute-node access to shared storage. Brent Welch discussed an initiative, being undertaken by a large working group, to draft proposed API enhancements to POSIX that may boost the performance of this class of applications. Many of the changes are based on the ideas of relaxing expensive semantics and providing hints to the storage system. Among the proposals are support for vector I/O, coherence (propagation or invalidation of data), lazy attributes in metadata, locking schemes, shared file descriptors, and layout hinting. For instance, a statlite() call extends stat() to poll only those attributes actually needed by the application; and ACLs match the new NFSv4 semantics rather than the old POSIX ACL semantics. More information can be found at http://www.pdl.cmu.edu/posix.

■ ***Storing Trees on Disk Drives***

Medha Bhadkamkar, Fernando Farfan, Vagelis Hristidis, Raju Rangaswami, Florida International University

Many modern applications store tree-structured data, such as those using XML, those storing directory hierarchies, and those implementing suffix-tree alignments for bioinformatics. Because of this, being able to store tree-structured data efficiently is an important factor affecting the performance of these applications. Currently used schemes (such as relational databases or flat files) do not take advantage of the tree structure or the performance characteristics of disk drives.

Raju Rangaswami proposed tree-structured placement, a way of matching the data structure of a tree to the semi-sequential access patterns of a disk drive. Under this technique, the root is placed at the outermost track, with its children residing on the next free track, placed such that accessing the first child results in a semi-sequential access (that is, one which incurs no rotational delay because it falls under the disk head just as the seek to that track completes). Subsequent children are placed just after the first one and incur only a slight rotational delay. The drawbacks of this approach are high space fragmentation and poor random access times. A second strategy, the optimized tree-structured placement strategy, places child nodes in non-free tracks and permits some limited rotational latency to reach the first child on that track, increasing the flexibility of placement; it also stores multiple nodes in a single disk block. In the future, the authors plan to explore how to store arbitrary graphs more efficiently on disks.

■ ***Efficient Disk Space Management for Virtual Machines***

Abhishek Gupta and Norman Hutchinson, University of British Columbia

Virtual machines are being used for various purposes, but the problem of efficiently providing storage for each virtual machine has not been entirely solved. Frequently, multiple VMs share the same disk image and software configuration; existing solutions such as LVM (the Linux Volume Manager) and Parallax share blocks between the images and provide copy-on-write to achieve good space utilization. Abhishek Gupta described weaknesses in these systems: LVM has a high cost when the VM running on the master image overwrites a block (the original copy of the block must then be propagated to all the mirrored images or else they will see the changed block, unless they have performed a copy-on-write to that block). LVM also does not support hierarchical copy-on-write images (recursive snapshots).

Parallax can do recursive snapshots, but it is unclear how efficient it is: the cost of traversing the radix-tree data structure to translate a block address may be high, and there is no space reclamation. Gupta discussed how to explore possible solutions to these limitations: first, the authors have implemented radix trees in LVM so that they can be benchmarked and the existing solutions can be quantified; next, they will either try to fix the problems in current approaches or propose a new data structure that will support faster snapshots.

■ ***Intelligent Data Placement in a Home Environment***

Brandon Salmon, Carnegie Mellon University

Consumer media devices are proliferating in the home, and they are increasingly capable of handling high-quality videos, music, and photos. At the same time, the devices have varying degrees of mobility, storage capacity, and access to power. Because of this, Brandon Salmon highlighted the fact that there is a data synchronization problem in getting desired data to the right device at the right time. Currently, data transfer is typically done manually, which is not a scalable solution. Pushing data to all devices is not feasible, because power or capacity may be at a premium or some mobile devices may be out of range; yet on-demand access is not sufficient, because it is often not reliable.

Salmon's plan is to match data to appropriate destination devices by using metadata (such as ID3 tags in music files), easily observed access patterns, and machine learning to anticipate upcoming requests. Unlike hoarding, he plans to use information about data and device access patterns to match data to a device, rather than using inter-file access patterns alone. He also plans to leverage known cliques (such as a cell phone usually being near a laptop, but rarely near a DVR) to optimize caching.

- ***Functionality Composition Across Layers in a Storage System***

  Florentina Popovici, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

  Various functions, such as caching, prefetching, layout, and scheduling, are implemented at each component in a layered system. For instance, a Web server may have a cache that duplicates data in the buffer cache, RAID controller cache, and hard disk cache, resulting in inefficient use of available memory. Florentina Popovici argued that exclusive caching would result in superior use of resources, and that analogous coordination of other functions such as prefetching and scheduling would similarly improve efficiency. She enumerated a number of research questions, such as where the best layer is to implement a particular algorithm (such as prefetching); how performance is influenced by a combination of decisions at different layers; and how quality of service is influenced by the hierarchy of layers.

- ***Transaction Support in the Windows NTFS File System***

  Surendra Verma, Microsoft

  Windows Vista's NTFS file system implementation is expected to include ACID semantics for transactions consisting of arbitrary file-system operations. Surendra Verma gave a product demo of TxF, the code name for the transactional support in Vista, showing how file-system manipulations wrapped in transactions being performed in two different command prompt windows were not visible to each other. For instance, if a directory is deleted in one window but the transaction has not yet been committed, then the directory is still visible from the other window. Conflicts between concurrent transactions result in errors and aborted transactions to preserve the semantics.

ON THE MEDIA

No summaries available

- ***On Multidimensional Data and Modern Disks***

  Steven W. Schlosser, Intel Research Pittsburgh; Jiri Schindler, EMC Corporation; Stratos Papadomanolakis, Minglong Shao, Anastassia Ailamaki, Christos Faloutsos, and Gregory R. Ganger, Carnegie Mellon University

  Awarded Best Paper!

- ***Database-Aware Semantically-Smart Storage***

  Muthian Sivathanu, Google Inc.; Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

- ***Managing Prefetch Memory for Data-Intensive Online Servers***

  Chuanpeng Li and Kai Shen, University of Rochester

ON THE WIRE

Summarized by Kristal Pollack

- ***A Scalable and High Performance Software iSCSI Implementation***

  Abhijeet Joglekar, Michael E. Kounavis, and Frank L. Berry, Intel Research and Development

  This work focused on iSCSI software solutions rather than common hardware implementations that use TCP/IP offload engines or iSCSI host bus adapters. An iSCSI software implementation offers the advantage that it can scale with CPU clock speed and the number of processors. Furthermore, it was shown in earlier work that hardware offload engines can become a bottleneck for small block sizes.

  For the majority of their work, the authors used a user-level sandbox implementation of the iSCSI protocol, coupled with an optimized TCP/IP implementation. They discovered that the two main bottlenecks in iSCSI processing are CRC generation and data copies.They therefore set out to make these two operations more efficient. They were able to improve CRC generation performance by a factor of 3 by replacing the standard CRC algorithm, developed over a decade ago by Sarwate, with a new algorithm, Slicing-by-8 (SB8), which takes advantage of more modern computer architectures. SB8 requires fewer operations per byte on the input stream and takes advantage of the size of the processor cache by using appropriately sized lookup tables. Data copy performance was improved by interleaving the CRC generation with the data-copy operations.

  iSCSI processing performance in the authors' sandbox environment showed a factor-of-two improvement by changing the CRC algorithm to SB8. They gained an additional 32% performance improvement when the interleaving data copy with CRC generation was added. With both optimizations they improved throughput from 175MB/sec to 445MB/sec. SB8 was then implemented in UNHs iSCSI implementation and improved the overall throughput by 15%. The authors attribute this lower gain to the significant overheads in the Linux 2.4 implementation of SCSI and TCP/IP.

  In the Q&A session someone asked why they chose 8 for their algorithm. It was from empirical results and may be processor-dependent. Another questioner asked how close they were to running at maximum memory—very close to maximum, almost memory limited, was the answer.

- ***TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization***

  Navendu Jain and Mike Dahlin, University of Texas, Austin; Renu Tewari, IBM Almaden Research Center

  TAPER is a solution for synchronizing data across distributed replicas. The authors' solution aims to minimize the bandwidth required for this task by using multiple phases of redundancy

elimination. The authors introduced a method for quick elimination of identical files by using content-based hierarchical hash trees. They also developed a method for similarity detection using bloom filters.

These new methods were combined with existing techniques to form their overall protocol. In phase 1 they eliminate all identical files using their content-based hierarchical hash tree technique. In phase 2 they eliminate all identical data chunks in the remaining files by using content-defined chunks similar to LBFS. In phase 3 they use their bloom filter technique to find a similar file at the target for each file that has not been completely matched at the source. The unmatched pieces of the files at the source are broken into fixed-sized blocks, and their signatures are sent to the target. At the target these signatures are used in a sliding-block technique over the chosen similar file to find identical blocks. Finally, in phase 4 they use their bloom filter technique again for similarity detection between the remaining unmatched chunks and the already matched data at the source. The unmatched chunks are delta-encoded against the matched data, and the delta encodings are sent to the target to complete the synchronization.

TAPER was compared with rsync for several software sources, object binaries, and Web data sets. Gzip compression was used before sending data over the wire. In terms of bandwidth reduction, TAPER saved 18–25% for software sources, 32–39% for object binaries, and 12–57% for Web data when compared with rsync.

In the Q&A session someone asked if TAPER was compared with any other products, such as Tivoli. The answer was that rsync was the most relevant comparison. Another questioner asked if most of the savings came from phase 2. The answer was that 60% of the total savings came from the first two phases.

- ***VXA: A Virtual Architecture for Durable Compressed Archives***

  Bryan Ford, MIT CSAIL

  Both general-purpose compression and multimedia encoding schemes have evolved rapidly over the past few decades. This presents a challenge for digital preservation of compressed data as encodings and the software to read them become obsolete. The author observes that instruction encodings are far more durable than data encodings. He points out that the x86 architecture has experienced few major changes over time, and has made efforts to be backwards-compatible. The author takes advantage of this observation by implementing Virtual eXecutable Archives (VXA), which save executable x86 decoders along with compressed data.

  The VXA architecture uses a specialized virtual machine to run the decoders in. Decoders have access to computational primitives, but can only read from a given stream and write the decoding back. The decoders are extremely isolated and cannot use any of the operating system services. An implementation of this architecture was built using the zip/unzip tools. When compressed files are input into the system they are attached with decoders to the encoding they are already in. If the file can be compressed further, a lossless compression technique is used that best matches the file type, and the file is tagged with the appropriate decoder. When files are read, the appropriate decoder is loaded into the virtual machine, then executed on the stream of encoded data to produce the decoded data. The decoders are stored in a compressed format using a standard compression algorithm to reduce their overhead as well.

  The performance for the VXA implementation was tested using six common decoders. The storage overhead for these ranged from 26KB to 130KB. The performance overhead on an x86-32 execution was 0–11%, while the performance overhead for the x86-64 execution was 8–31%. This can be attributed to the fact that the VXA decoders are 32-bit.

  In the Q&A session someone asked if the system assumptions were violated by gzipping the gzip compiler. The answer was that even though he demonstrated VXA with open source decoders, the goal was really to use this system for proprietary encodings that are more likely to disappear. If one was worried that gzip might go away, the gzip compiler could be left unencoded. The next questioner was concerned that this is only for the x86 instruction set and wondered why it and not a universal one, such as Raymond Lorie's, was chosen. The answer was that we won't forget x86; it's ubiquitous. Someone asked if extracting semantic content was addressed, and the answer was no. The last question was, how do you ensure that decoder code is trusted and how do you verify that the sandbox environment is safe? The answer was that you have to trust the library for the emulator.

TOOLS

Summarized by Abhishek Gupta

- ***I/O System Performance Debugging Using Model-Driven Anomaly Characterization***

  Kai Shen, Ming Zhong, and Chuanpeng Li, University of Rochester

  Performance problems in complex systems are hard to identify and debug, due to the presence of manifold system features and configuration settings coupled with dynamic workload behaviors and special cases. In a nutshell, the approach presented by Kai Shen is

to construct simple and comprehensive models of system components using their corresponding high-level design algorithms. Later, discrepancies between model prediction and actual system performance are used to discover performance anomalies. In order to quantify these, Kai introduced the notion of a parameter space, a multi-dimensional space in which each workload condition and system configuration parameter is represented by a single dimension. The occurrence of a performance anomaly under one setting is identified as a single point in this space. It was observed that if samples were chosen randomly and independently, the chances of missing a bug decrease exponentially with the increase in the number of samples. Since, anomalous settings could be due to multiple bugs, a hyper-rectangular clustering algorithm was invented to offset the shortcomings of classical algorithms such as k-means.

For evaluation purposes these models were applied to data-intensive online servers hosted on Linux 2.6.10. These servers access large disk-resident data sets while serving multiple clients simultaneously. Using this scheme, four performance bugs in Linux were successfully discovered.

- ***Accurate and Efficient Replaying of File System Traces***

  Nikolai Joukov, Timothy Wong, and Erez Zadok, Stony Brook University

  Nikolai Joukov presented Replayfs, an accurate and efficient method to replay file system traces. Replayfs can replay traces faster than any known user-level system, and can even handle replaying of traces with spikes of I/O activity or high rates of events. In fact, with their optimizations in place, Replayfs can replay traces captured on the same hardware faster than the original program that produced the trace.

  Nikolai opined that in developing a file system trace replayer it is often difficult to identify its suitable position within the operating system stack. To this extent, user-level replayers are easier to implement and thoroughly exercise the file system, but they do not support memory-mapped operations and have high memory/CPU overheads. Network-level replaying avoids the high memory/CPU costs, but it often misses out on client-side cached or aggregated events that do not translate into protocol messages. Replayfs overcomes all of these shortcomings by installing itself, as a kernel module, just beneath the VFS level and above classical file systems. In doing so it enjoys direct access to the buffer cache, exercises control over process scheduling, and benefits from reduced context switching, though at the cost of reduced portability.

  During Q&A, Ralph Becker from IBM Almaden Research asked how Replayfs could handle traces from large-scale clusters. Nikolai replied that in such a case they would have to run Replayfs on multiple clients and be more intelligent while capturing traces. Daniel Ellard from Sun Microsystems wanted to know if the zero-copy optimization could be turned off. Nikolai replied, yes, it is configurable.

- ***TBBT: Scalable and Accurate Trace Replay for File Server Evaluation***

  Ningning Zhu, Jiawu Chen, and Tzi-Cker Chiueh, Stony Brook University

  In this talk, Ningning Zhu presented the design, implementation, and evaluation of TBBT, a comprehensive NFS trace replay tool. The author described TBBT as a turn-key solution that can automatically detect and repair missing operations in a trace, derive a file-system image required to successfully replay the trace, initialize and age the file-system image appropriately, and eventually drive the file server according to a user-configurable trace workload.

  The author began her talk by highlighting the shortcomings of synthetic benchmarks, which are currently the most common workloads for file-system evaluations. Time-varying and site-specific parameters make it harder for synthetic benchmarks to mimic real-world workloads. Also, the time taken to develop a high-quality benchmark is often outpaced by the time taken for changes to trickle in to the workloads of specific target environments. TBBT is proposed as a complementary approach to synthetic benchmarks and is aimed at evaluating the performance of a file system/server on a site by capitalizing on the file access traces collected from that site.

  During Q&A, someone from Seagate wanted to know how good this approach is in replaying the traces on a server that has capacities different from those of the one from which the traces were collected. The author replied that in order to evaluate this they would first have to classify traces according to localities within them.