;login:
JUNE 2013 VOL. 38, NO. 3
ELECTRONIC SUPPLEMENT

Conference Reports

FAST '13: 11th USENIX Conference on File and Storage Technologies

San Jose, CA

February 12-15, 2013

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Keith Smith began FAST '13 by telling us that 127 papers were submitted and 24 accepted, and that the attendance had almost reached the same level as 2012, which was the record year for attendance. There were 20 full-length papers, four short ones, nine with just academic authors, five industry-only authors and 10 collaborations. Smith said he enjoyed having people from academia and industry in the same room talking.

FAST is a systems conference, and the top topics in submitted papers were those tagged with file-system design and solid state storage. Cloud storage has increased over time, as has caching, while file-system architectures have actually been decreasing.

There were more than 500 paper reviews, totaling more than 350,000 words of commentary (about the length of the book *David Copperfield*).

Yuanyuan Zhou, the co-chair, presented the best paper awards. *Unioning of the Buffer Cache and Journaling Layers with Non-Volatile Memory* by Lee et al. won the Best Short Paper award, and *A Study of Linux File System Evolution* by Lu et al. won Best Paper.

File Systems

Summarized by Morgan Stuart (stuartms@vcu.edu)

ffsck: The Fast File System Checker

Ao Ma, EMC Corporation and University of Wisconsin—Madison; Chris Dragga, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Ao Ma considered the creation of a file system that supports a faster checking utility. Ao first reviewed the necessity of file system integrity checkers and repairers. He explained that significant work has been done to prevent corruption or misuse of file systems, but no solution can guarantee a system free of such corruption. Therefore, the file-system checker is often thought of as a last resort, but it hasn't seen much improvement in some time. Given capacity increases, complexity growth, and general enterprise dependence, that storage administrators must still depend on offline, slow, and unpredictable file-system checkers is unfortunate.

In order to significantly improve file-system checking, the standard e2fsck utility was analyzed. The e2fsck checker completes its repairs in five phases, but the authors found that the utility spends more than 95% of its time in phase 1. During this phase, the checker scans all inodes and their corresponding indirect blocks and even requires an additional scan if multiply-claimed blocks are detected. Clearly any improvements to file-system checking should target the actions performed in this phase. Ao introduced a novel pairing of a file system, rext3, and a file system checker, ffsck, both of which complement each other to accelerate file-system checking.

The rext3 file system modifies the layout of a traditional ext3 file system to decouple the allocation of indirect blocks and data blocks. The indirect region in rext3 stores the indirect blocks contiguously to allow quick sequential access. In applying this reformation, rext3 achieves an improved metadata density that a modified checker could leverage. The strict organization also reduces file system aging from fragmentation. The separation proves to not result in extraneous seeks because a drive track buffer will often cache multiple indirect blocks with a single track load.

The fast file system checker (ffsck) leverages the contiguous indirect blocks of rext3 to increase scan speed. Because the indirect blocks and corresponding data blocks are physically rather than logically sequenced, however, ffsck requires that all metadata be read before it can be checked. In order to avoid memory saturation from storing the entire metadata of large systems, ffsck separates the checking process into its inherent two phases: a self-check phase and a cross-check phase. The self-check phase must use all the metadata to verify the file inodes individually, but the cross-check phase only needs a subset of the metadata in order to perform redundancy-based checks across data. Therefore, the self-check is completed first, followed by the removal of data not needed by the cross-check, and finally the cross-check is performed. This method helps reduce the average memory footprint over time for the checker.

The end result of the file system file-checker cooperation is the ability to scan and correct the file system at nearly ten times the speed of e2fsck and without hindrance from disk fragmentation. In most cases, rext3 performance is similar to ext3, but does incur about a 10% penalty when dealing with smaller files. Impressively, rext3 actually outperforms ext3 by up to 43% for random reads and up to 20% for large sequential writes. These improvements are attained by improving journal checking with the metadata density and by more efficiently using the track buffer.

Andreas Dilger (Intel) asked whether the authors had considered submitting an upstream patch to ext3. Ao said that

the source code still needs to be polished, but they do intend to open source their work. Someone from Symantec Labs asked what level of performance increase was seen without the file system modifications. Ao explained that they can still achieve between 50% to 200% improvement with only the in-order scan. Brent Welch (Panasas) requested more detail about the indirect region—specifically, whether they enforced a hard limit and what they did if the region was filled. Ao said that the size was fixed and that further experimentation is required as an appropriate size is hard to determine.

Building Workload-Independent Storage with VT-Trees

Pradeep Shetty, Richard Spillane, Ravikant Malpani, Binesh Andrews, Justin Seyster, and Erez Zadok, Stony Brook University

Pradeep Shetty began with a simple question: "What should file systems do?" He explained that file systems must allow for crash recovery, perform efficiently for both sequential and random accesses, and provide low overhead application-level transactions. Most real-world solutions don't meet all of Pradeep's requirements, giving him and his co-authors their motivation. Pradeep further alluded to the major discourse for today's administrators: they can either choose fast lookup transaction-based relational databases or instead opt for file systems that support high volumes of sequential and random accesses. Often neither is completely sufficient, as modern workloads are large and complex with randomized access patterns.

The proposed solution describes a new data structure, the VT-Tree, based on LSM-Trees. The LSM-Tree provides fast random insertions but significantly slower queries, making the LSM-Tree popular in large data sets where queries can be parallelized. The LSM-Tree uses a memtable to hold r-tuples of recently inserted items in a buffer. Once the buffer fills, the memtable, along with a Bloom filter and secondary index, is flushed to disk. The combination of these components is referred to as an SSTable. Consequently, the workload produces more SSTables as more tuples are created. Because queries often have to search through the majority of the SSTables, the queries slow down over time. To combat this, a limit on the number of SSTables is typically used to bound the lookup and scan latency of the system. A primary weak point of the LSM-Tree is its repeated copying of tuples as SSTables are serialized and compacted to the disk. These copies in the minor compaction allow for the quick lookup, but are considered unnecessary if the incoming data is already sorted.

Following his explanation of the LSM-Tree, Pradeep began to outline the goals of their VT-Tree. In order to optimize the minor compaction, which produces the extra copies, stitching was introduced. Pradeep described stitching as way in which their system investigates the need for a merge during compaction. The stitching mechanism allows the VT-Tree to merge only the blocks that overlap and stitch non-overlapping blocks into appropriate locations. The repositioning of the tuples to perform a stitch introduces fragmentation and holes in the tree. This is prevented by storing the VT-Tree on a log-structured block device to allow a LFS-style defragmenter to reclaim lost space. The stitching threshold is the minimum size that a stitched region must accomplish in order for stitching to occur. This threshold therefore helps to limit the level of system fragmentation. The method for avoiding I/O in LSM-Trees is to use a Bloom filter, but the VT-Tree uses quotient filters instead to allow rehashing in RAM without the need for an original key.

Pradeep next outlined their file system, KVFS, and how it utilizes VT-Trees to provide novel functionality to a system. In actuality, KVFS translates requests into key-value operations that are then sent to KVDB, which then performs the necessary I/O operations. Three dictionary formats—nmap, imap, and dmap—can be used to create dictionaries, each backed by a VT-Tree. The nmap format is used for namespace entries, the imap format simply stores inode attributes, and the dmap format is used for the data blocks of files. The system's ACID transactions are snapshot-based, where each transaction gets its own private snapshot. This allows the system to avoid double writes and implement the standard begin, commit, and abort operations of transactional systems.

The resulting system, Pradeep described, performs comparable to other systems but still achieves the enhanced performance of standard LSM-Trees when performing random writes. This means that the VT-Tree is able to support both file system and database workloads efficiently. The transactional architecture supported by the VT-Trees provides 4% speedup with 10% overhead.

Peter Desnoyers (Northeastern) expressed concern about the system's background cleanup and asked whether the authors had pursued a way to adjust the stitching threshold to prevent cleaning from overloading the system. Pradeep said that they experimented with many thresholds and found 32 Kb or 64 Kb to work best. He added that while increasing the threshold may reduce fragmentation, it would negate the purpose of including it at all if it was increased too much. Margo Seltzer (Harvard) asked how their implementation differs from LFS segment cleaning. Pradeep agreed that it is indeed very similar and that they only look at the sequential data and examine the rate on it. The questioner further encouraged Pradeep to look at the age of the data as well.

A Study of Linux File System Evolution

Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Shan Lu, University of Wisconsin—Madison

Awarded Best Paper!

As winner of Best Paper, this analytic research presented a fascinating look into a significant portion of the Linux 2.6 file system development. The authors painstakingly reviewed 5096 patches across six file systems and categorized each patch as bug, performance, reliability, feature, or maintenance related.

Lanyue Lu began by noting the continued importance of local file systems to a crowd ripe with cloud storage enthusiasts. He explained that local storage is still common on mobile devices, desktops, and as the base file system for many cloud applications. Lanyue said that much can be learned from a large-scale development review, such as understanding where complexity comes from, how to avoid future mistakes, and to help improve current designs and implementations.

This comprehensive study required that Lanyue and his associates comb through eight years of Linux 2.6 file system patches. Utilizing each patch's commit message, code diffs, and community discussions, the authors accumulated granular data describing the process of developing major open source file systems. The file systems examined included ext3, ext4, XFS, Btrfs, ReiserFS, and JFS.

The researchers found that code maintenance and bug fixes account for the majority of the patches sampled, at 45% and just under 40%, respectively. Lanyue noted that the maintenance patches were deemed uninteresting early on and were not investigated in detail. The bug patches were further categorized into either semantic, concurrency, memory, or error code related bugs. Semantic bugs were the biggest offenders making up over 50% of all bug-related patches. Concurrency bugs were the next most common at about 20%. Interestingly, nearly 40% of these bug patches occurred on failure paths or error handling. Other than the bugs, performance and reliability patches also made up a significant portion of the patches studied, accounting for 8% and 7% of patches, respectively.

The results suggest that bugs do not necessarily diminish over time, as one might presume. Even the stable, well-tested, file systems seem to have a relatively constant rate of bug patches during the period. Of all the possibilities, data corruption bugs were the most dominant across all the file systems studied and caused the most severe problems, such as system crashes and deadlocks. Lanyue went on to discuss actual patch examples from each patch category, pointing out the types within each category responsible for the most patches.

Lanyue said that, although time-consuming, a large-scale study such as this is manageable and useful. He stressed the importance of research matching reality and said that history does indeed repeat itself.

Akshat Aranya (NEC Labs) asked whether any correlation between feature patches and bug patches was studied. Lanyue recognized this as possible area of study but said that he and his co-authors did not analyze it. Margo Seltzer asked just how in depth the group's initial study of maintenance fixes was before deeming them "uninteresting." Lanyue responded that these maintenance bugs were almost always attempts to simplify the core structure through refactoring and that it was difficult to relate it to a bug patch. Rick Spillane (Apple) asked about bug fixes introducing another bug. Lanyue confirmed that they found these and even labeled them as "fix-on-fix" patches.

The data set is available at http://research.cs.wisc.edu/wind/Traces/fs-patch/.

Caching

Summarized by Leonardo Marmol (marmoleox@gmial.com)

Write Policies for Host-Side Flash Caches

Ricardo Koller, Florida International University and VMware; Leonardo Marmol and Raju Rangaswami, Florida International University; Swaminathan Sundararaman and Nisha Talagala, FusionIO; Ming Zhao, Florida International University

Ricardo Koller began his presentation by pointing out the big performance gap between write-through (WT) and write-back (WB) policies for caches. Traditional caching solutions for network storage, he said, implement WT policies because these guarantee data consistency at the price of experiencing high latencies for every update. Inspired by the "Designing for Disasters" work, he described two new caching policies for locally attached SSDs, designed to perform similarly to WB while preserving point-in-time consistency. The first policy, ordered write-back (OWB), uses a graph to store the ordering dependencies for I/Os using only issue and completion times, in order to evict the block in the right order. The second policy, journaled write back (JWB), builds a journal on the cache and evicts transactions atomically over the network using an interface similar to that of Logical Disk. This policy also required modification to the network storage in order to write blocks atomically.

The experimental results showed the benefits that come with caching, not only read but also write requests for those applications that can tolerate some level of staleness. The caching solution was evaluated using several benchmarks, and the results showed that, in general, WT performed worse than any other policy. JWB outperformed OWB but not traditional WB. Other experiments were presented showing the throughput and the number of I/O updates sent to storage as a function of the cache size for each policy.

Wenguang Wang (Apple) asked whether it was possible to relax the ordering constraints of the OWB policy, pointing out that, typically, hard disks acknowledge the completion of writes once the data is in their internal buffer, and then these are not necessarily performed in the same order they were issued. Koller disagreed with the questioner's premise, saying that the order of writes in non-volatile caches sent to disk is maintained and therefore matters.

Warming Up Storage-Level Caches with Bonfire

Yiying Zhang, University of Wisconsin—Madison; Gokul Soundararajan, Mark W. Storer, Lakshmi N. Bairavasundaram, and Sethuraman Subbiah, NetApp; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Caching solutions determine the contents of the cache on-demand. As I/O requests come and go, the content of the cache changes to better reflect the current needs of the application. New technologies like SSDs have made it possible to increase the size of caches to be much bigger than DRAM memory, which slows the process of warming caches. To put things into perspective, Zhang mentioned that a cache of 1 TB takes about 2.5 hours to fill with sequential workloads and 6 days or more with random workloads. For these reasons, Zhang claimed that the on-demand approach to warming up caches is no longer appropriate. She proposed a solution, Bonfire, that monitors and logs I/O requests with the goal of speeding up the warming of caches by loading data in bulk.

To answer questions like what and how the data should be monitored and logged, and how to load warmed data into caches efficiently, Zhang et al. performed statistical analysis on the MSR-Cambridge traces [Narayanan'08]. The temporal and spatial access patterns found on the traces were used to shape the design goals of their system. Bonfire monitors I/O requests with a module that sits below the buffer cache and keeps a buffer for its own metadata. When the buffer is nearly full, this is written to a persistent storage in a circular log. When a cache restarts, Bonfire uses its metadata to load warm data from storage into the cache. In addition to metadata, Bonfire could also log data which can be used to further reduce the warm-up time.

The system was evaluated by replaying the MSR-Cambridge traces in a synchronous fashion using both metadata-only and metadata+data logging schemas and comparing them to the on-demand and always-warmed policies. The results showed that Bonfire could warm up caches from 59% to 100% faster than on-demand while reducing the storage I/O load by 38% to 200%. As a consequence, the I/O latency experienced by applications was reduced on average by 1/5 to 2/3 when compared to on-demand. Before concluding, Zhang mentioned the need for making more traces available to the research community and invited everyone to contribute.

Umesh Maheshwari (Nimble Storage) asked Zhang why they assumed that caches are volatile when they could use SSDs as caches and they are persistent. Zhang explained that even persistent caches need rewarming after a repartition of the cache or a server migration. Ajay Gulati (VMware) asked about the case in which workload does not follow the patterns seen in the study's traces. Zhang replied that Bonfire would default to on-demand. Someone asked how they kept stored data and Bonfire's buffer consistent. Zhang answered that the buffer is updated only after the data is written to storage. The questioner pointed out that this requires some form of synchronization among nodes sharing storage. Joe Buck (UC Santa Cruz) mentioned that Zhang's research group's logo is very similar to that of the Nintendo GameCube logo.

Unioning of the Buffer Cache and Journaling Layers with Non-Volatile Memory

Eunji Lee and Hyokyung Bahn, Ewha University; Sam H. Noh, Hongik University

Awarded Best Short Paper!

Eunji Lee pointed out that journaling is one of the most common techniques used by file system architects to provide data consistency. However, it comes at the cost of extra I/O operations. Lee suggested an interesting alternative that eliminates the extra I/Os associated with journaling while maintaining the same level of reliability by effectively using non-volatile memory. In particular, she argued for an architecture called UBJ that unifies the buffer cache and the journal into non-volatile memory. Unlike conventional journaling techniques, committing blocks in UBJ is a simple matter of marking them as frozen, eliminating both the copy operation and data duplication. In addition, frozen blocks are used as cache, reducing the latency of read requests. As in any journaling system, transactions are eventually checkpointed to storage, but UBJ makes use of copy-on-write techniques to allow the update of frozen blocks. This simple technique significantly reduces the latency of write-intensive workloads.

The UBJ prototype was implemented in Linux, and the NVM was simulated using DRAM. The implementation was compared to the ext4 file system configured to journal both data and metadata, and several I/O benchmarks were used to generate the workloads. The results showed that UBJ outperformed ext4 with 59% higher throughput, which translated into a 30% reduction in execution time on average. Due to the simplicity of the UBJ system's in-place commit mechanism, the latency of I/O is no longer dependent on the frequency of commit operations.

Richard Spillane (Apple) commented on how the buffer cache and in-memory journal were sized on one of the experiments. He suggested that she could have gotten a better performance by reducing the size of the journal and increasing the size of

the cache, as opposed to making them equal in size. Youyou Lu (Tsinghua University) asked about the performance penalty associated with protecting frozen blocks and performing the COW. Lee replied that the overhead is expected to be very small, but no data was available at the time.

Conference Luncheon

During the conference luncheon, the Test of Time award was presented for GPFS: A Shared-Disk File System for Large Computing Clusters, by Frank Schmuck and Roger Haskin of IBM Almaden Research Center. You can read this paper via http://static.usenix.org/publications/library/proceedings/fast02/schmuck.html.

Protecting Your Data

Summarized by Morgan Stuart (stuartms@vcu.edu)

Memory Efficient Sanitization of a Deduplicated Storage System

Fabiano C. Botelho, Philip Shilane, Nitin Garg, and Windsor Hsu, EMC Backup Recovery Systems Division

Storage sanitization can be described as any method that removes sensitive data from a device, such that it appears the guarded information never actually existed on the system. Effective sanitization methods have their place in many fields, including the government and highly regulated private sectors. With the rise of massive storage systems and deduplication, there is a need to revisit sanitization mechanisms.

After explaining the modern motivation for advanced sanitization methods, Fabiano Botelho explained that crypto-sanitization isn't a contender in this particular area for several reasons. Key management would be difficult in these large systems where blocks are shared in the namespace. Furthermore, crypto-sanitization sacrifices performance of normal file system operations to achieve its goal. Finally, the NIST and DOD do not accept encryption as a sanitization method. Fabiano solidified their requirements, stating that their solution must completely erase deleted data, maintain the availability of live data, use resources responsibly, and leave the storage system in a usable state while sanitization is being performed.

The widespread technique of deduplication and the need for bulk sanitization are the primary motivators of Fabiano's work. When files are written in a deduplicated storage system, the data is separated into chunks and each chunk's hash value is calculated. The hash value can be used to determine whether or not the chunk is unique, whether or not it needs to be stored. Files in these systems are represented as a list of fingerprints that can be used to reconstruct the original file. This methodology allows only one instance of duplicate chunks to be stored on the system, saving large amounts of storage. However, these chunk references present the primary challenge when attempting to sanitize a deduplicated storage system.

Fabiano and his co-authors investigated several methods of tracking unused data and objects known as dead chunks. After comparing the possible usage of reference counts, Bloom filters, bit vectors, and perfect hashing, they found that perfect hashing can best fulfill their requirements. Perfect hashing allows a mapping without collisions of a static key set, using a minimal number of bits to represent the mapping. The perfect hash function will map to perfect hash buckets that are variable size, but 16K fingerprints per bucket on average worked very well.

The five-step algorithm for read-only file system sanitization has Merge, Analysis, Enumeration, Copy, and Zero phases. The Analysis phase was described in more detail as the point in which the algorithm builds the perfect hash function, walks multiple perfect hash vectors in parallel, and records the range that the data structure is actually covering. An algorithm for read-write systems was also implemented, which must handle incoming fingerprints after both the Merge and Analysis phases. These chunk resurrections are handled by notifying the process of incoming deduplication and utilizing a second consistency point, or snapshot, to enumerate differences.

Three sets of experiments were used to formulate a control for storage systems. First, a system using only local compression with no deduplication wrote its entire file system at about 70 MB/s. Next, deduplication was added with a factor of about 7x, resulting in 5.06 GB/s data rates. This increase in performance correlating to the deduplication factor confirms that a system can scale performance according to the deduplication factor. The next benchmark added the sanitization mechanism as well as data ingest, running at 59% peak throughput and 70% peak throughput, respectively. Fabiano explained that this benchmark showed that the system's data ingest is CPU-intensive while the sanitization is I/O-intensive. A final benchmark removed the deduplication, leaving the sanitization and data ingest variables. Sanitization ran above 45% of its peak throughput in this test, with high CPU usage for the data ingest as well as high I/O usage for both sanitization and ingest.

Cheng Huang (MSR) asked if the deduplication system must hold all the fingerprints in memory in the first place. Fabiano recommended that Cheng attend the HP session the next day, where they describe techniques to avoid holding everything in memory. Cheng then asked whether the authors had looked into options other than the perfect hash data structure. Fabiano explained that they had not seen any better techniques.

SD Codes: Erasure Codes Designed for How Storage Systems Really Fail

James S. Plank, University of Tennessee; Mario Blaum and James L. Hafner, IBM Almaden Research Center

The past ten years have seen rapid growth in utilization of erasure codes to handle disk failures. However, recent research has exposed what James Plank terms the "RAID6 Disconnect": that storage administrators are effectively using entire disks to tolerate common latent sector errors rather than full disk failures. Latent sector errors are particularly bothersome because they are typically only detected once a read access is attempted on the data. This clearly wasteful use of resources has motivated James and his coauthors to develop an erasure code that can tolerate both full disk failures and latent sector errors. The goal is to allow administrators to devote the right amount of coding to match the failure mode.

James explained the theoretical view of a stripe to define their Sector-Disk (SD) code. More specifically, each disk holds r w-bit symbols in a system of n disks, where w is relatively small. The system also uses m disks and *s* sectors per stripe to tolerate simultaneous failures of any m disks plus any s sectors per stripe. The SD code uses Galois Field arithmetic, where more w-bit symbols decrease speed but make the code more robust. James noted that this Reed-Solomon-like code has large amounts of documentation and open source code, and said he would spare the audience the mathematics.

The SD code is slower than Reed-Solomon, but it outperforms the solutions that the SD code could replace. For example, replacing RAID6 with a one-disk-one-sector code achieves higher performance with less dedicated storage. A complete open source implementation, in C, was made available the week of the conference. The source code is intended to act as template for engineers wishing to use the code or experiment with it.

Someone pointed out that the assumption seems to be that the latent errors are somewhat random and therefore small in number, but disk drives, instead of flash drives, could have many kilobytes in error. James explained that the implementation of the code must have the sector size defined as sufficiently large to encompass these larger failures. Geoff Kuenning asked "What am I getting?" since the SD codes don't really solve the two disk failures previously resolved by RAID6. If RAID6 is used, you are already protected from both types of failures. James explained that if you want to allow for more disk failures, you need to increase the m for disks. He suggested that models be used to examine the need to tolerate these failures. IBM researchers performed data loss modeling to investigate the data loss of SD coding versus RAID6 and they showed that SD can get higher reliability.

HARDFS: Hardening HDFS with Selective and Lightweight Versioning

Thanh Do, Tyler Harter, and Yingchao Liu, University of Wisconsin—Madison; Haryadi S. Gunawi, University of Chicago; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Thanh Do began by describing the implementation of the cloud's reliability, describing it as complex systems made up of thousands of commodity machines, where once-rare failures become frequent. In these systems, machine crashes and disk failures have been generally mitigated. However, Thanh described "fail-silent" failures as a continuing problem for these large-scale systems. The fail-silents are failures where the machine or program exhibits incorrect behavior but doesn't entirely halt the system. These failures can be caused by many problems, but are often the result of corrupt memory or software bugs. The defining factors of fail-silent failures are that standard crash recovery mechanisms do not combat them, as the issue can quickly propagate across collaborating machines. Current solutions for these failures, found in N-Version programming's redundant implementation methodology, require extensive resources and engineering effort which results in its rare deployment.

Thanh introduced selective and lightweight versioning (SLEEVE) to combat the fail-silent failures that he describes. Rather than "telling a lie" by continuing to run after a silent failure, SLEEVE exploits the crash recovery support for systems if it detects a failure with its trusted sources. Detection of the erroneous operations is achieved by utilizing a second lightweight implementation of the functionality that requires SLEEVE's protection.

SLEEVE is described as selective due its small engineering effort and its ability to target important functionality for protection. For instance, the error checking can target bug sensitive portions of a program or system, such as subsystems that are frequently changed or even currently unprotected with internal mechanisms. The lightweight aspect of SLEEVE describes the absence of full state replication. Instead, SLEEVE encodes states to reduce required space. The hardened version of HDFS (HARDFS), protected with SLEEVE, was able to detect and recover from 90% of random memory corruption faults and 100% of the targeted memory corruption faults. Furthermore, HARDFS was able to detect and recover from five software bugs injected into the system.

SLEEVE is composed of four subsystems: an interposition module, state manager, action verifier, and a recovery module. The state manager only maintains important states of the main version and only adds new states incrementally. The state manager must also understand the semantics of the protocol messages and events in order to correctly update the state. The state manager encodes states with counting Bloom

filters, which supports insert, delete, and exist operations. Thanh noted that Bloom filter false positives are rare and that they simply lead to a tolerable yet unnecessary recovery. The action verifier performs micro-checks to detect incorrect actions in the main version. The recovery module supports both full recoveries, described as a crash and reboot, and micro-recoveries in which corrupted states are repaired from trusted sources.

The HARDFS implementation hardens HDFS's namespace management, replica management, and its read/write protocol. Thanh and his co-authors found that HARDFS reduced the number of silent failures from 117 to 9, which ultimately increased the number of crashes from 133 to 268. Additionally, by using the counting Bloom filter, their implementation incurred a relatively small space overhead of 2.6%. Thanh concluded by saying that a crash is better than a lie and that HARDFS turns these lies into crashes and leverages existing recovery techniques to bring systems back online.

John Badger (Quantum) asked about the Bloom filter and how facts are represented. Thanh said that only yes/no verification is supported and that it ultimately depends on the property you want to check; no "magic rule" can be applied. Brent Welch expressed concern about false positives, crashes, and the potential for a crash loop. Thanh agreed that this was possible and informed the audience that crashes can be counted and a hard limit for stopping crash loops can be enacted. Next, Rick Spillane cautioned against Thanh's statement of the Bloom filter's 2.6% overhead, telling him that it grows linearly. Finally, Jacob Lorch pointed out that since SLEEVE is the ultimate arbiter of the system, a bug in SLEEVE can potentially cause catastrophic consequences.

Big Systems, Big Challenges

Summarized by Min Li (limin@cs.vt.edu)

Active Flash: Towards Energy-Efficient, In-Situ Data Analytics on Extreme-Scale Machines

Devesh Tiwari, North Carolina State University; Simona Boboila, Northeastern University; Sudharshan Vazhkudai and Youngjae Kim, Oak Ridge National Laboratory; Xiaosong Ma, North Carolina State University; Peter Desnoyers, Northeastern University; Yan Solihin, North Carolina State University

Devesh presented Active Flash, an in-situ data analysis method, to help improve the performance and energy efficiency for scientific data analysis tasks. He started with the introduction of a two-step process of scientific data analysis which consists of scientific simulation and data analysis and visualization. Conventionally, data analysis is performed offline on a small-scale cluster involving expensive data migration between compute and storage infrastructure resulting in extra energy cost. Devesh observed enabling trends: SSDs are increasingly adopted in HPC for higher I/O throughput and energy efficiency; SSD controllers are becoming powerful; idle cycles exist at SSD controllers due to the natural I/O burst of scientific workload etc. Devesh proposed conducting scientific data analysis on SSD controllers in parallel with simulation without affecting I/O performance.

He organized the discussion of system design around two questions: (1) if SSD are deployed optimizing only I/O performance, is active computation feasible? (2) how much energy and cost saving can Active Flash achieve? The main constraints of SSD deployment without active computation support are capacity, performance, and write durability. On the other hand, modeling active computation feasibility depends on simulation data production rate, staging ratio, and I/O bandwidth. Their results showed that most data analysis kernels can be placed on SSD controllers without degrading scientific simulation performance. Moreover, he observed, additional SSDs are not required to sustain the I/O requirement of scientific simulations even with active computation enabled. Compared with an alternative approach of running active computation on partial simulation nodes, he suggested that Active Flash is able to achieve the same performance but with lower staging ratio and infrastructure cost.

He went on to analyze the energy and cost saving of Active Flash. Modeling a Samsung PM830 SSD, they considered multiple components such as energy consumption of I/O, compute idle periods, data movement, etc. He also mentioned briefly how they modeled the energy consumption of two other state-of-the-art approaches. The results showed that Active Flash is more cost and energy efficient compared with other approaches in many cases. Finally, he introduced the prototype which they developed, based on the OpenSSD platform, demonstrating that scientific data analytics with Active Flash is viable with OpenSSD.

Dave Anderson (Seagate) wondered whether SSDs have enough resources to perform the complex task designed in the paper. Devesh replied that he had researched several products and believed that the SSD controller will be more powerful and have more cores to do complex tasks, such as data analytics, in the near future. Song Jiang (Wayne State University) asked if some intelligence is implemented on the SSD controller. Devesh replied yes. The implementation allows the SSD controller to communicate with hosts and perform data analytics. Song followed up by asking how the active cache handles data that is striped across SSDs. Devesh said that in that case, they would need frameworks such as MapReduce to help coordinate between different SSDs and perform analysis.

MixApart: Decoupled Analytics for Shared Storage Systems

Madalin Mihailescu, University of Toronto and NetApp; Gokul Soundararajan, NetApp; Cristiana Amza, University of Toronto

Madalin started by pointing out that enabling data analytics platforms, such as MapReduce and Pig, to directly use data on enterprise storage can help eliminate the two-storage-silos problem. The traditional two storage silos require dedicated compute infrastructure and additional time to migrate the data, and increase the hardware cost in terms of expense and number of errors. Madalin then presented MixApart, a scalable on-disk cache which allows distributed computation frameworks to use single enterprise storage and supports transparent on-demand data ingestion.

Effective design of MixApart comes from the analysis and understanding of MapReduce workloads. Madalin introduced three key insights they observed: (1) jobs exhibit high data reuse rate; (2) the input phase of a MapReduce job is usually CPU intensive; (3) the I/O demands of jobs are predictable. He also showed that with a high data reuse rate, MixApart can effectively support around 2000 parallel tasks using an envelope calculation demonstrating the compute scale of MixApart. With the goal of preserving the scalability and performance gained from data locality and efficient bandwidth utilization of storage, cache, compute node, Madalin mentioned that MixApart designed per-job task I/O rates and job scheduling policy to maximally overlap computation with data fetch. More specifically, he introduced two components, a compute scheduler, which allows assigning map tasks to nodes with cached data, and a data transfer scheduler, which facilitates just-in-time parallel data prefetch within and across jobs based on job I/O rate prediction. He also illustrated MixApart in action by using an example. They reengineered Hadoop by implementing a cache-aware compute scheduler as a variant of the Hadoop task scheduler, and a data transfer scheduler as a module within the namenode. They also reused the namenode as the XDFS metadata manager and added support within HDFS to enable caching stateless data. In their evaluation, they ran MixApart on Amazon EC2 with three types of EC2 instances and compared with Hadoop. They found that MixApart can reduce job durations by up to 28% compared to the traditional ingest-then-compute approach and can closely match the performance of Hadoop when the ingest phase is ignored for HDFS.

Akshat (NEC Labs) asked whether they had considered the workloads that were I/O intensive in the Map phase. Madalin admitted that there is not much they can do if the workloads are I/O intensive in the Map phase. However, the Facebook trace they analyzed had shown that the average task's effective I/O rate is low, which allows moving data from the shared storage to distributed cache. He argued that there are efforts to scale out the shared storage system to provide more bandwidth, which enables MixApart to sustain large clusters. He also mentioned that they had the notion of limiting the network bandwidth consumption of MixApart to make sure it does not compete with regular network traffic. Kadir Ozdemir (EMC) asked whether they had thought of a case in which the system would affect the performance of the enterprise system. Madalin responded that they had done some experiments in terms of performance isolation, arguing that the quanta-based scheduling effectively minimized the interference effects. Joe Buck (UC Santa Cruz) asked whether he had noticed a trace from CMU which demonstrated that 99% of data are actually processed within an hour, which means a better strategy would be to stream the data directly into the cache system instead of just-in-time prefetching. Madalin replied that it was a great use case. Since their approach is more generic, their system could always plug in better prefetching schemes to accommodate special scenarios like the one just mentioned.

Horus: Fine-Grained Encryption-Based Security for Large-Scale Storage

Yan Li, Nakul Sanjay Dhotre, and Yasuhiro Ohara, University of California, Santa Cruz; Thomas M. Kroeger, Sandia National Laboratories; Ethan L. Miller and Darrell D. E. Long, University of California, Santa Cruz

Li began by pointing out that current HPC systems store their sensitive data using an unencrypted or simply encrypted approach, which increases the chance of data leakage due to an increased chance of compromised nodes within these large-scale HPC centers. These HPC systems depend on a vulnerable security model which has a hard exterior and a soft interior. There are also concerns of leaking critical information from both malicious insiders and untrusted service providers. However, he mentioned that traditional data encryption techniques could not be directly applied to peta-scale data sets since they are either coarse-grained or incur high key-management overhead. Moreover, they could not provide security even when few nodes are compromised or when the service provider is untrusted. To solve the problem, Li introduced their system, Horus, which enables fine-grained encryption-based security for peta-scale data sets with low key management overhead. The key idea was to use keyed hash trees (KHT) to generate different keys for each region of a file and allow keys to be produced for variously sized regions based on users' need. He stressed that by carefully designing KHT, Horus greatly simplified key distribution and key storage.

Li explained how Horus is made up of three major components: key distribution cluster (KDC), Horus client library, and key exchange protocol. KDC is stateless and independent from the storage and compute nodes within the HPC

system, which can help provide security, scalability, and easy deployment. Because only the KDC knows the root key while compute nodes receive the needed keys , any data leakage is confined when nodes are compromised. He then explained the key distribution process through an animation followed by a description of key distribution protocol. The experiments testing the raw performance of KDS showed that a single KDS can sustain about 140,000 queries per second, and it scales linearly with the number of KDSes. Next, he presented an experiment to adjust the system parameters, KHT branch and depth, in order to explore the tradeoff of shifting workloads between servers and clients. He showed that Horus is flexible enough to balance the compute resource between the KDS client and the network. He concluded that Horus supports fine-grained security, is easily deployed, and has high performance.

Mark Lillibridge (HP Labs) asked how to revoke permissions. Li answered that they chose to have two root keys for a file; when a client tries to access a region, it will test which key works for the file. The paper has a detailed discussion. Xubin He (Virginia Commonwealth University) asked how to handle a case in which keys are randomly scattered. Li replied that the read/write workloads are usually in a range. If the depth of KHT is as big as 28, Xubin followed up, what would be the overhead? Li replied that the KHT needs width not depth, and suggested referring to the paper for more details. Bill Bolosky (Microsoft Research) suggested trying different hash functions. Li responded that the focus here was to study the property of KHT; choosing a hash function could be future work. Bill said that using an inappropriate hash function would affect the performance. Li admitted that was true.

Poster Session and Reception I

Summarized by Muthukumar Murugan (muru0007@umn.edu)

SLM: Synchronized Live Migration of Virtual Clusters Across Data Centers

Tao Lu, Morgan Stuart, Xubin He, Virginia Commonwealth University

The authors address the problem of live migration of virtual clusters across geographically distributed datacenters. They claim that synchronizing the migration of all VMs in a virtual cluster can reduce the cost of communication and data sharing among VMs through the low bandwidth WAN and hence can avoid any significant performance degradation in the applications.

The proposed architecture has three components: (1) a status monitor to monitor the available resources and the resources currently used by VMs; (2) a migration simulator that predicts the migration impact on the performance of the VMs based on modeling and profiling of the system; and (3) a migration manager that initiates and schedules the migration of each VM. Contact: Tao Lu, cstao.lv@gmail.com

Energy-Aware Storage

Yan Li, Christina Strong, Ignacio Corderi, Avani Wildani, Aleatha Parker-Wood, Andy Hospodor, University of California, Santa Cruz; Thomas M. Kroeger, Sandia National Laboratories; Darrell D.E. Long, University of California, Santa Cruz

This work tries to address the problem of energy consumption in future large-scale HPC storage systems. The two issues that are addressed are providing high bandwidth and/or capacity under power constraints and reducing data movement to save power. The work proposes a new metric called "energy score," which accounts for the energy consumed by all components in the process of the data object generation and is comparable between systems. The work explores multiple options such as near-node storage, use of SSDs, and extensive use of compression, and it studies the impact of proposed approaches on energy consumption of the storage systems.

In order to evaluate the proposed approaches on large complex computer systems, the authors built a comprehensive energy simulator. They also proposed exploring energy-efficient data allocation to increase idle times in storage devices so that they can be transitioned to low-power modes. Contact: Yan Li, yanli@ucsc.edu

On-Demand Indexing for Large Scientific Data

Brian A. Madden, Aleatha Parker-Wood, Darrell D.E. Long, University of California, Santa Cruz

This work proposes an efficient on-demand indexing scheme for large-scale scientific data. The proposed system consists of three components: the filter, the indexer, and the storage substrate. The filtering process creates a map of files to features and attributes. The indexer manages the indices on the filtered data and avoids expensive parsing of all files by narrowing the search based on the filter data. Transducers specific to different file formats help in the filtering process as data is ingested. The filter and index are stored as column stores which serve as the storage substrate. Currently transducers have been built for CSV and XML formats, and Apache HBase is used as the column store. Contact: Brian A. Madden, madden@soe.ucsc.edu

Efficient Use of Low Cost SSDs for Cost Effective Solid State Caches

Yongseok Oh, Eunjae Lee, University of Seoul; Jongmoo Choi, Dankook University; Donghee Lee, University of Seoul; and Sam H. Noh, Hongik University

In this work the authors propose the use of Hybrid Solid State Cache (HySSC), a combination of SLC (Single Level Cell) and TLC (Triple Level Cell), to reduce the cost of Solid State Caches by integrating high performance SLC with low cost TLC. HySSC manages the SSC device, takes care of page replacement in the cache, and maintains the mapping between logical and physical blocks. HySSC manages SLC

SSC as read/write and TLC SSC as read-only. The proposed architecture is evaluated with the extended version of the DiskSim simulator and real-world workload traces. Contact: Yongseok Oh, yongsukoh@gmail.com

Energy-Efficient Cloud Storage Using Solid-State Drive Caching

Jorge Cabrera, Salma Rodriguez, Jesus Ramos, Alexis Jefferson, Tiffany Da Silva, Ming Zhao, Florida International University

This work explores the use of SSDs as a near-node storage layer to reduce the power consumption of storage systems. SSDs consume a lot less power than hard disks and are much faster than hard disks for certain workloads. The work uses a modified version of an existing SSD block-caching solution called DM-Cache to enable a write-back cache for the primary storage. A user-space daemon is implemented to talk to the shared storage layer in order to spin down or spin up the disks.

The experiments are carried out on a shared storage device with and without the SSD cache layer. The authors report significant savings in power consumption when the I/O requests are served from SSDs. Contact: Jorge Cabrera, jcabr020@fiu.edu

Cloud Storage System which Prohibits Information Leakage on Both Client and Server

Kuniyasu Suzaki, Toshiki Yagi, Kazukuni Kobara, National Institute of Advanced Industrial Science and Technology (AIST); Nobuko Inoue, Tomoyuki Kawade, Koichiro Shoji, SciencePark Corporation

This work proposes a mechanism to prevent information leakage on client and servers in a cloud storage system. The proposed system, Virtual Jail Storage System (VJSS), encrypts a file using All-Or-Nothing Transform (AONT), and cuts out a part of the encrypted file as a split tally. The split tally is stored in a local storage in the client, and the remaining portion of the file is stored in the cloud storage system after encoding with Reed-Solomon error correcting code. The original file is only reconstructed in the VJSS which has the corresponding split tally. The encryption and split tally prevent information leakage from servers.

The reconstructed file in the VJSS can be opened by a suitable application but cannot be copied, printed, or screen-captured and pasted. These actions are prevented by the access control library called NonCopy. NonCopy hooks APIs of the Windows kernel, functions of DLL, and event handler I/O APIs, and prevents the action related to information leakage. The current VJSS implementation is based on Loopback Content Addressable Storage (LBCAS) for Windows, which uses "Dokan" for user-mode file system and BerkeleyDB for managing data. Contact: Kuniyasu Suzaki, k.suzaki@aist.go.jp

Offline Deduplication-Aware Block Separation for Solid State Disk

Jeongcheol An and Dongkun Shin, Sungkyunkwan University

Summarized by Vasily Tarasov (tarasov@vasily.name)

Jeongcheol An presented a deduplication-based technique that increases the lifespan of Solid State Disks (SSDs). The method consists of inline and offline steps. During the inline step, the SSD computes a CRC32 checksum of every incoming chunk (the size of a chunk is equal to SSD's page size). CRC32 is not a collision-resistant hash, so it is used to classify chunks into those containing unique data and those of undetermined status. CRC32 is 12.5 times faster than collision-free hash functions such as SHA-1, so write latency is not severely penalized by the inline step. The data that is classified as unique is separated on the SSD from undetermined data. Later, during the offline step, the actual deduplication with strong SHA-1 hashes is performed. The number of pages invalidated by the deduplication in the undetermined area is significantly higher than when no block separation is used and, consequently, the number of page copies during garbage collection decreases considerably (by up to 5 times in some experiments). Associated write amplification diminishes and the lifespan of the SSD increases. Contact: Jeongcheol An (luckyjc7@skku.edu)

Extension of S3 REST API for Providing QoS Support in Cloud Storage

Yusuke Tanimura, National Institute of Advanced Industrial Science and Technology (AIST); Seiya Yanagita, National Institute of Advanced Industrial Science and Technology (AIST) and SURIGIKEN Co., Ltd.

Though popular today, the S3 REST API does not allow a user to specify performance reservations for read and write throughput. Yusuke Tanimura presented an extension to the S3 REST API that provides a QoS capability to the base protocol. The extension adds new optional arguments to the already existing 'PUT Bucket' and 'Put/Get Object' operations. In the 'Put Bucket' operation, a user can specify the bucket size, its lifetime, and read/write throughput reservations. In the 'Put/Get Objects' operation, one can specify a reservation ID. Reservations in this case are made using an external tool, but in the future such commands can be added to the main protocol. The authors implemented the extension for Papio backend, which already supports QoS internally. Preliminary results demonstrate a good control over the throughput reservations. Contact: Yusuke Tanimura (yusuke.tanimura@aist.go.jp)

Improved Analysis and Trace Validation Using Metadata Snapshot

Ian F. Adams and Ethan L. Miller, University of California, Santa Cruz; Mark W. Storer, NetApp; Avani Wildani and Yangwook Kang, University of California, Santa Cruz

The fact that an I/O trace does not miss important activities is a crucial requirement for making true trace-based conclu-

sions about the workload. Ian Adams presented an interesting approach for determining the coverage of a trace. Before the tracing starts, an initial file system metadata snapshot is taken. Immediately after the tracing is over, another snapshot, called a reality snapshot, is taken. By applying the trace records to the initial snapshot, one can obtain a so-called expected snapshot. The analysis of the differences between the expected and the reality snapshots allows identifying the coverage of the trace. The authors provide several examples of such an analysis that determines the periods of the logger failure, missing creates, renames, and permission changes. Contact: Ian F. Adams (iadams@soe.ucsc.edu)

An Efficient Data Deduplication Based on Tar-Format Awareness in Backup Applications

Baegjae Sung, Sejin Park, Youngsup Oh, Jeonghyeon Ma, Unsung Lee, and Chanik Park, Pohang University of Science and Technology (POSTECH)

Sejin Park presented an approach to improve the chunking algorithm for tar-files. It is known that typical tar-files consist of numerous concatenated sub-files. Traditional chunking algorithms, such as fixed chunking and content defined chunking (CDC), ignore sub-file boundaries, which degrades the deduplication ratio. The authors added to the Opendedup SDFS file system the ability to form chunks using the sub-file boundaries in tar files. Their experiments demonstrate that deduplication ratio for 20 Linux kernel sources in a single tar file increased from 2.5 for CDC to almost 8.5 for CDC with tar-aware chunking. Contact: Sejin Park (cipark@postech.ac.kr)

GreenDM: A Versatile Hybrid Drive for Energy and Performance

Zhichao Li, Ming Chen, and Erez Zadok, Stony Brook University

Zhichao Li and Ming Chen presented a design for a novel device mapper target—GreenDM. GreenDM rests on top of several block devices with varying performance and power consumption characteristics, e.g., SSDs and HDDs. Using a number of approaches to determine the hotness of the data, GreenDM transparently migrates the data between SSDs and HDDs to improve performance and reduce power consumption. Preliminary results demonstrate up to 330% performance improvements and up to 80% power savings. Contact: Zhichao Li (zhicli@cs.stonybrook.edu) and Ming Chen (mchen@cs.stonybrook.edu)

Using Hybrid Cloud and Mobile Platforms to Enhance Online Education

Rachel Chavez Sanchez and Ming Zhao, Florida International University

Moodle is a known open source educational system similar to BlackBoard. Currently it lacks the integration with virtualization technologies, where each student could, for example, have his or her own VM for the experiments. Rachel Chavez Sanchez presented vMoodle, an educational system that incorporates Virtual Machines (VMs) in Moodle. vMoodle supports Web-based and mobile application interfaces. For mobile application, the authors worked on developing intelligent caching algorithms to improve user experience when high-latency networks are employed. Another problem the researchers tried to tackle is the support of live VM migration from a private to public cloud. This can be useful in cases when the university, for example, does not have enough resources to run all VMs on its own hardware. Contact: Rachel Chavez Sanchez (rchav010@cs.fiu.edu)

Policy-Based Storage System for Heterogeneous Environments

Dai Qin, Ashvin Goel, and Angela Demke Brown, University of Toronto

Applications are often decoupled from storage even though these applications and file systems produce a variety of workloads. Most modern storage systems are not aware of application workloads and requirements and interact with the upper layers using a simple block interface. According to Dai Quin and his colleagues, solutions like ZFS and Btrfs that integrate storage management in a file system are not flexible enough for heterogeneous environments. Instead, the authors propose a modular framework that determines application semantics using previously developed introspection and hinting mechanisms, and adjust storage policies accordingly. Policies also allow handling hardware with different performance characteristics. Currently the work is focused on implementing a fast and consistent mapping layer for the virtual block device. In the future, the authors plan to develop a library of policies for different applications and devices. Contact: Dai Quin (mike@eecg.toronto.edu)

Keynote Address

Disruptive Innovation: Data Domain Experience

Kai Li, Princeton University

Summarized by Rik Farrow

Kai Li told the story of Data Domain, a tiny company he founded that set out to replace the tape libraries used in data centers. They wanted to reduce the data footprint and network bandwidth by an order of magnitude, and did. What once required 17 tape libraries, a huge row of systems, became three 3U rack-mounted systems, in an example Li cited.

Li first asserted that innovation in large companies is very difficult, but he had a much more disturbing message for academics later in his keynote. He also said that you must have customer-driven technical development, work with the best venture capital firms, raise more money than you need, and hire the best people you can, even if you miss hiring goals. As for hiring people, Li stated the goal was to have people who work well together, minimizing egos, and using the best ideas. Li also said that some people demonized VCs, but good

VCs helped them avoid many detours, and also helped with software design reviews and business plans.

Li presented a very interesting graph that compared income growth to lines of code. In the early years of Data Domain (2001-2007), they were producing 100,000 lines of production quality code every year, while growing the engineering team from ten to one hundred over this period. Li encouraged startups to stay focused, to carefully pick what features you code for—that is your roadmap.

In the early days, they had to find companies willing to install their product instead of tape libraries. Tape libraries are expensive, and that helped them have high margins, as the Data Domain hardware costs were low. And even though storage customers are very conservative and slow to change, they succeeded by having a product that worked. Li disparaged both trade shows and analyst groups, like Gardner, as a way to create new markets. Data Domain was successful long before analysts ever noticed the company.

Li pointed out that large companies like EMC, NetApp, and HP hopped on the data deduplication bandwagon early, but discontinued their efforts soon after. Except for NetApp, these larger companies eventually acquired small companies with successful deduplication, just as EMC acquired Data Domain.

As for reasons why big companies often fail, Li suggested that engineers can lack motivation because they feel ignored by the company, including lack of incentives (stock options). Another reason is that the process used in big companies can be very wrong: checking with lead customers and research firms, and having many meetings structured around PowerPoint graphics. Li said, "Microsoft has reduced the productivity of millions," a statement greeted with enthusiastic applause. Another reason is that established companies are afraid of killing their own children, their cash cows, with new products that will compete with them.

Finally, Li put the focus on academic research. Deduplication was not developed in a university. He and others left their positions to focus on their research, saying you can't both research and innovate. If you want to do a startup, you cross over, rather than stand in "two canoes, research and startup innovation."

Someone from EMC asked how often can you go from academia to a startup with no prior experience. Li replied that he is not saying your prior research has nothing to do with success. It's just that the skill set for making a product successful is not taught in universities. You must put yourself into the market environment, and work to make your product successful. Margo Seltzer pointed out that Michael Stonebraker was another model of how this can work. Li agreed while pointing out that Stonebraker's current project (VoltDB) is already VC funded. Margo replied that Stonebraker said it is easier to get VC funding than research funding. How do we get a supportive systems research program going? Li had no answer. Someone asked if following technical trends was a good idea, and Li laughed and said that it was a good question. He pointed out that we are moving away from spindles to flash memory, using forms of cloud to minimize the cost of running private DCs. But moving to the cloud for large companies will not work because of the cost of network bandwidth.

Keith Smith (NetApp) wondered why large companies struggle with innovation, and Li replied that there is just not enough innovation juice in large companies, and that very little innovation has happened at Data Domain since it was acquired. Someone from EMC said that he was a researcher now, and Li countered by saying that Apple killed their research lab when Steve Jobs came back, and Amazon, Cisco and EMC don't have research labs. Li cannot find the destructive type of product developed mainly due to researchers, as they are not exposed to learning the market. Li did have a small research lab at Princeton which did make important contributions, including deduping data before network transmission. Randal Burns (John Hopkins) suggested SBIR (Small Business Innovation Research, sbir.gov) as an example of an attempt to extract innovation where it occurs in research. Li replied that SBIR is good and getting better, and that if there was a way for SBIR efforts to interact with many customers and team up with professionals, that would be great. Tech people are trained not to listen to other people, to believe "my idea is better and we know more than you," and after years of doing that, they lose the ability to hear what people want.

During his keynote, Li kept hinting that he had more to say, but wouldn't because his talk was being recorded (the video is available free online). As it was, Li's speech was both disruptive and very enlightening.

Deduplication

Summarized by Min Li (limin@cs.vt.edu)

Concurrent Deletion in a Distributed Content-Addressable Storage System with Global Deduplication

Przemyslaw Strzelczak, Elzbieta Adamczyk, Urszula Herman-Izycka, Jakub Sakowicz, Lukasz Slusarczyk, Jaroslaw Wrona, and Cezary Dubnicki, 9LivesData, LLC

Strzelczak presented a deletion algorithm for a distributed content-addressable storage (CAS) system with global deduplication. Data deletion with deduplication enabled all the time is motivated by the fact that otherwise the storage consumption would be increased significantly since successive backups are usually very similar. Strzelczak explained that data deletion with deduplication enabled was challenging.

Reasons were that deduplication resulted in several owners of chunks, dynamic system changes such as adding/deleting nodes, and failures. The requirements of deletion are continuous system availability, no read-only period, negligible impact on user operations, scalability, and fault tolerance. He then discussed a simplified data model in a CAS storage system followed by the challenges for deletion in CAS.

The data model for a CAS storage system has been trees built bottom up sharing deduplicated blocks. Challenges lie in the root set determination and block resurrection through deduplication. Their deletion algorithm is comprised of two phases: garbage collection and space reclamation. Each deletion run proceeds in three subphases. More specifically, to solve the problem that a retention root is written to block A after deletion starts yet A is deleted mistakenly, they proposed to allow the counter to be increased between the first and the second advance. To deal with the problem of block A becoming a duplicate after deletion start or being deleted wrongly, they use an undelete marker to preserve deduplicated blocks. Strzelczak went on to discuss how they extend the algorithm to support distributed CAS systems. The main difficulty is to decide consistently whether to preserve or remove all fragments of a block. The solution they proposed is to leverage redundancy of computation from good peers, which have good enough data state and counter validation. When mismatches are found, the deletion would be aborted.

In terms of implementation, Strzelczak explained that they implemented the algorithm with a commercial system, HYDRAstor, which is designed for backup and archival data. Their evaluation showed that the deletion reduces performance less than 30% while using 30% system resources under the default configuration. When given minimum system resources, the deletion impacts performance within 5%.

Neville Carvalho (EMC) asked what size of block and of identifier were used. Strzelczak answered the chunk size in HYDRAstor is 64 KB and the block address has 20 bytes. Mark Lillibridge (HP Lab) asked what happens if you put an undelete marker on a block that is later going to be deleted. Strzelczak replied that if a counter was positive, the system did not do anything, but otherwise it knew the block should be deleted.

File Recipe Compression in Data Deduplication Systems

Dirk Meister, André Brinkmann, and Tim Süß, Johannes Gutenberg University, Mainz

Meister introduced the concept of file recipes, which consists of lists of fingerprints of variable-sized chunks belonging to a file. He pointed out that file recipes occupy increasingly significant disk capacity since chunk data grow with post-deduplication space while file recipes grow with pre-deduplication space. To reduce the storage usage, he proposed compressing the file recipes by leveraging shortened code words rather than the fingerprint in the file recipe with low overhead in terms of memory, I/O, storage, and limited impact on write and restore speeds. He mentioned several assumptions: fingerprinting-based data deduplication systems, full chunk index availability, backup workloads, and reverse lookup necessity.

Next, Meister discussed three techniques used in their file recipe compression system. First, based on the observation that few chunks exhibit a high number of references such as zero-chunks, Meister proposed optimizing the case by using a one-byte code word, eliminating the need to store and look up the fingerprint. Secondly, they adopted a chunk index page-based approach to assign a code word to each fingerprint. In particular, the code word is assigned by the page ID and a unique identifier in the page. Thirdly, they utilized statistical mechanisms which generalize zero-chunk suppression and assign shorter code words to fingerprints based on statistics of the chunk usages. Meister went on to discuss the evaluation result. They used a trace-based simulation of weekly full backup. The figures he presented illustrated that their technique shrinks file recipes by more than 90%. He also concluded that file recipe allows additional storage saving, and it calls for exploration in storage deduplication research.

Michael Condict (NetApp ATG) asked whether they conducted experiments to reduce the average size of deduplication chucks since the compression of file recipes opens up opportunities to enable smaller size of chunks. Meister replied no, because this was not the only metadata overhead; as the size of chunks is reduced, the size of the chunk index increases and, for performance purposes, it was not quite special. Akshat Aranya (NEC Labs) asked whether they have the lookup table stored on SSD, mapping the compressed code words to the actual hash. Meister answered no, they did not need extra indexes; the code word itself consists of a page ID and unique identifier in a page, and can be used as the lookup keys. This is a quite nice property of this approach. Akshat then said he would follow up the question offline.

Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication

Mark Lillibridge and Kave Eshghi, HP Labs; Deepavali Bhagwat, HP Storage

Mark Lillibridge started by pointing out that the restore speed in chunk-based deduplication systems gets slower over time due to worsening chunk fragmentation. Due to the fact that chunks of backups get scattered around the whole system, restoration suffers when it has to jump back and forth

between different chunk groups of different ages. "Why not just defragment data periodically like we did for the disks?" Mark asked. He mentioned two reasons. One was that there usually did not exist a chunk layout that reduces the fragmentation for all the backups. The other was that rearranging chunks required expensive data movement.

To deal with the problem, they investigated three techniques: increasing the cache size, using a forward assembly area, and container capping. Next, he explained that they measure fragmentation by using the mean number of containers read per MB of backup restored since that is proportional to the extent of chunk fragmentation. They also measured the restore speed to be the inverse of mean containers read per MB of data restored, which allowed them to focus on the dominant cost, container reading, ignoring noise and other factors. He next described how a baseline restoration algorithm works and highlighted the effect of cache size on restoration speed. A graph illustrated how restore speed is inversely proportional to the measure of fragmentation and how larger cache size yielded faster restoration speed. Another finding was that the increasing fragmentation levels result in unacceptable restoration speeds in emergencies.

Mark explained the forward assembly area approach they designed which leverages the accurate knowledge from the backup recipe to perform better caching and prefetching and reduce the memory required during restoration. The method contained two variants, M-byte slices and rolling. M-byte slices control the amount of data to be assembled at one time in the forward assembly area that can be sent out in a single piece; rolling utilizes a ring buffer to effectively use memory to ensure that each container is loaded at most once every M bytes. He also showed an animation explaining how this technique works. Mark presented a chart showing how rolling effectively improves the speed factor compared with fixed case and LRU. An interesting point he mentioned was that given a backup workload, there would be sweet spots for LRU. Next, he switched to capping techniques which are used to exploit the tradeoff between deduplication and faster restore speed. The basic idea is to bound the containers read per MB ingested. Using an animation, he explained how it worked. They first divide the backup streams into segments, such as 20 MB fixed size, read a segment into I/O buffer, then check which of the chucks are stored and in which containers. Next they choose up to T old containers to use, and finally they compute the recipe section for the segment and append any new chunks to the open container. The evaluation results he mentioned illustrate that the capping technique provided a good tradeoff between deduplication efficiency and restoration speed.

One attendee asked about the impact of capping on ingestion time and backup speed. Mark answered that it was not much, and actually might be faster. He then suggested the attendee go to the poster session and have a more detailed discussion with him. Geoff Kuenning asked about the order of containers in the assembly area, and Mark replied that you could use a variant of an elevator algorithm. Fred Douglis (EMC) wondered whether by focusing on read performance you would have a really large look-ahead buffer for the recipe. Mark answered that there are various data structures that you can use in walking the recipe in linear time to create backpointers.

Work-in-Progress Reports (WiPs)

Summarized by Thanh Do (thanhdo@cs.wisc.edu)

A Deduplication Study for Host-Side Caches with Dynamic Workloads in Virtualized Data Center Environments

Jingxin Feng and Jiri Schindler, NetApp Inc.

Jiri Schindler said that it is unclear whether host-side caches are effective for dynamic workloads, e.g., virtual machine (VM) migration, in virtual desktop infrastructure (VDI). For such workloads, re-warming the caches after VM migration may be costly; the caches may contain many copies of the same content because each VM disk image is a separate entity. This work analyzes real dynamic VDI workload traces to assess the deduplication opportunity for large host-side caches. The study finds that deduplication can reduce the data footprint inside the caches by as much as 67%. As a result, deduplication enables caching larger data sets and improving cache hit rates, therefore alleviating load from networked storage systems during I/O intensive workloads.

IBIS: Interposed Big-Data I/O Scheduler

Yiqi Xu, Adrian Suarez, and Ming Zhao, Florida International University

Yiqi Xu started his presentation with the problem of I/O scheduling in current big-data systems like Hadoop MapReduce. Such systems do not expose management of shared storage I/O resources, leading to potential performance degradation under high I/O contention among applications. To solve that problem, he proposed a new I/O scheduler framework, called IBIS, which provides performance differentiation for competing applications. Implemented in the Hadoop framework, IBIS schedules I/Os based on application bandwidth demands at individual data nodes as well as across distributed data nodes. Preliminary results showed the benefit of IBIS. Someone from HP Labs asked whether the framework considered network contention. Yiqi answered that network contention was not a concern because IBIS exploited data locality, i.e., task was very likely scheduled in the same node where data was stored.

Adaptive Resource Allocation in Tiered Storage Systems

Hui Wang and Peter Varman, Rice University

Peter Varman explained the tradeoff between utilization and fairness in tiered storage systems, which are composed of SSD and disk arrays, with a simple example. The example showed that fairly allocating weights among clients with different hit ratios leads to non-optimized system utilization. Peter argued that a better allocation scheme would lead to better system utilization. To maximize system utilization, he proposed that weights for clients should be dynamically computed, based on their hit ratios. He showed some simulation results to prove that the proposed method helps to improve system utilization.

Trace Analysis for Block-Level Caching in Cloud Computing Systems

Dulcardo Arteaga and Ming Zhao, Florida International University; Pim Van Riezen and Lennard Zwart, Cloud VPS

The goal of this work is to assess the efficiency of using SSD caches in cloud systems. To that end, various traces from real-world private and public cloud systems are analyzed in order to answer key questions about the proper size of SDD caches and the caching policies that work best. The analysis shows some preliminary but interesting answers. For instance, I/O patterns vary across workloads; write-back cache is best for write-intensive workloads. Someone asked when the trace would be available. The answer was taken offline.

Radio+Tuner: A Tunable Distributed Object Store

Dorian J. Perkins, Curtis Yu, and Harsha V. Madhyastha, University of California, Riverside

Dorian Perkins started his presentation with a dilemma: there are no one-size-fits-all storage systems. As a result, it is hard for system administrators to choose the "right" systems for their workloads. Furthermore, as new workloads emerge, new systems need to be built. To address this challenge, Dorian proposed Radio+Tuner. While Radio offers flexible storage configuration, Tuner picks the most cost-effective configuration for Radio, given input specification about cluster hardware, application workload, and performance SLO. Finally, he showed initial results to prove the benefit of Radio+Tuner. Someone asked whether Dorian assumed the underlying storage system was a black box. Dorian clarified that he built the system from scratch, meaning no black-box assumptions here. Another person asked how many nodes Radio+Tuner could scale to. Dorian answered that in his current prototype, there were 12 nodes in the system; to scale to many more nodes would require a more accurate algorithm.

JackRabbit: Improved Agility in Elastic Distributed Storage

James Cipar, Lianghong Xu, Elie Krevat, Alexey Tumanov, and Nitin Gupta, Carnegie Mellon University; Michael A. Kozuch, Intel Labs; Gregory R. Ganger, Carnegie Mellon University

Building an elastic storage system that has high performance, good fault tolerance, flexibility to shrink to a small fraction of servers, and the ability to quickly resize the system footprint (termed "agility") with minimal data migration overhead is hard. Rabbit, an elastic distributed system, provides good agility but has poor write performance. JackRabbit improves Rabbit with new policies for data placement, workload distribution, and data migration. For instance, JackRabbit takes read requests away from low numbered servers, which are bottlenecks for writes, to improve write throughput. These new policies allow JackRabbit to shrink to a small number of nodes while still maintaining performance goals. Preliminary results show these policies as beneficial.

High Performance & Low Latency in Solid-State Drives Through Redundancy

Dimitris Skourtis, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz

SSDs provide many benefits such as fast random access, but they also have problems. For instance, garbage collection in the background can degrade performance, especially in the case of mixed workloads. This work proposes a new design based on redundancy that provides consistent performance and minimal latency for reads by physically isolating reads from writes. The idea is to have a cache layer sitting on top of two SSDs, each of which serves reads or writes. One major challenge is to keep data "in sync" across two drives. Initial results are promising.

SATA Port Multipliers Considered Harmful

Peng Li, University of Minnesota; James Hughes and John Plocher, FutureWei Technologies; David J. Lilja, University of Minnesota

This work studies the reliability of SATA port multipliers (PMs) by proposing a reproducible process for creating actual failures in the HDDs. The authors conducted two experiments, one with the SATA PMs and one without them. In all experiments, a fatal HDD error was emulated by removing the HDD's cover. Experimental results showed that without SATA PMs, HDD failure was independent. However, at least one combination of the SATA controllers, the SATA PMs, and the HDDs did not provide resiliency when a single HDD failed. Investigating why this occurred was left for future work. Someone made a comment asking for another way to emulate fatal errors without destroying the disk, perhaps by putting a bullet through it.

Beyond MTTDL: A Closed-Form RAID 6 Reliability Equation

Jon Elerath and Jiri Schindler, NetApp Inc.

Jiri Schindler argued that although simple, the original RAID reliability equation that expressed mean-time-to-data loss (MTTDL) is no longer accurate, because today RAID systems are much more complex, with many processes for proactive scanning and repair of media defects. Moreover, researchers now have a better understanding of HDD failure modes and non-constant time-to-failure distributions. As a result, Jiri proposed a new equation that is more accurate, easy to use, easy to understand, and could help system designers to quickly explore a variety of design points. The new equation takes into account many factors, such as HDD operational failures, their restorations, latent (sector) defects, and disk media scrubbing. The great news was that this new equation is available online for anyone who wants to try it out at http://raideqn.netapp.com/. Finally, Jiri presented some results showing that the new equation is more accurate than the original. Someone asked whether the new equation models "wetware," i.e., the human factor. Jiri answered that the model actually covers the human factor.

Reverse Deduplication: Optimizing for Fast Restore

Zhike Zhang, Preeti Gupta, Avani Wildani, Ignacio Corderi, and Darrell D.E. Long, University of California, Santa Cruz

Deduplicated storage systems suffer from data fragmentation, as more and more data are added and more data chunks are shared. Due to the nature of existing deduplication algorithms, the most recent backup is the most fragmented, resulting in performance issues. This work proposes to invert the deduplication process in order to make restoring the most recent copy more efficient. Specifically, new data segments will be written contiguously, and older data segments that share chunks in the new segments will reference those chunks. However, because older backups will develop more and more holes, restoring them would be costly. Preliminary results show that retrieving the most recent backup in reverse deduplication is more efficient than in traditional deduplication.

Quality-of-Data Consistency Levels in HBase for GeoReplication

Álvaro García Recuero, Instituto Superior Técnico; Luís Veiga, INESC-ID Lisboa, Distributed Systems Group

HBase only supports eventual consistency for replication between the local site and remote sites: updates are replicated asynchronously between datacenters. Thus, it is challenging to ensure a given level of quality of service for delivering data to remote master replicas. This work extends some of the main components of HBase to replace the eventual consistency model with an adaptive consistency one. It outlines the architecture of a quality-of-service layer proposed for HBase.

Something for Everyone

Summarized by Dorian Perkins (dperkins@cs.ucr.edu)

Shroud: Ensuring Private Access to Large-Scale Data in the Data Center

Jacob R. Lorch, Bryan Parno, and James Mickens, Microsoft Research; Mariana Raykova, IBM Research; Joshua Schiffman, AMD

Jacob Lorch addressed this question: how can we prevent the cloud from learning our private data? Even when encryption is used, cloud services can still learn up to 80% of the content in email. This approach is based on previous work on oblivious RAM (ORAM), a technique used to obfuscate a client's access patterns to data. However, the authors note that ORAM is far too slow in practice: for example, a map application serving a single map tile to one user can take up to one week! Shroud leverages parallelism to speed up this technique while preserving privacy, reducing I/O time, and providing fault tolerance.

Overall, Shroud aims to fetch data from the cloud without the service knowing which block a user actually wants to access. Shroud uses trusted, secure coprocessors (smart cards which cost approximately $4 each) throughout the datacenter as proxies to each storage node. Users convey requests over secure channels with these proxies, which then access the data in-parallel. The coprocessors then employ a binary tree ORAM selection technique to randomize the access patterns to data. Each time a block needs to be accessed, an adversary can only know how far down the tree the block may be, but has no idea where it actually is; subsequent access must use a different path to access the block. When a block is found, all coprocessors must send their blocks to the same node, which then uses a technique called oblivious aggregation to efficiently and securely compute a collective XOR. Jacob said that Shroud was deployed on 139 machines at MSR using emulated smart cards (due to availability), and was tested using various workloads, including Facebook images and map tiles.

Someone asked how Shroud scaled. Jacob said that performance increases linearly until around 10K coprocessors, where performance gains begin to taper off. Jacob noted that Shroud is still more about theory than practice, as performance is still very slow, taking about 45 seconds to serve a map tile, and around nine seconds for serving a tweet. The clear performance bottleneck is the low-bandwidth smartcards they use as coprocessors, which only have around 12 KB/s bandwidth. The authors leave as future work employing high-bandwidth tamper-resistant FPGAs as coprocessors to improve performance, and admitting the hard drive to the trusted computing base to allow the use of more efficient ORAM protocols.

Getting Real: Lessons in Transitioning Research Simulations into Hardware Systems

Mohit Saxena, Yiying Zhang, Michael M. Swift, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Mohit Saxena noted there has been much work on SSD design, and common evaluation methods focus on three techniques: modifying the SSD by replacing the FTL (generally limited to vendors); FPGA prototyping which is flexible and fast, yet hard and time-consuming; and simulators/emulators, which replay block traces and implement device models, and are generally used by the research community. Commonly, simulators are used when designing or evaluating new solid state drive (SSD) designs; however, the problem is that simulators cannot model real hardware accurately as they do not capture the complete interaction with the operating system in their model (e.g., timing dependencies, request alignment, etc.). Mohit pointed out that in the last three years, most papers have used simulators to validate their SSD designs. Instead, Mohit suggests a better approach using the OpenSSD hardware platform with a Jasmine board to develop new SSD designs. Yet the OpenSSD platform was not without issues, so Mohit explained how his team spent time optimizing this board to improve its performance.

Mohit discussed their prototyping experience with two previous works, Solid-State Cache (SSC) and Nameless-Writes (NW-SSD), noting the challenges, solutions, and lessons learned from optimizing their hardware design and evaluation suite for flash storage devices. SSC aims to improve performance compared to using an SSD as block cache, while NW-SSD introduces new commands to build cheap and fast SSDs, by exposing the flash block layout and interacting directly with the OS when serving reads and writes. Mohit separated his prototyping experience into three sections: new forward commands, new device responses, and real hardware constraints for SSC and NW-SSD. Mohit summarized each of his points with lessons learned. When designing new forward commands, Mohit urges that you should always consider all layers of the OS, especially I/O schedulers merging and re-ordering operations, and also consider the complete SSD ecosystem, including encoding sub-types and accelerating new command queues. Designers of new device responses should again consider all OS layers, such as race conditions for callbacks in the device and file system, and handling of frequent benign errors in the storage device drivers. A prototyping lesson also learned here is simplicity and correctness; make the Kernel-FTL a simpler block layer OS interface and enforce correct erase-before-overwrite operations in the Device-FTL.

In their performance evaluation, they compared their two systems (SSC and NW-SSD) with a bare SSD using filebench. They validated the previous performance claims of SSC (168% better than common hybrid FTL) design by showing that it performs 52% better than a faster page-map FTL and that NW-SSD can substantially reduce the amount of device memory required with performance close to a page-map FTL. In conclusion, Mohit found that OpenSSD is a valuable tool for evaluating new SSD designs. Mohit shared his first high-performance open-source FTL at http://cs.wisc.edu/~msaxena/new/ftl.html.

To Zip or Not to Zip: Effective Resource Usage for Real-Time Compression

Danny Harnik, Ronen Kat, Oded Margalit, Dmitry Sotnikov, and Avishay Traeger, IBM Research—Haifa

Danny Harnik lightheartedly began his talk by asking, "To zip, or not to zip, that is question." Danny introduced his work with the motivating goal of reducing time, cost, rackspace, and cooling requirements. The challenge of this work is to add "seamless" compression to a storage system with little effect on performance. Danny noted that it's okay to pay the compression overhead if you are going to gain something, but it is not always worth the effort. The authors' goal is to avoid compressing "incompressible" data, while also maximizing the compression ratio of their stored data. To tackle this problem, Danny noted that there is no established, accurate method for estimating compression ratio, outside of actually compressing the data. Other solutions included deducing from empirical application data or file extensions, but these are neither accurate nor always available.

Danny explained how the authors tackled this problem from a micro and macro scale. For macro, they considered a large multi-GB/TB volume in which the time to compress was on the order of hours, where estimates only took a short time (minutes) and they could actually obtain accuracy guarantees. At this scale, they choose M random locations to test local compression ratios and compute an average compression ratio for these locations. However, he noted that in practice, this straightforward sampling method can have issues of compression locality. So they tweaked their method to evaluate the "compression contribution" of single bytes from regions throughout the file, and define the contribution of a byte as the compression ratio of its locality region (where locality depends on the specific compression method at hand). They then proved through statistical analysis for estimating averages that their method estimated the overall ratio with guaranteed accuracy (the actual parameters depend on the sample size but do not depend on the volume size). This macro-scale compression estimate is also very useful as an evaluation and sizing tool. A version of this tool can by found by searching "IBM Comprestimator" or at this link: http://www-01.ibm.com/support/docview.wss?uid=ssg1S4001012.

Danny noted that at the micro-scale, they consider single write, KB-sized files which take milliseconds to compress; since estimation has to be ultra-quick they rely on heuristics. Danny pointed out that it's impossible to get guarantees as the locality in this case amounts to the entire chunk. They considered two approaches: a prefix-based estimation and a heuristic indicator method. In the latter they collect some basic indicators about the data and output a recommendation accordingly. For this method they employ a number of techniques to improve the time performance of the estimation. Danny discussed the performance evaluation of the two estimation methods on over 300 GB (17790 files) of mixed data types, showing that the heuristics approach wins out over the 1 KB prefix sampling, and both improve on the option of running a full compression on an 8 KB chunk. In a time versus compression tradeoff analysis, prefix compression has 74% CPU utilization with 2.2% capacity overhead, while the heuristics method has 65% CPU utilization at a nominally higher 2.3% capacity overhead.

In summary, Danny concluded that when most data is compressible use prefix estimation, when a significant percentage is incompressible use the heuristics method, and when most is incompressible, turn off compression and run macro-scale offline to detect a change. Michael Condit (NetApp) noted that other compression techniques are faster than the one studied in the paper, and this work depends on the compression algorithm's latency. Danny replied that the work generalizes to other methods as well, but may be less relevant to some. For example, Snappy is a compression algorithm that already uses prefix estimation.

Poster Session and Reception II

Summarized by Matias Bjorling (mabj@itu.dk)

Examining Scientific Data for Scalable Index Designs

Aleatha Parker-Wood, Brian A. Madden, Michael McThrow, and Darrell D.E. Long, University of California, Santa Cruz

Aleatha Parker-Wood argued that modern file systems with billions of files are no longer tractable for conducting searches of scientific data. The vast amount of data and ever larger metadata, describing scientific observations, has become unmanageable. They argue that databases optimized for sparse data, column-based compression, and high cardinality are a better choice as a file-system index database. They evaluated five architectures: row stores, column stores, key-value stores, document stores, and spatial trees and compared each in regard to sparseness, dimensions, cardinality, specialized access, and ad hoc queries. They found column and document stores to be efficient structures for the scientific metadata. Further investigations include novel indexing strategies, such as on-demand indexing on a per-column basis.

Reliability Analysis of Distributed RAID with Priority Rebuilding

Hiroaki Akutsu and Tomohiro Kawaguchi, Yokohama Research Laboratory, Hitachi, Ltd.

The storage capacity of hard drives has been increasing exponentially, leading to longer RAID rebuild times and increased risk of data loss. Distributed RAID is a technique to decrease the rebuild time. Because of the expanded rebuild range, more drives are prone to fault during rebuilding. Priority rebuilding is used to restore data with the lowest redundancy first. To estimate the redundancy reliability, Hiroaki Akutsu presented a reliability analysis of distributed RAIDs that they can use as a model. They found that distributed RAID reliability is roughly equal to that of a level-1 redundancy method (e.g., mirroring, RAID5); reliability becomes roughly constant, independent of the number of drives in a level-2 redundancy method (e.g., triplication, RAID6); and reliability increased due to the increase in the number of drives in the over level-3 redundancy method (e.g., triple parity RAID, high-redundancy erasure-coding).

Radio+Tuner: A Tunable Distributed Object Store

Dorian J. Perkins, Curtis Yu, and Harsha V. Madhyastha, University of California, Riverside

There are many storage systems today, each designed with its own specific workload and performance goals. However, no single distributed storage system design is cost-optimal for meeting performance goals of all workloads. Dorian Perkins presented Radio+Tuner, a tunable distributed object store (Radio) and its configuration engine (Tuner). Radio offers a simple GET/PUT interface, with three system components: NodeMetadataStore, DiskMetadataStore, and DataStore. Each component offers multiple implementations allowing for "mix-and-match" configurations, with the benefits that as new workloads emerge, new implementations may be added to the system (instead of designing a new system). Tuner takes as input the workload's parameters and performance SLOs, as well as hardware and component implementation specifications, and simulates the operation of Radio to obtain a GET/PUT latency distribution. It then outputs the lowest cost configuration which meets the workloads goals. Initial results show that Radio+Tuner is able to adapt to disparate workloads, and does so at up to 5x cost savings when using the Tuner-recommended configurations. Future work includes unifying Radio with prior solutions which consider consistency and availability requirements, and expanding Radio to handle multiple conflicting workloads on the same hardware.

JackRabbit: Improved Agility in Elastic Distributed Storage

James Cipar, Lianghong Xu, Elie Krevat, Alexey Tumanov, and Nitin Gupta, Carnegie Mellon University; Michael A. Kozuch, Intel Labs; Gregory R. Ganger, Carnegie Mellon University

Distributed storage is often expensive to scale and requires aggressive write periods when new nodes are added or removed. Recent research in elastic storage systems, such as Rabbit and Sierra, enable better elasticity by new data layouts and mechanisms, but both suffer from write degradation or poor agility. Lianghong Xu presented JackRabbit. It focuses on new policies, designed to maximize the agility of elastic storage, while accommodating both performance and fault tolerance. Evaluation shows that JackRabbit comes closer to the ideal machine hour elasticity (within 4%) and improves over state-of-the-art elastic storage systems by 6–120%.

High Performance & Low Latency in Solid-State Drives Through Redundancy

Dimitris Skourtis, Scott Brandt, and Carlos Maltzahn, University of California, Santa Cruz

Dimitris Skourtis presented an approach to having both high performance and low latency in solid-state drives using redundancy. By separating read and write patterns, only one drive is being written at a time. Thus, the other drive is solely available for reads. After a variable amount of time, the disk responsibility is switched. The to-be-written data is cached and then flushed. The evaluation shows reads have consistently less variation and double throughput for 256 KB blocks. Future work includes quality of service for mixed workloads and evaluation under live workloads such as databases and VMs.

A Deduplication Study for Host-Side Caches with Dynamic Workloads in Virtualized Data Center Environments

Jingxin Feng and Jiri Schindler, NetApp Inc.

Jiri Schindler presented their deduplication study of host-side caches in virtualized datacenter environments. Host-side caches can be rather large, and re-warming the cache for migrated virtual machines may take up to several hours. In virtual desktop infrastructure (VDI) deployments, a virtual machine is a separate entity, but the host-side cache might contain many copies of the same content even though the network-attached shared storage system would only store a single instance. The goal of their study is to explore the effectiveness of deduplication for large host-side caches running dynamic VDI workloads. Their preliminary results show a disk space saving of 54% to 67% using deduplication and a larger saving if reads and writes are observed separately. They argue that the "deduplication degree" metric captures a useful concept for evaluating cache effectiveness for dynamic workloads. Future work includes analyzing similarity of VDI traffic, deduplication sensitivity to cache block size, and other aspects that can improve the host-side cache in VDI environments.

Summarized by Jorge E. Cabrera (jcabr020@cs.fiu.edu)

Adaptive Resource Allocation in Tiered Storage Systems

Hui Wang, Peter Varman, Rice University

Peter Varman addressed the challenge of providing both fairness and high utilization guarantees in multi-tiered storage systems. Their work is centered around dynamically computing a reservation and limit value for all clients based on their hit ratio. These values are used to guarantee fairness by providing a minimum allocation, and to provide remaining I/Os to other clients to obtain maximum system utilization. Evaluation results using a process-driven simulator show that their allocation model can potentially pull up utilization to maximum throughput or close to it depending on the reservation values of all the clients. Future work entails extending the allocation model to include relative shares.

Quality-of-Data Consistency Levels in HBase for GeoReplication

Álvaro García Recuero, Instituto Superior Técnico; Luís Veiga, INESC-ID Lisboa, Distributed Systems Group

A major challenge in cloud storage systems is providing quality-of-data consistency levels for data-replication mechanisms. Alvaro García Recuero presented a mechanism to extend the replication mechanisms of HBase, an open-source version of BigTable. Currently, HBase uses a best-effort delivery mechanism of data by using an eventual consistency mode. The proposed approach is to leverage the vector field consistency model into a framework that provides the HBase core with a QoD layer that allows it to prioritize specific client replicas to deliver replica updates with the agreed quality of data. Current evaluation is pending, and expected results promise a reduction in bandwidth usage and more control of the interval when replication occurs.

IBIS: Interposed Big-Data I/O Scheduler

Yiqi Xu, Adrian Suarez, and Ming Zhao, Florida International University

Yiqi Xu presented IBIS (Interposed Big-data I/O Scheduler), which tries to solve the scheduling problem that exists in big-data systems (e.g., Hadoop/MapReduce) because they do not expose management of shared storage I/O resources. As such, an application's performance may degrade in unpredictable ways under I/O contention, even with fair sharing of computing resources. IBIS provides performance differentiation for the I/Os of competing applications in a shared MapReduce-type big-data system. IBIS is implemented in Hadoop by interposing HDFS I/Os and use an SFQ-based proportional-sharing algorithm. Experiments show that IBIS provides strong performance isolation for one application

against another highly I/O-intensive application. IBIS also enforces good proportional sharing of the global bandwidth among competing parallel applications, by coordinating distributed IBIS schedulers to deal with the uneven distribution of local services in big-data systems.

Trace Analysis for Block-Level Caching in Cloud Computing Systems

Dulcardo Arteaga and Ming Zhao, Florida International University; Pim Van Riezen and Lennard Zwart, Cloud VPS

Client-side caching by using SSDs can potentially improve the performance of shared block-level storage systems that can suffer from scalability issues when the number of clients grows. Dulcardo Arteaga presented a trace analysis for this type of caching with the goal of analyzing the effective use of SSD devices as caches. Specifically, there are three factors that are studied: size of SSD device through working set size analysis, a comparison of three caching policy configurations, and dynamic and static allocation of shared caches among concurrent VM clients. The types of traces analyzed include both public and private cloud environments comprising Web servers, file servers, and VM clients. The types of caching policies used are write-back, write-through, and write-allocate. Some of the interesting results show that both public and private clouds have an average cache hit ratio of 74% and 78%, respectively, using write-back policy. In addition, working set sizes can be accurately predicted 90 percent of the time.

Beyond MTTDL: A Closed-Form RAID 6 Reliability Equation

Jon Elerath and Jiri Schindler, NetApp Inc.

The complexity of RAID systems and new HDD technologies has risen to a level where old MTTDL models cannot be applied to obtain accurate results. New systems have improved designs that employ repair mechanisms that did not exist in older HDDs. Jiri Schindler presented a project based on developing a more accurate and reliable MTTDL equation model, specifically for RAID6 setups. The result of this research is a new closed-form RAID6 reliability equation that can better model data-loss events compared to the old MTTDL equation. This equation can yield estimations for HDD operational failures, latent defects, and disk media scrubbing. The equation was formulated by using a two-parameter Weibull distribution using parameters obtained from real-world data. The equation was verified against a Monte Carlo model simulation, and the results shows similar accuracy. In addition, the new MTTDL equation can yield results in milliseconds, whereas a single MC simulation ran between 14 seconds and 18 hours. A Javascript implementation of the model is available for the public at http://raideqn.netapp.com. Evaluation results show that in comparison to the old model, the new equation shows more realistic results when it comes to predicting the occurrence of failures.

Reverse Deduplication: Optimizing for Fast Restore

Zhike Zhang, Preeti Gupta, Avani Wildani, Ignacio Corderi, and Darrell D.E. Long, University of California, Santa Cruz

Preeti Gupta explained that as the number of shared data chunks increases, the amount of data fragmentation increases and can lead to decreased performance in deduplicated storage systems. In particular, the most recent backup is the most fragmented of this data. The goal of this project is improve the performance access of the most recent backup in deduplicated backup systems. The proposed approach entails the inversion of the deduplication process. Instead of mapping new chunks to already existing chunks, each new data segment is written contiguously, and older data is mapped to the new chunks. Evaluation results show that they can significantly reduce fragmentation for the most recent data segments. Specifically, retrieval time can be 4 to 19 times faster. While this solution is great for the most recent backup, it does pose a tradeoff for accessing older backups, which develop portions of data that are no longer referenced.

Flash and SSDs

Summarized by Leonardo Marmol (marmol@cs.fiu.edu)

LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives

Kai Zhao, Rensselaer Polytechnic Institute; Wenzhe Zhao and Hongbin Sun, Xi'an Jiaotong University; Tong Zhang, Rensselaer Polytechnic Institute; Xiaodong Zhang, The Ohio State University; Nanning Zheng, Xi'an Jiaotong University

Current SSDs use Bose-Chaudhuri-Hocquengham (BCH) error correction mechanisms. However, as NAND flash technology becomes denser it also becomes less reliable, rendering BCH incapable of dealing with the several types of interference present in NAND. As an alternative, Kai Zhao proposed the use of low-density parity-check (LDPC) techniques and explores its potential and limitations. LDPC is known to provide stronger error correction capabilities, but the performance penalty associated with LDPC has made it impractical so far. Zhao addressed these limitations with a technique that combines the simplicity and high speed of hard-decision coding with the strong error correction capability of soft-decision coding.

At a high level, the idea is to use hard-decision at first and only apply a soft-decision in the presence of failures. By combining look-ahead memory sensing to reduce the total latency, fine-grained progressive sensing and decoding as needed, and smart data placement interleaving, Zhao et al. managed to provide a solution that significantly reduced the average response time delay while still providing high reliability for dense flash technologies.

The implementation was evaluated using the DiskSim simulator and six sets of traces of different workloads. The experimental work flow consisted of running a high number of program/erase cycles followed by a baking session to emulate the wear-out recovery. The baking time was determined using Arrhenius's Law. Next, random data is programmed into the chips, and these are baked once again to emulate one month retention time. Finally, the data is read and compared with the original data to get page error statistics. The results showed a comparison between the proposed techniques and the basic two-step sensing process. In general, the combined use of look-ahead, progressive sensing, and interleaving lead to a reduction of response time delay from over 100% to less than 20%.

Joseph Tucek (HP Labs) asked how his solution would play with RAID systems with their own built-in error correction mechanisms. Zhao replied that having the upper layer doing error correction is an orthogonal solution that in most cases will not suffice. Peter Harllee (CMU) asked whether the error information was used to redefine new voltage thresholds. Zhao answered that it can only be done at the word granularity. On a related note, someone asked about the possibility of providing hints to the decoder to avoid interleaving pages of different qualities.

Extending the Lifetime of Flash-Based Storage Through Reducing Write Amplification from File Systems

Youyou Lu, Jiwu Shu, and Weimin Zheng, Tsinghua University

Youyou Lu explained how the increased density of flash memory has also made it less tolerant to leakage and noise interference, taking a toll on the reliability and lifetime of flash memory. He also pointed out that traditional file systems were developed assuming the use of hard disks and not flash, the reason for which common mechanisms like journaling, metadata synchronization, and page-aligned update can induce extra write operations that further reduce the lifetime of flash. As a solution, Lu proposed an object-based flash translation layer design (OFTL) that makes file systems no longer responsible for storage management and exports a byte-unit access interface to them. This decoupling allows the OFTL to lazily update metadata indexes and eliminates journals without losing any consistency properties by making use of the page metadata area. In addition, OFTL makes it possible for coarse-grained block state maintenance to reduce free management overheads using units of erase blocks rather than file system blocks. Finally, the byte-unit interface allows OFTL to compact and better co-locate small updates, reducing the total number of updates and amortizing the cost of page writes across unaligned pages.

The system was evaluated with several workloads and traces and implemented as a Linux kernel module. For every workload, Lu et al. measured the write amplifications—defined as the total size or number of writes to the flash memory divided by the total size or number of writes issued from the application layer—across several file systems, including ext2, ext3, Btrfs and their OFTL implementation. The results showed that the OFTL-based system offers a write amplification reduction of 47% to 90% with synchronous workloads and 20 % to 64% with asynchronous workloads.

3for only one question. Richard Spillane (Apple) asked why sequential workloads were causing so much write amplification in one of the experiments. Lu explained that data is duplicated, once for the journal and again for the actual write.

Understanding the Robustness of SSDs under Power Fault

Mai Zheng, The Ohio State University; Joseph Tucek, HP Labs; Feng Qin, The Ohio State University; Mark Lillibridge, HP Labs

Mai Zheng started by mentioning the wide adoption of SSDs due to their many good qualities. However, very little has been said about the behavior of SSDs under adverse conditions. In particular, Zheng studied the behavior of SSDs under power failures. Among the potential types of failures a SSD can experience, he listed bit corruption, metadata corruption, inconsistency in the internal state of the device, shorn and flying writes, and unserializable writes.

To test the resilience of SSDs under power failures, Zheng et al. created a testing framework made of four main components: scheduler, workers, switcher, and checker. Each test cycle consisted of three stages. Initially, the workers stress the SSD with many write operations, ideally making the device as vulnerable as possible by triggering wear leveling and garbage collection. Next, the device's power is cut off asynchronously by a circuit controlling the SSD's dedicated power source. Finally, the checker reads back the data and checks for potential failures. The data written to the device is carefully arranged and contains enough metadata to uniquely identify all types of errors and other possible interferences such as dynamic data compression by the SSD.

For the evaluation, 15 SSDs were subjected to a long series of controlled power failures, and 13 of them exhibited some form of failure. While unserializable writes were by far the most common type of failure, all other types of failures were found as well. One device exhibited metadata corruption after only eight injected faults which caused 72 GB of data to be permanently lost. Other devices were rendered undetectable by the host after 136 injected faults.

Zheng was asked what types of error he found when the devices were not stressed, but that was not considered in the evaluation. John Badger (Quantum) asked about the devices

that claimed to have some form of power failure protection and whether those also failed. Zheng replied that three out of four did. Fred Glover (Oracle) asked whether they tried cutting the AC power supply instead of DC. Zheng said they didn't, as it's not easy to automate and perform many power cycles by cutting the AC. Bill Bilofsky (Microsoft) asked whether they tried cutting off power for a short period of time and whether the error happened during powering off or powering up the device. Zheng said that a quick power restore was not part of the evaluation and the experimental setup did not provide enough insight to determine exactly when failures took place.

Performance Improvements and Measurements

Summarized by Dorian Perkins (dperkins@cs.ucr.edu)

Gecko: Contention-Oblivious Disk Arrays for Cloud Storage

Ji-Yong Shin, Cornell University; Mahesh Balakrishnan, Microsoft Research; Tudor Marian, Google; Hakim Weatherspoon, Cornell University

Cloud infrastructure largely takes advantage of virtual machines (VMs), which export virtual interfaces to the resources on a machine. Because each VM has its own virtual resources, it is agnostic to the I/O scheduling of other VMs. Thus, we see in practice that multiple VMs each sequentially accessing a given disk is actually random access at the storage layer. Ji-Yong Shin presented Gecko, a contention-oblivious storage solution which leverages a chain-logging design to eliminate most of the contention caused by simultaneous access to a shared disk. Gecko builds upon the log file system (LFS) design, which still suffers from shared access I/O contention and poor performance during garbage collection (GC). To address the shortcomings of LFS, Gecko identifies the sources of this I/O contention, including the three causes of disk seeks—write-write, read-read, and write-read operations—and the two causes of GC contention, write-GC and read-GC read operations.

Ji-Yong noted that the philosophy behind Gecko is that "a single sequentially accessed disk is better than multiple randomly seeking disks," which is apparent in its chain-logging design. In Gecko, data is written in log format to a single disk at the tail of the log, which is logically separate from the body. Writing to only the tail disk eliminates the write-write and write-GC read contention, and reduces write-read contention. Ji-Yong explained they use various layers of caching to reduce contention even further. To eliminate read contention to the tail of the log, they cache hot data in-memory, and additionally cache warm data on SSD. Another small flash cache is added to the body of the log to minimize read contention. Ji-Yong showed that Gecko can be mirrored or striped for fault-tolerance and for additional read performance. When in mirrored format, the mirrored body disks can be turned off for power savings. For record keeping, they maintain two mappings: logical to physical primary in-memory mapping, and physical to logical inverse mapping persists on flash storage. Ji-Yong noted that for 8 TB of storage, each of these maps takes less than 8 GB.

For their evaluation, Ji-Yong showed measurements for both synthetic and real-world workloads obtained from Microsoft and MSR. Gecko's aggregate throughput is up to 3x higher than Log + RAID0 with synthetic workloads, and also up to 3x higher than Log + RAID10 with real workloads. They also show how Gecko can achieve higher cache hit rates, less read-write contention, and increased SSD lifetime.

Lakshmi Bairavasundaram (NetApp) asked whether the authors looked into RAID5 or RAID6, in addition to RAID0 and RAID10, since they would end up with too many small writes if they were not appropriately using the log portion. Ji-Yong responded they had not. Daniel Peek (Facebook) asked about the lifetime and replacement cycle of SSD cache. Ji-Yong responded that they are using only 32 GB of SSD for caching and if they use 128 GB SSDs, the replacement cycle from wear-out will be extended by 4x. An attendee from IBM asked if GC caused segmentation in the journal after numerous overwrites. Ji-Yong answered that they have not thoroughly tested GC but they did test whether the journal is fragmented and needed to be compacted or relocated, and they achieved similar performance to their presented GC collection results. Another attendee from EMC asked about the performance under read-read contention during GC, which may happen during first write. Ji-Yong mentioned that this remains future work but noted that they do get better performance than RAID0 by adding a body cache to reduce read-GC read contention.

Screaming Fast Galois Field Arithmetic Using Intel SIMD Instructions

James S. Plank, University of Tennessee; Kevin M. Greenan, EMC Backup Recovery Systems Division; Ethan L. Miller, University of California, Santa Cruz

James S. (Jim) Plank gave a high-energy talk about his work with optimizing Galois Field arithmetic (GF) by harnessing the power of the Intel SIMD instruction set. Jim began by noting the prevalence of erasure coding in today's systems. Traditionally, Reed-Solomon codes are used in erasure codes, but they are much slower than XOR operation. This is inconvenient because Reed-Solomon codes are powerful, general, and flexible. However, their inefficient performance has led to the development of a huge number of XOR codes. Jim noted that this talk details the "secret handshake" he has sought to understand about how to speed up GF arithmetic. The secret is harnessing the power of specific instructions in the Intel SSE3 SIMD instruction set to perform GF arithmetic fast

enough such that it is only limited by the L2/L3 cache line speed, a 2.7x to 12x improvement on XOR codes!

Storage systems use GF arithmetic for erasure codes, which are structured as linear combinations of w-bit data words in a GF. W is the number of bits in the erasure-coding word, and as W increases you trade increased robustness for slower speeds. Suppose you want to take a constant and perform 1024 multiplications (doing fewer is the secret). Most Intel architectures provide special instructions that can now use 128-bit words, allowing for 128-bit XOR or AND operations, or two 64-bit left shifts. The important instruction here is the mm_shuffle_epi8() instruction. This allows us to do 16 simultaneous table lookups in one operation of a 16-byte table with 16 4-bit indices. You can then use precomputed tables and generate a mask to do a 16-bit table lookup, e.g., 16 simultaneous multiplications. Then, using the distributive law of multiplication, the authors split words into smaller components and multiply using another left-shifted table to further optimize the problem. Jim evaluated performance by varying the buffer size from 1 KB to 1 GB with values of W = 4, 8, 16, 32, and baselines which show the L2/L3 cache effects. Jim noted that using the techniques they developed, the lines for W = 4, 8, and 16 are now cache limited. Overall, their evaluation shows a time improvement of up to 12x compared to XOR methods. Their open-source GF arithmetic library, GF-Complete (written in C), is available for use now via the author's website: http://web.eecs.utk.edu/~plank/plank/www/software.html.

An attendee suggested running the evaluation benchmarks on a server class processor, since the memory interface is much more powerful than a desktop processor. Another attendee asked whether ARM servers have a comparable instruction set. Jim responded they had not yet checked to see whether it was exactly the same. An attendee mentioned that from a pure hardware point-of-view, performing GF is as easy as any other multiplication.

Virtual Machine Workloads: The Case for New NAS Benchmarks

Vasily Tarasov, Stony Brook University; Dean Hildebrand, IBM Research, Almaden; Geoff Kuenning, Harvey Mudd College; Erez Zadok, Stony Brook University

Customers consider a number of parameters, including performance, when choosing network attached storage (NAS). In reality, many customers end up disappointed because benchmarks often promise better performance than NAS delivers. Vasily Tarasov argued that this discrepancy stems from the fact that, in practice, many clients use VMs while performance benchmarks do not. The goal of this work is to create benchmarks better suited for the real-world VM environments. Vasily's work focuses on a VDI (virtual disk images) on NAS approach to solving this issue. He noted that 90% of the shipped storage capacity is for NAS, not direct attached storage (DAS), and the growth rate for NAS storage is 60% a year. Additionally, virtualization is very dominant in the market, as 70% of systems run virtual environments. The inherent problem with benchmarking NAS systems serving VMs is that the I/O stack is deep, and requests change significantly by the time they hit storage (e.g., request reordering, merging and trimming, I/O contention, etc.). Vasily aims to provide a new benchmarking tool for VM-NAS environments, as using old benchmarks is cumbersome, inflexible, and sometimes prohibitively expensive.

The benchmark environment used in this work runs Filebench and Jetstress on Linux/Windows using ext3/NTFS atop the VMware ESXi 5 hypervisor with a black box NAS appliance (running GPFS on Linux). Vasily then characterized the workload they found in their evaluation system. Compared to physical non-VM systems, Vasily noted that when making observations about the workloads seen at the NAS appliance, VM systems have almost no metadata operations because they are handled in the VM layer, all writes are synchronous, in-VDI file randomness is increased, I/O size changes depending on how VM layers handle requests, and lastly, the read-modify-write operation is more frequent. They considered the following workload features: read/write ratio, I/O size distribution, jump distance distribution (LBA distance), and offset reuse. Vasily briefly described how they created their benchmarks using their T2M converter tool, also showing that any mix of VM combinations can be used, and directed those who seek more details to review the paper. Their evaluation monitors 11 parameters, and they measured accuracy compared to physical benchmark as they increased the number of VMs in their benchmark. They showed that they maintain an 8% error rate, which grows slowly with the number of VMs. They left for future work the exploration of other storage stack configurations, targeting VM-specific workloads and emulation of I/O request transformations. Vasily noted that their benchmark tool can be found by searching "t2mpublic" or via the URL in the paper.

An attendee from Symantec asked whether this benchmark could be used for SAN. Vasily replied that the same technique can be applied for SAN, but the benchmarks cannot be directly applied unless you have a file system interface. Kiran-Kumar Muniswamy-Reddy (Amazon), noted they are using the CFQ scheduler, and if the workload is random, they may want to use a different scheduler; but would doing so change their system? Vasily mentioned this as future work; the I/O scheduler definitely affects the workload, and they want to explore more configurations to see which parameters affect workload and which are less influential.