REPORTS

Conference Reports

In this issue:

FAST ’14: 12th USENIX Conference on File and Storage Technologies
February 17–20, 2014 , San Jose, CA

Opening Remarks
Summarized by Rik Farrow

Bianca Schroeder (University of Toronto) opened this year’s USENIX Conference on File and Storage Technologies (FAST ’14) by telling us that we represented a record number of attendees for FAST. Additionally, 133 papers were submitted, with 24 accepted. That’s also near the record number of submissions, 137, which was set in 2012. The acceptance rate was 18%, with 12 academic, three industry, and nine collaborations in the author lists. The 28 PC members together completed 500 reviews, and most visited Toronto in December for the PC meeting.

Eno Thereska (Microsoft Research), the conference co-chair, then announced that “Log-Structured Memory for DRAM-Based Storage,” by Stephen M. Rumble, Ankita Kejriwal, and John Ousterhout (Stanford University) had won the Best Paper award, and that Jiri Schindler (Simplivity) and Erez Zadok (Stony Brook University) would be the co-chairs of FAST ’15.

Big Memory
Summarized by Michelle Mazurek (mmazurek@cmu.edu)

Log-Structured Memory for DRAM-Based Storage
Stephen M. Rumble, Ankita Kejriwal, and John Ousterhout, Stanford University

Awarded Best Paper!

The primary author, Stephen Rumble, was not available, so John Ousterhout presented the paper, which argued that DRAM storage systems should be log-structured, as in their previous work on RAMCloud. When building a log-structured DRAM system, an important question is how memory is allocated. Because DRAM is (relatively) expensive, high memory utilization is an important goal. Traditional operating-system allocators are non-copying; data cannot be moved after it is allocated. This results in high fragmentation and, therefore, low utilization. Instead, the authors consider a model based on garbage collection, which can consolidate memory and improve utilization. Existing garbage collectors, however, are expensive and scale poorly. They wait until a lot of free space is available (to amortize cleaning costs), which can require up to 5x over-utilization of memory. When the garbage collector does run, it can consume up to three seconds, which is slower than just resetting the system and rebuilding the RAM store from the backup log on disk.

The authors develop a new cleaning approach that avoids these problems. Because pointers in a file system are well-controlled, centrally stored, and have no circularities, it is possible to clean and copy incrementally (which would not work for a more general-purpose garbage-collection system). In the authors’ approach, the cleaner continuously finds and cleans some segments with significant free space, reducing cleaning cost and improving utilization. Further, the authors distinguish between the main log, kept in expensive DRAM with high bandwidth (targeted at 90% utilization), and the backup log, stored on disk where capacity is cheap but bandwidth is lower (targeted at 50% utilization). They use a two-level approach in which one cleaner (“compaction”) incrementally cleans one segment at a time in memory, while a second one (“combined cleaning”) less frequently cleans across segments in both memory and disk. Both cleaners run in parallel to normal operations, with limited synchronization points to avoid interference with new writes. The authors’ evaluation demonstrates that their new approach can achieve 80–90% utilization with performance degradation of only 15–20% and negligible latency overhead.

Bill Bolosky (Microsoft Research) asked how single segments cleaned via compaction can be reused before combined cleaning has occurred. Ousterhout explained that compaction creates “seglets” that can be combined into new fixed-sized chunks and allocated. A second attendee asked when cleaning occurs. Ousterhout replied that waiting as long as possible allows more space to be reclaimed at each cleaning to better amortize costs.

Strata: High-Performance Scalable Storage on Virtualized Non-Volatile Memory
Brendan Cully, Jake Wires, Dutch Meyer, Kevin Jamieson, Keir Fraser, Tim Deegan, Daniel Stodden, Geoffrey Lefebvre, Daniel Ferstay, and Andrew Warfield, Coho Data

Brendan Cully discussed the authors’ work developing an enterprise storage system that can take full advantage of fast non-volatile memory while supporting existing storage array customers who want to maintain legacy protocols (in this case, NFSv3). A key constraint is that, because flash gets consistently cheaper, enterprise customers want to wait until the last possible moment to purchase more of it, requiring the ability to add flash dynamically. The authors’ solution has three key pieces.

- Device virtualization: clients receive a virtual object name for addressing while, underneath, an rsync interface manages replication.
- Data path abstractions: for load balancing and replication, keep object descriptions in a shared database that governs paths and allows additional capacity and rebalancing, even when objects are in use.
- Protocol virtualization targeting NFSv3: the authors use a software-defined switch to control connections between clients and servers. Packets are dispatched based on their contents, allowing rebalancing without changes to the client. All nodes have the same IP and MAC address, so they appear to the client to be one node.

To evaluate the system, the authors set up a test lab that can scale from 2 to 24 nodes (the capacity of the switch) and rebalance dynamically. As nodes are added, momentary drops in IOPS are observed because nodes must both serve clients and support rebalancing operations. Overall, however, performance increases, but not linearly; the deviation from linear comes because, as more nodes are added, the probability of nodes doing remote I/O on behalf of clients (slower than local I/O) increases. This effect can be mitigated by controlling how clients and objects are assigned to promote locality; with this approach, the scaling is much closer to linear. CPU usage for this system is very high, so introduction of 10-core machines improves IOPS significantly.

An attendee asked whether requests can always be mapped to a node that holds an object, avoiding remote I/O. Cully responded that clients can't be moved on a per-request basis, and they will ask for objects that may live on different nodes. Niraj Tolia of Maginatics asked how the system deals with multiple writers. Cully responded that currently only one client can open the file for writing at a time, which prevents write conflicts but requires consecutive writers to wait; multiple readers are allowed.

Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches

Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chiu, IBM Almaden Research Center

Hyojun Kim presented a measurement study of a prototype SSD with 45 nm 1 GB phase-change memory. PCM melts and cools material to store bits: amorphous = reset, crystalline = set. Writing set bits takes longer. Write latency is typically reported at 150 ns, which is more than 1000x faster than flash and only 3x slower than DRAM, but which does not count 50 ns of additional circuit time. Write throughput is limited (4 bits per pulse) by the chip's power budget for heating the material. Kim directly compared PCM with flash, normalized for throughput, using Red Hat, a workload generator, and a statistics collector. They found that for read latency, PCM is 16x faster on average and also has much faster maximums. For write latency, however, PCM is 3.4x slower on average, with a maximum latency of 378 ms compared to 17.2 ms for flash.

The second half of the talk described how simulation was applied to assess how and whether PCM is useful for enterprise storage systems. First, the authors simulated a multi-tiered system that writes hot data to flash or PCM and cold data to a cheap hard drive, incorporating the following relative price assumptions: PCM = 24, flash = 6, disk = 1 per unit of storage. They evaluated a variety of system combinations using x% PCM, y% flash, and z% disk, based on a one-week trace from a retail store in June 2012, and measured the resulting performance in IOPS/unit cost. Using an ideal static placement based on knowing the workload traces a priori, the optimal combination was 30% PCM, 67% flash, and 3% disk. With reactive data movement based on I/O traffic (a more realistic option), the ideal combination was 22% PCM, 78% flash. A second simulation used flash or PCM as application-server-side, write-through, and LRU caching, using a 24-hour trace from customer production systems (manufacturing, media, and medical companies). The results measured average read latency. To include cost in this simulation, different combinations of flash and PCM with the same IOPS/cost were simulated. For manufacturing, the best results at three cost points were 64 GB of flash alone, 128 GB of flash alone, and 32 GB PCM + 128 GB flash. In summary, PCM has promise for storage when used correctly, but it's important to choose accurate real-world performance numbers.

One attendee asked whether the measurements had considered endurance (e.g., flash wearout) concerns as well as IOPS. Kim agreed that it's an important consideration but not one that was measured in this work. Someone from UCSD asked whether write performance for PCM is unfairly disadvantaged unless capacity-per-physical-size issues are also considered. Kim agreed that this could be an important and difficult tradeoff. A third attendee asked about including DRAM write buffers in PCM, as is done with flash. Kim agreed that the distinction is an important one and can be considered unfair to PCM, but that the current work attempted to measure what is currently available with PCM—write buffers may be available in the future. A final attendee asked for clarification about the IOPS/unit cost metric; Kim explained that it's a normalized relative metric alternative to capturing cost explicitly in dollars.

Flash and SSDs

Summarized by Jeremy C. W. Chan (cwchan@cse.cuhk.edu.hk)

Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance

Xavier Jimenez, David Novo, and Paolo Ienne, Ecole Polytechnique Fédérale de Lausanne (EPFL)

Xavier Jimenez presented a technique to extend the lifetime of NAND flash. The idea is based on the observation that inside a block of a NAND flash, some pages wear out faster than others. As a result, the endurance of a block is determined by the weakest page.

To improve the lifetime, Xavier and his team introduced a fourth page state, “relieved,” to indicate pages not to be programmed during a Program/Erase (P/E) cycle. By measuring the bit error rate (BER), they showed that relieved pages possess higher endurance than unrelieved ones. Xavier continued about how relief can be performed in the case of a multi-level cell (MLC). In MLC, each one of the two bits is mapped to a different page, forming an LSB and MSB page pair. A full relief means both pages are skipped in a P/E cycle, while a half relief means only the MSB is relieved. Although a half relief produces higher relative wear, it is more effective in terms of written bits per cycle.

Xavier then described two strategies on how to identify weak pages. The simple strategy is reactive, which starts relieving a page when the BER reaches a certain threshold. However, the reactive approach requires the FTL to read a whole block before erasing it, which adds an overhead to the erasing time. Therefore, the authors further proposed the proactive strategy, which predicts the number of times weak pages should be relieved to match the weakest page's extended endurance by first characterizing the endurance of the LSB/MSB pages at every position of the block. The proactive strategy is used together with adaptive planning, which predefines a number of lookup tables called plans. The plan provides the probability for each page to be relieved.

In the evaluation, Xavier and his team implemented the proactive strategy on two previously proposed FTLs, ROSE and ComboFTL, and on two kinds of 30 nm class chips, C1 and C2. C1 is an ABL chip with less interference, whereas C2 is an interleaved chip, which is faster and more flexible. The evaluation on real-world traces shows that the proactive strategy improves lifetime by 3–6% on C1 and 44–48% on C2. Only a small difference is observed in execution time because of the efficient half relief operation.

Alireza Haghdoost (University of Minnesota) asked whether Xavier's approach is applicable to a block-level FTL. Xavier explained that block-level relieving is impractical and that the evaluation is entirely based on page-level mapping. Steve Swanson (UCSD) asked if page skipping would affect the performance of reading the neighboring pages. Xavier said the endurance of the neighboring pages will be decreased by at most 2%, and because they are the stronger pages, the impact is minimal on the block's lifetime.

Lifetime Improvement of NAND Flash-Based Storage Systems Using Dynamic Program and Erase Scaling

Jaeyong Jeong and Sangwook Shane Hahn, Seoul National University; Sungjin Lee, MIT/CSAIL; Jihong Kim, Seoul National University

Jaeyong Jeong presented a system-level approach called dynamic program and erase scaling (DPES) to improve the lifetime of NAND flash-based storage systems. The approach exploits the fact that the erasure voltage and the erase time affect the endurance of NAND flash memory.

Jaeyong began the presentation with an analogy illustrated by interesting cartoons. In the analogy, NAND flash is a sheet of paper, the program action is writing on the paper with a pencil, and the erase action is using an eraser to clear out the word for the whole page. Finally, the flash translation layer (FTL) is a person called Flashman. With this analogy, Jaeyong explained why NAND endurance had decreased by 35% during the past two years despite the 100% increase in capacity. He said that advanced semiconductor technology is just like a thinner piece of paper, which wears down more easily than a thick piece of paper after a certain number of erasure cycles. However, low erase voltage and long erase time are the two main keys to improving the endurance of NAND flash.

The fundamental tradeoff between erase voltage and program time is that the lower the erase voltage, the longer the program time required. With this observation, Jaeyong and his team proposed the DPES approach, which dynamically changes the program and erase voltage/time to improve the NAND endurance while minimizing negative impact on throughput.

They implemented their idea on an FTL and called it AutoFTL. It consists of a DPES manager, which selects the program time, erase speed, and erase voltage according to the utilization of an internal circular buffer. For instance, a fast write mode is selected to free up buffer space when its utilization is high. In the evaluation, Jaeyong and his team chose six volumes with different inter-arrival times from the MSR Cambridge traces. On average, AutoFTL achieves a 69% gain on the endurance of the NAND flash with only negligible impact (2.2%) on the overall write throughput.

Geoff Kuenning (Harvey Mudd College) asked why high voltage causes electrons to get trapped in the oxide layer. Jaeyong reemphasized that the depletion of the tunnel side has an exponential relationship to the erase voltage and time. Yitzhak Birk (Technion) said that the approach of programming in small steps works but may bring adverse effects to the neighboring cells. Peter Desnoyers (Northeastern University) asked how they manage to select the appropriate reading method according to the voltage level applied. Jaeyong replied that the lookup table would be able to track the voltage level. Peter followed up that a scan for pages is not possible because you cannot read a page before knowing the voltage level.

ReconFS: A Reconstructable File System on Flash Storage

Youyou Lu, Jiwu Shu, and Wei Wang, Tsinghua University

Youyou Lu began with the novelty of ReconFS in metadata management of hierarchical file systems. This work addresses a major challenge in namespace management of file systems on solid-state drives (SSDs), which are the scattered small updates and intensive writeback required to maintain a hierarchical namespace with consistency and persistence. These writes cause write amplification that seriously hurts the lifetime of SSDs. Based on the observation that modern SSDs have high read bandwidth and IOPS and that the page out-of-band (OOB) area provides some extra space for page management, Youyou and his team built the ReconFS, which decouples maintenance of volatile and persistent directory trees to mitigate the overhead caused by scattered metadata writes.

Youyou presented the core design of ReconFS on metadata management. In ReconFS, a volatile directory tree exists in memory, which provides hierarchical namespace access, and a persistent directory tree exists on disk, which allows reconstruction after system crashes. The four triggering conditions for namespace metadata writeback are: (1) cache eviction, (2) checkpoint, (3) consistency preservation, and (4) persistence maintenance. Although a simple home-location update is used for cache eviction and checkpoint, ReconFS proposes an embedded inverted index for consistency preservation and metadata persistence logging for persistence maintenance. Inverted indexing in ReconFS is placed in the log record and the page OOB depending on the type of link. The key objective of inverted indexing is to make the data pages self-described. Meanwhile, ReconFS writes back changes to directory tree content to a log to allow recovery of a directory tree after system crashes. Together with unindexed zone tracking, ReconFS is able to reconstruct the volatile directory tree in both normal and unexpected failures.

To evaluate the ReconFS prototype, Youyou and his team implemented a prototype based on ext2 on Linux. Using filebench to simulate a metadata-intensive workload, they showed that ReconFS achieves nearly the best throughput among all evaluated file systems. In particular, ReconFS improves performance by up to 46.3% in the varmail workload. Also, the embedded inverted index and metadata persistence logging enabled ReconFS to give a write reduction of 27.1% compared to ext2.

Questions were taken offline because of session time constraints.

Conference Luncheon and Awards

Summarized by Rik Farrow

During the conference luncheon, two awards were announced. The first was the FAST Test of Time award, for work that appeared at a FAST conference and continues to have a lasting impact. Nimrod Megiddo and Dharmendra S. Modha of IBM Almaden Research Center won this year's award for "ARC: A Self-Tuning, Low Overhead Replacement Cache" (https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache).

The IEEE Reynolds and Johnson award went to John Ousterhout and Mendel Rosenblum, both of Stanford University, for their paper "The Design and Implementation of a Log-Structured File System" (http://www.stanford.edu/~ouster/cgi-bin/papers/lfs.pdf). Rosenblum commented that he was Ousterhout's student at UC Berkeley when the paper was written. Later, Rosenblum wound up working with Ousterhout at Stanford.

Personal and Mobile

Summarized by Kuei Sun (kuei.sun@utoronto.ca)

Toward Strong, Usable Access Control for Shared Distributed Data

Michelle L. Mazurek, Yuan Liang, William Melicher, Manya Sleeper, Lujo Bauer, Gregory R. Ganger, and Nitin Gupta, Carnegie Mellon University; Michael K. Reiter, University of North Carolina

Michelle Mazurek began her presentation by showing us recent events where improper access control led to mayhem and privacy invasions. The main issue is that access control is difficult, especially for non-expert users. In their previous work, the authors identified users' need for flexible policy primitives, principled security, and semantic policies (e.g., tags). To this end, they based the design of their system on two important concepts: tags, which allow users to group contents, and logical proof, which allows for fine-grained control and flexible policy. For every content access, a series of challenges and proofs needs to be made before access is granted. On each device participating in the system, a reference monitor exists to protect the content that the device owns, a device agent that performs remote proofs for enabling content transfer across the network, as well as user agents that construct proofs on behalf of the users. Michelle walked us through an example of how Bob could remotely access a photo of Alice on a remote device in this system. Michelle then described the authors' design of strong tags. Tags are first-class objects, such that access to them is independent of content access. To prevent forging, tags are cryptographically signed.

In their implementation, the authors mapped system calls to challenges. They cached recently granted permissions so that the same proof would not need to be made twice. In their evaluation, they wrote detailed policies drawn from user studies using their policy languages, all of which could be encoded in their implementation and showed that their logic had sufficient expressiveness to meet user needs. They simulated access patterns because they do not have a user study based on the perspective of the user accessing content. They ran two sets of experimental setups on their prototype system: one with a default-share user and the other with a default-protect user. The main objective of the experiments was to measure latency for system calls and see whether they were low enough for interactive users. The results showed that with the exception of readdir(), system calls fell well below the 100 ms limit that they set. The authors also showed that access control only accounted for about 5% of the total overhead. Finally, it took approximately 9 ms to show that no proof could be made (access denied!), although variance in this case can be quite high.

Tiratat Patana-anake (University of Chicago) wanted clarification on the tags. Michelle explained that you only need access to the tags required for access to the file. Someone from University of California, Santa Cruz, wanted to know which cryptographic algorithm was used, what its overhead was, and which distributed file system was used. Michelle said the proofer uses the crypto library from Java and that it doesn't add too much overhead. The distributed file system was homemade. Another person from UCSC asked whether the authors intended to replace POSIX permission standards, and the response was yes. He went on to ask whether they have reasonable defaults because most users are lazy. Michelle first explained that their user study indicates a broad disagreement among users on what they want from such a system. However, she agreed that users are generally lazy so more research into automated tagging and a better user interface would be helpful. Finally, someone asked about transitivity, where one person would take restricted content and give it to an unauthorized user. Michelle believed that there was no solution to the problem where the person to whom you grant access is not trustworthy.

On the Energy Overhead of Mobile Storage Systems

Jing Li, University of California, San Diego; Anirudh Badam and Ranveer Chandra, Microsoft Research; Steven Swanson, University of California, San Diego; Bruce Worthington and Qi Zhang, Microsoft

Jing Li began the talk by arguing that in spite of the low energy overhead of storage devices, the overhead of the full storage stack on mobile devices is actually enormous. He presented the authors' analysis of the energy consumption used by components of the storage stack on two mobile devices: an Android phone and a Windows tablet. After giving the details of the experimental setup, he showed the microbenchmark results, which revealed that the energy overhead of storage stack is 100 to 1000x higher than the energy consumed by the storage device alone. He then focused on the CPU's busy time, which showed that 42.1% of the busy time is spent in encryption APIs while another 25.8% is spent in VM-related APIs.

Li and his team first investigated the true cost of data encryption by comparing energy consumption between devices with and without encryption. Surprisingly, having encryption costs on average 2.5x more energy. Next, Li gave a short review of the benefits of isolation between applications and of using managed languages. He then showed the energy overhead for using managed languages, which is anywhere between 12.6% and 102.1% (for Dalvik on Android!). Li ended his talk with some suggestions. First, storage virtualization can be moved into the storage hardware. Second, some files, such as the OS library, do not need to be encrypted. Therefore, a partially encrypted file system would help reduce the energy overhead. Finally, hardware-based solutions (e.g., DVFS or ASIC) can be used to support encryption or hardware virtualization while keeping energy cost low.

Yonge (University of California, Santa Cruz) asked how the energy impact of DRAM was measured. Li said that they ran the benchmark to collect the I/O trace. They then replayed the I/O trace to obtain the energy overhead of the storage stack. As such, the idle power was absent from the obtained results.

ViewBox: Integrating Local File Systems with Cloud Storage Services

Yupu Zhang, University of Wisconsin–Madison; Chris Dragga, University of Wisconsin–Madison and NetApp; Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin–Madison

Yupu Zhang started by reminding us that cloud storage services are gaining popularity because of promising benefits such as reliability, automated synchronization, and ease of access. However, these systems can fail to keep data safe when local corruption or a crash arises, which causes bad data to be propagated to the cloud. Furthermore, because files are uploaded out of order, the cloud may sometimes have an inconsistent set of data. To provide a strong guarantee that the local file system's state is equal to the cloud's state, and that both states are correct, the authors developed ViewBox, which integrates the file system with the cloud storage service. ViewBox employs checksums to detect problems with local data, and utilizes cloud data to recover from local failures. ViewBox also keeps in-memory snapshots of valid file system state to ensure consistency between the local file system and the cloud.

Yupu presented the results of detailed experimentation, which revealed the shortcomings of the current setup. In their first experiment, the authors corrupted data beneath the file system to see whether it was propagated to the cloud. ZFS detected all corruptions because it performed data checksumming, but services running on top of ext4 propagated all corruptions to the cloud. In the second experiment, they emulated a crash during synchronization. Their results showed that without enabling data journaling on the local file system, the synchronization services would behave mostly erratically. All three services that the authors tested violated causal ordering that is locally enforced by fsync().

Next, Yupu gave an overview of the architecture. They modified ext4 to add checksums. Upon detecting corrupt data, ext4-cksum can communicate with a user daemon named Cloud Helper via ioctl() to fetch correct data from the cloud. After a crash, the user is given the choice of either recovering inconsistent files individually or rolling back the entire file system to the last synchronized view. The View Manager, on the other hand, creates consistent file system views and uploads them to the cloud. To provide consistency efficiently, they implemented two features: cloud journaling and incremental snapshotting. The basic concept of cloud journaling is to treat the cloud as an external journal by synchronizing local changes to the cloud at file system epochs. View Manager would continuously upload the last file system snapshot in memory to the cloud. Upon failure, it would roll the file system back to the latest synchronized view. Incremental snapshotting allows for efficient freezing of the current view by logging namespace and data changes in memory. When the file system reaches the next epoch, it will update the previous frozen view without having to interrupt the next active view.

In their evaluation, the authors showed that ViewBox could correctly handle all the types of error mentioned earlier. ViewBox has a runtime overhead of less than 5% and a memory overhead of less than 20 MB, whereas the workload contains more than 1 GB of data.

Someone asked how much they roll back after detecting corruption. Yupu responded that if you care about the whole file system, ideally, you roll back the whole file system, but generally you just roll back individual files. The same person asked what happens if an application is operating on a local copy that gets rolled back. Yupu said that the application would have to be aware of the rollback. Mark Lillibridge (HP) suggested that a correctly written application should make a copy, change that copy, and then rename the copy to avoid problems like this. Yupu agreed.

RAID and Erasure Codes

Summarized by Yue Cheng (yuec@vt.edu)

CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization

Alberto Miranda, Barcelona Supercomputing Center (BSC-CNS); Toni Cortes, Barcelona Supercomputing Center (BSC-CNS) and Technical University of Catalonia (UPC)

Alberto Miranda presented an economical yet effective approach for performing RAID upgrading. The authors' work is motivated by the fact that storage rebalancing caused by degraded uniformity is way too expensive. Miranda proposed CRAID, a data distribution strategy that redistributes only hot data when performing RAID upgrading by using a dedicated small partition in each device as a persistent disk-based cache.

Taking advantage of the I/O access pattern monitoring that is used to keep track of data access statistics, an I/O redirector can strategically redistribute/rebalance only frequently used data to the appropriate partitions. The realtime monitoring provided by CRAID guarantees that a newly added disk will be used as long as it is added. Statistics of data access patterns are also used to effectively reduce the cost of data migration. The minor disadvantage of this mechanism is that the invalidation on the cache partition results in the loss of all the previous computation. Trace-based simulations conducted on their prototype showed that CRAID could achieve competitive performance with only 1.28% of all available storage capacity, compared to two alternative approaches.

Someone asked Alberto to comment on one of the workload characteristics that can impact the rebalancing algorithm. Miranda said that the available traces they could find were limited.

STAIR Codes: A General Family of Erasure Codes for Tolerating Device and Sector Failures in Practical Storage Systems

Mingqiang Li and Patrick P. C. Lee, The Chinese University of Hong Kong

Patrick Lee presented STAIR codes, a set of novel erasure codes that can efficiently tolerate failures in both device and sector levels. What motivates STAIR is that traditional RAID and erasure codes use multiple parity disks/parity sectors (the cost of which is prohibitive) to provide either device-level or sector-level tolerance. They proposed a general (without any restriction) and space-efficient family of erasure codes that can tolerate simultaneous device and sector failures.

The key idea of STAIR codes is to base protection against sector failure on a pattern of how sector failures occur, instead of setting a limit on tolerable sector failures. The actual code structure builds based on two encoding phases, each of which builds on any MDS code that works as long as the parameters support the code. The interesting part of their approach is the upstairs and downstairs encoding that can reuse computed parity results, thus providing space efficiency and complementary performance advantages. Evaluation results showed that STAIR codes improve encoding by up to 100% while achieving storage space efficiency. Their open-sourced coding library can be found at: http://ansrlab.cse.cuhk.edu.hk/software/stair.

Someone asked whether the authors managed to measure the overhead (reading the full stripe) of some special (extra) sectors' encoding every time they had to be recreated in a RAID device. Patrick said their solution needed to read the full stripe from the RAID so as to perform the upstairs and downstairs encoding. Umesh Maheshwari (Nimble Storage) asked whether the scenario where errors come in a burst was an assumption or a case that STAIR took care of; his concern was that in SSD the errors might end up in a random distribution. Patrick said the error burst case was an issue they were trying to address.

Parity Logging with Reserved Space: Towards Efficient Updates and Recovery in Erasure-Coded Clustered Storage

Jeremy C. W. Chan, Qian Ding, Patrick P. C. Lee, and Helen H. W. Chan, The Chinese University of Hong Kong

Patrick Lee also presented CodFS, an erasure-coded clustered storage system prototype that can achieve both high update and recovery performance. To reduce storage costs and footprint, enterprise-scale storage clusters now use erasure-coded storage rather than replication mechanisms that incur huge overheads. However, the issues faced by erasure coded storage systems are that updates are too costly and recovery is expensive as well. To deal with this, Patrick and his students propose a parity-logging scheme with reserved space that adopts a hybrid in-place and log-based update mechanism with adaptive space readjustment.

They studied two real-world storage traces and, based on the observed update characteristics, propose a novel delta-based parity logging with reserved space (PLR) mechanism that reduces disk seeks by keeping each parity chunk and its parity delta next to each other with additional space, the capacity of which can be dynamically adjusted. The challenges lie in how much reserved space is most economical and the timing of reclaiming unused reserved space.

In-place updates are basically overwriting existing data and parity chunks while log-based updates are appending changes by converting random writes to sequential writes. Their hybrid approach smartly maintains the advantages of the above two update schemes while mitigating the problems that might occur.

Konstantin Shvachko (WANdisco) asked how restrictions on I/O patterns (i.e., cluster storage systems that only support sequential reads/writes) affect the performance of workloads. Patrick said performance purely depends on workload types, and their work looked in particular at server workload that they attempted to port into cluster storage systems. Brent Welch (Google) mentioned two observations: (1) real-world workloads might consist of lots of big writes due to the fact that there are many big files distributed in storage; (2) an Object Storage Device layer is unaware of data types and is decoupled from the lower layer file system. Patrick agreed with these observations and said (1) people can choose a different coding scheme based on the segments, and (2) the decoupling problem remains an issue left to be explored in future work.

Poster Session I

Summarized by Sonam Mandal (somandal@cs.stonybrook.edu)

In-Stream Big Data Processing Platform for Enterprise Storage Management

Yang Song, Ramani R. Routray, and Sandeep Gopisetty, IBM Research

This poster presents an approach for in-stream processing of Big Data, which is becoming increasingly important because of the information explosion and subsequent escalating demand for storage capacity. The authors use Cassandra as their storage, Hadoop/HDFS for computation, RHadoop as the Machine Learning algorithm, and IBM InfoSphere Streams for their use case example. They implemented many ensemble algorithms like Weighted Linear Regression with LASSO, Weighted Generalized Linear Regression (Poisson), Support Vector Regression, Neural Networks, and Random Forest. They use their technique to identify outliers in 2.2 million backup jobs each day for jobs having 20+ metrics. They try to identify anomalies using the Contextual Local Outlier Factor algorithm before storing on HDFS/Cassandra. To do so, they leverage backup-specific domain knowledge and have shown the results of their outlier detection experiment in the poster.

Page Replacement Algorithm with Lazy Migration for Hybrid PCM and DRAM Memory Architecture

Minho Lee, Dong Hyun Kang, Junghoon Kim, and Young Ik Eom, Sungkyunkwan University

The authors came up with a page replacement algorithm to benefit hybrid memory systems using PCM by reducing the number of write operations to them. PCM is non-volatile and has in-place update and byte-addressable memory with low read latency. PCM suffers from a low endurance problem, where only a million writes are possible before it wears out, and the write latency of PCM is much slower than its read latency. PCM reduces the overhead of migration by using lazy migration.

When a page fault occurs, it is always allocated in DRAM regardless of whether it was a read or write operation; and, when the DRAM fills up, pages in the DRAM are migrated to the PCM. When a write operation occurs on a page in the PCM, it attempts to migrate this page to DRAM. If there is no free space in the PCM, a page is evicted according to rules from the CLOCK algorithm.

The authors' results show that they obtained a high hit ratio regardless of PCM size in hybrid memory architectures. They reduced the number of PCM writes by up to 75% compared to state of the art algorithms and by 40% compared to the CLOCK algorithm.

On the Fly Automated Storage Tiering

Satoshi Iwata, Kazuichi Oe, Takeo Honda, and Motoyuki Kawaba, Fujitsu Laboratories

The authors present issues with existing storage-tiering techniques and propose their own technique to overcome the shortcomings of existing approaches. Storage tiering combines fast SSDs with slow HDDs such that hot data is kept on SSDs and cold data is stored on HDDs. Less frequently accessed data in SSDs is swapped out at predefined intervals to follow changes in workloads. Existing methods have difficulty following workload changes quickly as the migration interval cannot be reduced too far. Shorter intervals lead to lower I/O performance due to more disk bandwidth consumed by migration.

The authors propose an on-the-fly agreed service time (AST) to follow any workload changes within minutes, instead of the granularity of hours or a day with previous methods. Less frequently accessed data is filtered out from migration candidates, thus decreasing bandwidth consumption, even though it results in unoccupied SSD storage space. They use a two-stage migration-filter approach. The first stage filters out hotter but not very hot segments by checking access concentrations. The second filters out segments for which the hot duration is not long enough.

When 60% of the total I/O is sent to fewer than 20 segments (approximately 10% of data size), then these segments are marked as candidates. When a segment has been marked as a candidate three times in a row, it is migrated to the SSD. The authors' evaluation results back their claims and show that workload changes can be followed in a matter of minutes using their approach.

SSD-Tailor: Customization System for Enterprise SSDs

Hyunchan Park, Youngpil Kim, Cheol-Ho Hong, and Chuck Yoo, Korea University; Hanchan Jo, Samsung Electronics

This poster presents SSD-Tailor, a customization system for SSDs. With the increasing need to satisfy customers for requirements of high performance and reliability for various workloads, it becomes difficult to design an optimal system with such a large number of potential configuration choices. SSD-Tailor determines a near-optimal design for a particular workload of an enterprise server. It requires three inputs: customer requirements, workload traces, and design options. It has three components: Design Space Explorer, Trace-driven SSD Simulator, and Fitness Analyzer, which work together iteratively in a loop to produce a near-optimal design for the given requirements and workload traces.

Design options consist of a type of flash chip, FTL policies, etc. Customer requirements may include low cost, high performance, high reliability, low energy consumption, and so on. Workload traces are full traces rather than extracted profiles.

The Design Space Explorer used genetic algorithms to find near-optimal SSD design. Genetic algorithms mimic the process of natural selection. The Trace-driven SSD Simulator (the authors used DiskSim) can help change and display the design options easily, because in spite of a reduced set of design options, too many still need to be analyzed. The Fitness Analyzer evaluates the simulation results based on scores.

The authors show that tailoring overhead is high but needs to be done only once. The benefits obtained are quite high according to their results. They compared SSD-Tailor with a brute force algorithm as their baseline.

Multi-Modal Content Defined Chunking for Data Deduplication

Jiansheng Wei, Junhua Zhu, and Yong Li, Huawei

The authors of this paper have come up with a deduplication mechanism based on file sizes and compressibility information. They identified that many file types such as mp3 and jpeg are large, hardly modified, and often replicated as is; such files should have large chunk sizes to reduce metadata volume without sacrificing too much in deduplication ratio. Other file types consist of highly compressible files, some of which are modified frequently, and these benefit from having small chunk sizes to maximize deduplication ratio.

The authors propose two methods, both of which require a preprocessing step of creating a table for size range, compressibility range, and the expected chunk size. The first method divides data objects into fixed-sized blocks and estimates their compression ratio using sampling techniques. Adjacent blocks with similar compression ratios are merged into segments. Segments are divided into chunks using content-defined chunking techniques, and these chunk boundaries may override segment boundaries. Then the chunk fingerprints are calculated.

In the second approach, many candidate chunking schemes using Content Defined Chunking (CDC) with different expected chunk sizes are generated in a single scan. One chunking scheme is used to calculate the compression ratio of its chunks, and chunks with similar compression ratios are merged together. These chunking results are directly used and their fingerprints are calculated. Their experimental results show that their Multi-Modal CDC can reduce the number of chunks by 29.1% to 92.4%.

Content-Defined Chunking for CPU-GPU Heterogeneous Environments

Ryo Matsumiya, The University of Electro-Communications; Kazushi Takahashi, Yoshihiro Oyama, and Osamu Tatebe, University of Tsukuba and JST, CREST

Chunking is an essential operation in deduplication systems, and Content-Defined Chunking (CDC) is used to divide a file into variable-sized chunks. CDC is slow as it calculates many fingerprints. The authors of this poster came up with parallelizing approaches that use both GPU and CPU to chunk a given file. Because of the difference in speed of CPU and GPU, the challenge becomes that of task scheduling. They propose two methods, Static Task Scheduling and Dynamic Task Scheduling, to efficiently use both the GPU and CPU.

Static Task Scheduling uses a user-defined parameter to determine the ratio of dividing the file such that one part is assigned to the GPU and the other part to the CPU. Each section is further divided into subsections, which are each assigned to a GPU thread or CPU thread.

Dynamic Task Scheduling consists of an initial master thread, which divides a file into distinct, fixed-sized parts called large sections. Each is assigned to a GPU thread. For detection of chunk boundaries lying across more than one segment, small subsections are created, including data parts across boundaries of large sections. Each small subsection is assigned to the CPU. Worker threads are created for GPU and CPU to handle these sections. While the task queue is not empty, each worker will perform CDC; if a queue becomes empty, then a worker will steal tasks from another worker's queue.

The authors ran experiments to find the throughput of their static and dynamic methods and compared them to CPU-only and GPU-only methods. They showed that static performs the best and dynamic follows closely behind it. The benefit of having a dynamic method is to avoid tuning parameters as is required for the static method.

Hash-Cast: A Dark Corner of Stochastic Fairness

Ming Chen, Stony Brook University; Dean Hildebrand, IBM Research; Geoff Kuenning, Harvey Mudd College; Soujanya Shankaranarayana, Stony Brook University; Vasily Tarasov, IBM Research; Arun O. Vasudevan, Stony Brook University; Erez Zadok, Stony Brook University; Ksenia Zakirova, Harvey Mudd College

This poster uncovers Hash-Cast, a networking problem that causes identical NFS clients to get unfair shares of the network bandwidth when reading data from an NFS server. Hash-Cast is a dark corner of stochastic fairness where data-intensive TCP flows are randomly hashed to a small number of physical transmit queues of NICs and hash values collide frequently. Hash-Cast influences not only NFS but also any storage servers hosting concurrent data-intensive TCP streams, such as file servers, video servers, and parallel file system servers. Hash-Cast is related to the bufferbloat problem, a phenomenon in which excessive network buffering causes unnecessary latency and poor system performance. The poster also presents a method to work around Hash-Cast by changing the default TCP congestion control algorithm from TCP CUBIC to TCP VEGAS, another algorithm that alleviates the bufferbloat problem.

MapReduce on a Virtual Cluster: From an I/O Virtualization Perspective

Sewoog Kim, Seungjae Baek, and Jongmoo Choi, Dankook University; Donghee Lee, University of Seoul; Sam H. Noh, Hongik University

The authors of this poster analyze two main questions with respect to the MapReduce framework making use of virtualized environments. They try to analyze whether Hadoop runs efficiently on virtual clusters and whether any I/O performance degradation is seen, then whether they can be mitigated by exploiting the characteristics of I/O access patterns observed in MapReduce algorithms. They ran a Terasort benchmark and found that the I/O is triggered in a bursty manner, requested intensively for a short period, and sharply increased and decreased. Some phases utilize a memory buffer. Virtual machines share I/O devices in a virtual cluster; thus, this bursty I/O may cause I/O interference among VMs. When VMs request bursty I/Os concurrently, the I/O bandwidth suffers a performance drop of about 31%, going from 1 to 4 VMs. Additionally, long seek distance and high context switch overheads exist among VMs.

To help mitigate this issue, the authors propose a new I/O scheduler for Hadoop on a virtual cluster, which minimizes the I/O interference among VMs and also exploits the I/O burstiness in MapReduce applications. Their new I/O scheduler controls bandwidth of VMs using Cgroups-blkio systems and operate at a higher layer than the existing scheduler. The Burstiness Monitor detects bursty I/O requests from each VM. The Coarse-grained Scheduler allows a bursty VM to use the I/O bandwidth exclusively for a time quantum in round-robin manner. This allows the overhead caused by context switching among multiple VMs to be reduced, along with reducing overall seek distance and execution time. Their experimental results verify the performance gains using their approach.

Keynote Presentation

FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers

Krste Asanović, University of California, Berkeley

Summarized by Tiratat Patana-anake (tiratatp@uchicago.edu)

Krste Asanović told a story about the past, present, and future of Warehouse Scale Computing (WSC), which has many applications but will gain popularity because of the migration to the extreme, with the cloud backing all devices. We moved from commercial off-the-shelf (COTS) servers to COTS parts, and we'll move to custom everything, said Asanović.

Asanović stated that the programming model for WSC needs to change. Currently, the silo model is used, but he believes that Service Oriented Architecture (SOA) is better. In SOA, each component is a service that connects via network, and each application is composed of many services. The benefits of SOA are reusability and ease of management. Moreover, by decomposing big software into small services, the services are easier to tailor to each user subset. A statistic shows that small (less than $1 million) projects have higher success rates than big projects.

Asanović then compared old wisdom to new wisdom in many related aspects. In the old wisdom, we cared only about building fault-tolerant systems. Today, we need to care about tail-latency tolerance, too, which means building predictable parts from less predictable ones. Many techniques can be used to build a tail-tolerant service. We can use software to reduce component variation by using different queues or breaking up tasks into small parts, or try to cope with variability by hedging requests when the first request result was slow, for example, or by trying requests in different queues. We can also try to improve the hardware by reducing overhead, reducing queuing, increasing network bisection bandwidth, or using partitionable resources.

The second thing that Asanović compared was the memory hierarchy. In the old days, we had DRAM, disk, and tape. Now, we have DRAM, NVRAM, and then disk. In the future, we will have a new kind of NVM that has DRAM read latency and endurance. In terms of memory hierarchy, there will be more levels in the memory hierarchy, and we might see a merging between high-capacity DRAM and flash memory into something new such as PCM.

Third, Moore's Law is dead (for logic, SRAM, DRAM, and likely 2D flash), said Asanović. The takeaway is that we have to live with this technology for a long time, and improvements in system capability will come from above the transistor level. More importantly, without Moore's Law and scaling, the cost of custom chips will come down because of the amortized cost of technology.

Fourth, Asanović discussed which ISA (Instruction Set Architecture) was better, ARM or x86. He said the real important difference was that we could build a custom chip with ARM, not Intel. He added that ISA should be an open industry standard. The goal is to have an open source chip design, such as Berkeley's RISC-V, which is an open ISA.

Moreover, Asanović said that security is very important. The key is to have all data encrypted at all times. He also said that rather than using shared memory to do inter-socket communication, message passing has won the war, which is also a better match for SOA.

Next, Asanović presented the FireBox, which is a prototype of 2020 WSC design. It is a custom "Supercomputer" for interactive and batch application that can support fault and tail tolerance, will have 1000 SoC, 1 terabit/s high radix photonic switch, and 1000 NVM modules for 100 PB total. FireBox SoC will have 100 homogeneous cores per SoC with cache coherence only on-chip and acceleration module (e.g., vector processors). NVM stack will have photonic I/O built in. Photonic switch is monolithic integrated silicon photonics with wave-division multiplexing (WDM). Photonics will have bandwidth of 1 Tb in each direction. Moreover, data will always be encrypted. Asanović said the bigger size will reduce operation expenses, can support a huge in-memory database, and has low latency network to support SOA. There are still many open questions for FireBox, such as how do we use virtualization, how do we process bulk encrypted memory, and which in-box network protocol should we use?

Finally, Asanović talked about another related project, DIABLO, which is an FPGA that simulates WSC. By using DIABLO, they found that the software stack is the overhead.

Rik Farrow wondered about the mention of Linux as the supported operating system. Asanović said that people expected a familiar API, but that FireBox would certainly include new operating system design. Kimberly Keeton (HP Labs) asked who is the "everybody" that considers using custom design chips. Asanović answered big providers are building custom chips, and if we move to an open source model, we will start seeing more. Keeton also asked a question about code portability. Asanović explained that 99.99% of code in applications doesn't need an accelerator. So, only the .01% that does need accelerators will need any changes.

Someone from VMware asked about the role of disk in FireBox. Asanović replied that disk and DC-level network are outside the scope of this project right now. Tom Spinney (Microsoft) asked about protecting keys and cryptography engines. Asanović responded that they planned on using physical mechanisms, and that this was an area of active research.

Experience from Real Systems

Summarized by Xing Lin (xinglin@cs.utah.edu)

(Big) Data in a Virtualized World: Volume, Velocity, and Variety in Cloud Datacenters

Robert Birke, Mathias Bjoerkqvist, and Lydia Y. Chen, IBM Research Zurich Lab; Evgenia Smirni, College of William and Mary; Ton Engbersen, IBM Research Zurich Lab

Mathias Bjoerkqvist started his presentation by noting that virtualization is widely used in datacenters to increase resource utilization, but the understanding of how I/O behaves in virtualized environments is limited, especially at large scales. To get a better understanding, they collected I/O traces in their private production datacenters over a three-year period. The total I/O trace was 22 PB, including 8,000 physical host machines and 90,000 virtual machines. Most of virtual machines were Windows and ran within VMware. Then they looked at how capacity and data changed and characterized read/write operations at the virtual machine layer and the host layer. They also studied the correlation between CPU, I/O, and network utilization for applications.

What's most interesting in their findings is that most contributions to the peak load came from only one third of virtual machines, which implies that we could improve the system by optimizing these few VMs. In their study, they also found a diverse set of file systems were used: For each virtual machine, as many as five file systems were used. Thus, about 20 virtual file systems were used in each host machine on average. The more CPUs and memory the host machine has, the more file systems. The data churn rate at the virtual machine layer is lower than at the host layer: 18% and 21%, respectively. They also looked at I/O amplification and deduplication rate. Comparing the number of I/Os at the virtual machine layer and the physical block layer at the host operating system, they found that amplification appears more often than deduplication. Finally, they used a k-means clustering algorithm to classify application workloads and found a strong correlation between CPU usage and I/O or network usage.

One person asked for the breakdown of true deduplication and caching effect. He suggested that the deduplication rates presented could be the combination of both. The author acknowledged that he was correct: They measured the total I/Os at the virtual machine layer and the physical host layer and were not able to distinguish between the caching effect and true deduplication. Fred Douglis (EMC) pointed out that the comparison of the data churn rate presented in this work and his own previous work was not appropriate because the workloads for this paper were primary workloads, whereas Fred's work studied backup workloads. Yaodong Yang (University of Nebraska-Lincoln) asked about the frequency of virtual machine migration in their datacenters. The authors replied that the peak load varied over time; the peak load in the middle of night could be correlated with virtual machine migration activities. Christos Karamanolis (VMware) asked a few fundamental clarification questions about their measurements, such as what the definition of a virtual I/O was, what the side-effect was from write buffering and read caching at the guest OS, and which storage infrastructure was used at the back end. The authors said they collected I/O traces at the guest OS and host OS level and suggested that people come to their poster for more details.

From Research to Practice: Experiences Engineering a Production Metadata Database for a Scale Out File System

Charles Johnson, Kimberly Keeton, and Charles B. Morrey III, HP Labs; Craig A. N. Soules, Natero; Alistair Veitch, Google; Stephen Bacon, Oskar Batuner, Marcelo Condotta, Hamilton Coutinho, Patrick J. Doyle, Rafael Eichelberger, Hugo Kiehl, Guilherme Magalhaes, James McEvoy, Padmanabhan Nagarajan, Patrick Osborne, Joaquim Souza, Andy Sparkes, Mike Spitzer, Sebastien Tandel, Lincoln Thomas, and Sebastian Zangaro, HP Storage

Kimberly Keeton characterized her team's work as special, covering their experience in transforming a research prototype (LazyBase) into a fully functional production (Express Query). Unstructured data grows quickly, at 60% every year. To make use of unstructured data, the metadata is usually used to infer the underlying structure. Standard file system search function is not feasible for providing rich metadata services, especially in scale-out file systems. The goal of their work is to design a metadata database, to allow rich metadata queries for their scale-out file systems.

LazyBase was designed to handle high update rates, at the expense of some amount of staleness. This matches well with the design of Express Query. Thus, they started with LazyBase and made three main changes to transform it into Express Query. The first change was to eliminate automatic incremental ID for long strings and ID remap because the mapping cannot be done lazily and the assignment of ID for a string cannot be done in parallel. Through experimentation, they found that ID-based lookup or joins was inefficient: minutes for ID-based versus seconds for non-ID based.

The second change was related to the transaction model in LazyBase, which allows updates to be applied asynchronously at a later time. When users delete a file, there is no way to reliably read and delete the up-to-date set of custom attributes. To deal with this problem, the authors introduced timestamps to track file operation events and attribute creations. These timestamps were then used to check for attribute validation during queries. To get the metadata about files, they put a hook in the journaling mechanism in the file system. Then the metadata was aggregated and stored into LazyBase. To support SQL-like queries, they used PostgreSQL on top of LazyBase. A REST API was designed to make Express Query easier and more flexible to use.

Scott Auchmoody (EMC) asked about using hashing to get IDs. Kimberly said that would break the locality; tables are organized according to IDs and with hashing, records for related files could be stored far away from each other. Brent Welch (Google) suggested that distributed file systems usually have an ID for each file, which could be taken and used. Kimberly acknowledged that they had taken advantage of that ID to be a unique identity for each file, and the index and sorting are based on pathname. One person noted that if metadata for files within a directory was organized as a tree structure, the tree structure could become very huge and wondered how much complexity was involved in managing large tree structures. Kimberly answered that they stored metadata as a table, and the directory structure was encoded in the path name. Shuqin Ren (Data Storage Institute, A*STAR) asked what the overhead was when adding the hook in the journaling mechanism to collect metadata. Kimberly replied that the journaling happened anyway, and they did not add any other instrumentation to the file system so the overhead was small.

Analysis of HDFS under HBase: A Facebook Messages Case Study

Tyler Harter, University of Wisconsin—Madison; Dhruba Borthakur, Siying Dong, Amitanand Aiyer, and Liyin Tang, Facebook Inc.; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Tyler Harter presented his work on analyzing I/O characterizations for the layered architecture of HBase for storing and processing messages in Facebook. Reusing HBase and HDFS lowers the development cost and also reduces the complexity of each layer. However, Tyler also pointed out that layered architecture could have some performance overheads and that performance could potentially be improved if the system were layer-aware. Their work studied how each layer altered I/O requests and exploited two use cases to demonstrate that performance could be improved by making the system layer-aware.

Tyler showed that at the HDFS layer, only 1% of all read/write requests it received were writes. However, writes became 21% because of compaction and logging used in HDFS. At the local file system layer, 45% of requests were writes because of three-way replication. At the disk layer, the percentage of writes was 64%. The same trend held when they considered I/O size. Two thirds of data is cold. Then Tyler presented the distribution of file sizes. Half of files used in HBase had sizes smaller than 750 KB. For access locality, they found a high temporary locality: the hit rate was 25% if they kept accessed data for 30 minutes and 50% for two hours. Spatial locality was low, and as much as 75% reads were random. Then they looked into what hardware upgrade was most effective in terms of I/O latency and cost. They suggested that adding a few more SSDs gave the most benefit with little increase in cost; buying more disks did not help much. To demonstrate that performance is increased by making the system layer-aware, Tyler presented two optimizations: local compaction and combined logging. Instead of sending compacted data, they proposed sending the compact command to every server to initiate compaction. This change reduced the network I/O by 62%. With combined logging, a single disk in each server machine was used for logging while other disks could serve other requests. This change reduced disk head contention, and the evaluation showed a six-fold speedup for log writes and performance improvement for compaction and foreground reads.

Bill Bolosky (Microsoft Research) pointed out that file size distribution usually had a heavy tail and thus the mean file size would be three times larger than the median size. He asked whether the authors had looked into the mean size. Tyler acknowledged that in their paper they did have numbers for mean file size but he did not remember them. He also suggested that, because most files were small, what mattered for performance was the number of files, specifically metadata operations. Brad Morrey (HP Labs) noted that for their local compaction to work, related segments had to be stored in a single server. This would affect recovery performance if a server died. Tyler replied that they did not do simulation for recovery. However, the replication scheme was pluggable and they should be able to exploit different replication schemes. Margo Seltzer (Harvard) suggested that it was probably wrong to use HDFS to store small files; HDFS was not designed for that, and other systems should probably be considered. Tyler acknowledged that was a fair argument, but Facebook uses HDFS for a lot of other projects and that argues for using a uniform architecture.

Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces

Yang Liu, North Carolina State University; Raghul Gunasekaran, Oak Ridge National Laboratory; Xiaosong Ma, Qatar Computing Research Institute and North Carolina State University; Sudharshan S. Vazhkudai, Oak Ridge National Laboratory

Yang Liu started his talk by introducing the second fastest supercomputer in the world: TITAN, which has 18,000 compute nodes. More than 400 users use this supercomputer for various scientific computations, such as climate simulation. The Spider file system is used to provide file system service: In total, TITAN can store 32 PB of data and can provide 1 TB/s bandwidth. Because the supercomputer is shared by multiple users, workloads from different users could compete for the shared storage infrastructure and thus interfere with each other. Thus, it is important to understand the I/O behavior of each application. With that understanding, we can do a better job in scheduling these jobs and thus reduce I/O contention. Their work proposed an approach to extract I/O signatures for scientific workloads running from server-side tracing.

Client-side tracing is the other alternative but has a few drawbacks. Client-side tracing requires a considerable development effort, and it usually introduces some performance overhead, ranging from 2% to 8%. Different applications probably use different trace formats and may not be compatible. What's worse, client-side tracing introduces extra I/O requests. So they decided to use the coarse-grain logging at the RAID controller level at the server-side. It has no overhead and does not require any user effort. However, I/O requests are mixed at the server-side. The challenge is to extract I/O signatures for a particular application from this mixed I/O traffic. Fortunately, there are a few inherent features in scientific applications that can help to achieve that goal.

These applications usually have two distinct phases: compute phase and I/O phase. During the I/O phase, applications typically request large writes of either intermediate results or checkpointing, so there are periodic bursts for these applications. Besides, users tend to run the same application multiple times with the same configuration. Thus, I/O requests are repetitive for these applications. Given the job scheduler logs with start and end time for each job and the server-side throughput logs from Spider, they can extract the samples for a particular job. The insight from the authors is that the commonality across multiple samples tends to belong to the target application.

The challenges of extracting I/O signatures from these samples include background noise and I/O drift. To deal with these challenges, the authors proposed three stages to extract the I/O signature: (1) data preprocessing that eliminates outliers, refines the granularity of the samples, aligns durations, and reduces noise by removing light I/O traffic; (2) use of wavelet transform for each sample to make each sample smoother and to more easily distinguish bursts from background noise; and (3) use of the CLIQUE (Agrawal: SIGMOD '08) clustering algorithm to identify common bursts across samples. For evaluation, they used I/OR, a benchmark tool for parallel I/O to generate a few synthetic workloads and a real-world simulation: S3D. The signature extracted by their tool matched well with the actual I/O signature. They also compared the accuracy of their algorithm with Dynamic Time Warping (DTW) and found I/OSI outperformed DTW.

Kun Tang (Virginia Commonwealth University) asked whether it was true that users actually ran an application multiple times. Yang answered yes, based on what they observed from the job scheduler log and several previous works. One person suggested that if they already had the I/O signature, their tool would be able to reproduce that signature. Yang replied that based on their observation, each application would likely exhibit the same I/O pattern in future runs, and they could use this insight for better scheduling.

Works-in-Progress

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

The FAST '14 Works-in-Progress session started with Taejin Kim (Seoul National University), who presented his work on "Lifetime Improvement Techniques for Multiple SSDs." He observed that write amplification in a cluster of SSDs may degrade the device's lifetime because of intermediate layers which perform replication, striping, and maintenance tasks such as scrubbing. As an example, he showed how writes could be amplified by 2.4x under Linux's RAID-5 implementation. He proposed integrating many different write prevention techniques, such as compression, deduplication, and dynamic throttling, to lower the write workload, as well as smaller stripe units, delta compression, and modifying erasure coding algorithms to use trim in place of writes.

Dong Hyun Kang (Samsung Electronics) presented "Flash-Friendly Buffer Replacement Algorithm for Improving Performance and Lifetime of NAND Flash Storages." He explained how traditional buffer replacement algorithms are based on magnetic drives and proposed an algorithm called TS-Clock to perform cache eviction. TS-Clock is designed to limit write-backs and improve cache hit rate. His preliminary results show that on DBench the TS-Clock algorithm has up to a 22.7% improvement in hit rate and extends flash lifetime by up to 40.8%.

Douglas G. Otstott (Florida International University) described his work towards developing a "holistic" approach to scheduling. In his presentation, "A Host-Side Integrate Flash Scheduler for Solid State Drives," he listed the inefficiencies of the OS to flash interface. Flash devices themselves have limited resources to consider complex scheduling. However, in the relatively more powerful OS-layer, most of the potentially useful details about individual request performance, such as read vs. write latencies, write locations, and logical to physical block mappings, are hidden. In his alternate approach, device management occurs in the

OS stack but pushes commands down to the device to balance load, ensure that writes aren't committed ahead of reads, and limit GC performance costs. His proposed interface includes isolated per-die queues for read, write, and garbage collection. He is working to validate his Linux prototype in DiskSim and is working on an FPGA implementation.

Next, Eunhyeok Park (University of Pittsburgh) described his work: "Accelerating Graph Computation with Emerging Non-Volatile Memory Technologies." Algorithms that access large graph data structures incur frequent and random storage access. After describing the CPU bottlenecks in these workloads, he described Racetrack, an approach based on an idealized memory model. Park's simulations based on Pin show that it is faster and has lower energy consumption than alternatives. His preliminary results suggest that some simple computations, for example PageRank, could be done by a CPU or FPGA on the storage device itself.

In "Automatic Generation of I/O Kernels for HPC Applications," Babak Behzad (University of Illinois at Urbana-Champaign) described his efforts to autonomously generate an I/O kernel, which is a replayable trace that includes dependency constraints. His approach is to wrap I/O activity with an application-level library to trace parallel I/O requests in high-performance computing environments. His work promises to provide better analytics for future optimization, I/O auto-tuning, and system evaluation.

Florin Isaila (Argonne National Laboratory) described "Clarisse: Cross-Layer Abstractions and Runtime for I/O Software Stack of Extreme Scale Systems." His goal is to enable global optimization in the software I/O stack to avoid the inefficiencies of I/O requests that pass through multiple layers of uncoordinated memories and nodes, each of which provides redundant function. He proposed providing cross-layer control abstractions and mechanisms for supporting data flow optimizations with a unified I/O controller. Instead of the traditional layered model, Clarisse intercepts calls at different layers. Local control modules at each stack layer talk over a backplane to a controller that coordinates buffering, aggregation, caching prefetching, optimization selection, and event processing.

Howie Huang (George Washington University) presented "HyperNVM: Hypervisor Managed Non-Volatile Memory." He observed that warehouse-scale datacenters are increasingly virtualized and that many VMs performing large amounts of I/O require high performance memory subsystems and need better systems support for emerging non-volatile memories. To solve these problems, he proposed that the hypervisor needs to be more memory aware and that the OS and applications should pass more control of memory management to the hypervisor. He proposed a system called Mortar, which is a general-purpose framework for exposing hypervisor-controlled memory to applications.

Dan Dobre (NEC Labs Europe) described his work on "Hybris: Robust and Consistent Hybrid Cloud Storage." He noted that potential users of cloud storage services are concerned about security and weak consistency guarantees. To address these two concerns, he proposed a hybrid solution that employs a private cloud backed by multiple public clouds. His approach maintains a trusted private cloud to store metadata locally, with strong metadata consistency to mask the weak data consistency offered by public clouds.

In "Problems with Clouds: Improving the Performance of Multi-Tenant Storage with I/O Sheltering," Tiratat Patana-anake (University of Chicago) described an approach to limit the performance impact of random writes in otherwise sequential workloads. In his approach, called "I/O Sheltering," random writes are initially written sequentially inline with sequential writes. Later, these sheltered writes are moved to their in-place locations. He argued that this approach would be successful because random write applications are rare, memory is abundant, and NVM provides a better location to shelter indexes. He concluded by showing significant performance improvements in multi-tenant Linux, when one random writer ran in conjunction with many sequential writers. He proposed to integrate this technique into the file system and journal in the future.

Michael Sevilla (University of California, Santa Cruz) argued that load balancing should be based on a deeper understanding of individual node resources and contention in his work on "Exploring Resource Migration Using the CephFS Metadata Cluster." Sevilla has been using Ceph as a prototyping platform to investigate migration and load balancing. He has observed that decisions about whether to migrate are often non-intuitive. Sometimes migration helps mitigate loads, but it may also hurt, because it denies access to a full metadata cache. He is attempting to develop a distributed, low-cost, multiple objective solver to make more informed migration decisions. In closing, he demonstrated the practicality of the problem by detailing an experiment where re-ordering name servers in a cluster under high load resulted in a considerable performance improvement.

With his work on "Inline Deduplication for Storage Caching," Gregory Jean-Baptiste (Florida International University) proposed enhancing existing client-side flash caches with inline deduplication to increase their effective capacity and lifespan. He has a prototype system in place that shows better performance with the IOZone benchmark over iSCSI. He noted that this approach may lead to other architectural changes, such as replacing LRU with an eviction algorithm that is aware that evicting a widely shared block could be more painful that an unshared one.

Alireza Haghdoost (University of Minnesota) presented "hfplayer: High Fidelity Block I/O Trace Replay Tool." hfplayer can reproduce previously captured SAN workload with high fidelity for the purposes of benchmarking, performance evaluation, and validation with realistic workloads. His design is based on a pool of worker threads that issue I/O requests and a harvester thread to process completion events and keep track of in-flight I/O requests. To control replay speed, a load-aware algorithm dynamically matches the number of in-flight I/O requests with the load profile in the original trace. When inter-arrival times are very short, threads bundle requests together to avoid system call overhead.

Nagapramod Mandagere (IBM Research) has been analyzing the performance of backup servers and has found some interesting opportunities for optimization. In his presentation titled "Towards a Framework for Adaptive Backup Management," he described some of his observations. He noted that backup workloads show temporal skew, where most traffic arrives at hourly boundaries but large bursts of heavy utilization are possible. Another problem is spatial skew, in which one backup server is overloaded while others are not. Both these problems stem from backups that are managed by static policies while the workload they create is very dynamic. As client backup windows get shorter and shorter (due to lack of idle times), server initiated backups are harder to schedule, but when the clients are instead allowed to push updates themselves, the backup servers can become overloaded. Furthermore, backup servers themselves need idle periods for their own maintenance. He proposes an adaptive model-based backup management framework that dynamically determines client/server pairings to minimize client backup windows.

Performance and Efficiency

Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation

Hui Wang, Peter Varman, Rice University

Hui Wang started her presentation by talking about the trend of multi-tiered storage in modern datacenters with both solid state drives and traditional hard disks. This kind of combination has several advantages, including better performance in data access and lower cost. At the same time, it also brings some challenges, such as providing fair resource allocation among clients and maintaining high system efficiency. To be more specific, there is a big speed gap between SSD and HD, so scheduling proper workloads to achieve high resource utilization becomes very important. Hui Wang talked about fair resource allocation and high performance efficiency to make up for some disadvantages of heterogeneous clusters. She talked about her team's motivation using several examples: single device type, multiple devices, and dominant resource from both fairness and efficiency angles.

Based on these considerations, she presented a new allocation model, Bottleneck-Aware Allocation (BAA), based on the notion of per-device bottleneck sets. Clients bottlenecked on the same device receive throughputs in proportion to their fair shares, whereas allocation ratios between clients in different bottleneck sets are chosen to maximize system utilization. In this part, she discussed the fairness policy first, showed the bottleneck sets and fairness requirements of BAA, and then introduced their optimization algorithm of allocation.

They performed two simulations: one to evaluate the BAA's efficiency, and the other monitoring BAA's dynamic behavior under changing workloads. They also implemented a prototype, interposing BAA scheduler in the I/O path and evaluated BAA's efficiency and fairness.

Umesh Maheshwari (Nimble Storage) asked if that kind of dramatic ratio somehow changed the nature of this work. Hui Wang said, suppose it was for a single disk compared with SSD. It is most likely that HD would be the bottleneck, but if you have a large HD array, the speed gap between the two is not so dramatically large, so you would very easily have the balanced set cluster on both. Umesh asked again, assuming there was such a large difference between the two tiers, would the BBA model still hold the same performance using this approach? Hui Wang answered yes. Shuqin (Data Storage Institute Singapore) asked whether their model would work on multiple nodes. Hui Wang said she thought that their model could be easily extended to multiple nodes with coordination between schedulers. Kai Shen (University of Rochester) asked what was particularly challenging in this project. Hui Wang said the most challenging part was to accurately estimate the capacity of the system.

SpringFS: Bridging Agility and Performance in Elastic Distributed Storage

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

In this presentation, Xu introduced the concept of elasticity in distributed storage. Elastic distributed storage means storage that is able to resize dynamically as workload varies, and its advantages are the ability to reuse storage for other purposes or reduce energy usage; they can decide how many active servers to provide to work with a changing workload. By closely monitoring and reacting quickly to changes in workload, machine hours are saved. But most current storage, such as GFS and HDFS, is still not elastic. If they deactivate the nodes, it will cause some data to not be available. Before Xu talked about SpringFS, he gave two examples of prior elastic distributed storage, Rabbit and Sierra, discussing the differences between them and the disadvantages of each. Fortunately, SpringFS provides balance and fills the gap between them.

First, Xu showed a non-elastic example: in this case, almost all servers must be "active" to be certain of 100% availability, so it has no potential for elastic resizing. He then discussed the general rule of data layout in elastic storage and tradeoff space. Based on this, the authors proposed an elastic storage system, called SpringFS, which can change its number of active servers quickly, while retaining elasticity and performance goals. This model borrows the ideas of write availability and performance offloading from Rabbit and Sierra, but it expands on previous work by developing new offloading and migration schemes that effectively eliminate the painful tradeoff between agility and write performance in state-of-the-art elastic storage designs. This model, combined with the read offloading and passive migration policies used in SpringFS, minimizes the work needed before deactivation or activation of servers.

Muhammed (HGST) asked what is used to predict the performance ahead of time and whether the offload set value could be adjusted. Xu said they actually did not design this part of the workload in their paper. They just assumed they had the perfect predictor, and that it was possible to integrate some workload predictor into their work. One person asked whether this model tolerates rack fails. Xu said yes, because their model was modified from HDFS, it could tolerate rack fails as well as HDFS. Another questioner asked what offloading in SpringFS essentially means. Xu explained that offloading means redirecting requests from a heavy loaded server to a lightly loaded server.

Migratory Compression: Coarse-grained Data Reordering to Improve Compressibility

Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane and Grant Wallace, EMC Corporation-Data Protection and Availability Division, University of Utah

Lin presented Migratory Compression (MC), a coarse-grained data transformation, to improve the effectiveness of traditional compressors in modern storage systems. In MC, similar data chunks are relocated together to improve compression factors. After decompression, migrated chunks return to their previous locations. This work is motivated by two points. The first is to exploit redundancy across a large range of data (i.e., many GB) during compression by grouping similar data together before compressing. The second point is to improve the compression for long-term retention. However, the big challenge in doing MC is to identify similar chunks efficiently and scalably; common practice is to generate similarity features for each chunk because two chunks are likely to be similar if they share many features.

There are two principal use cases of MC. The first case is mzip, that is, compressing a single file by extracting resemblance information, clustering similar data, reordering data in the file, and compressing the reordered file using an off-the-shelf compressor. The second case is archival, involving data migration from backup storage systems to archive tiers or data stored directly in an archive system, such as Amazon Glacier. The evaluation result showed that adding MC to a compressor significantly improves the compression factor (23–105% for gzip, 18–84% for bzip2, 15–74% for 7z, and 11–47% for rzip), and we can also see the improvement of compression throughput with MC. In all, MC improves both compression factor and throughput by deduplication and reorganization.

Cornel Constantinescu (IBM Almaden Research Lab) asked whether this model compressed the file. Lin said that in the mzip case, they un-compacted the whole file. Constantinescu asked whether they did a comparison with other similar work, such as work used in Google's BigTable. Lin admitted he did not know that. Jacob Lorch (Microsoft Research) asked whether they were starting to look at modifying the compression algorithms. Lin said that he did not think they modified compression itself and that MC could be used as a generic preprocessing stage that could benefit all of the compression. And, if they are improving compressions, they can get the same benefits by doing this as a separate stage.

Poster Session II

Summarized by Kai Ren (kair@cs.cmu.edu)

VMOFS: Diskless and Efficient Object File System for Virtual Machines

Shesha Sreenivasamurthy and Ethan Miller, UC Santa Cruz

This project is to design an object file system for virtual machine environment by deduplicating common files shared in many VM images. In their architecture, guest machines will run a VFS layer software called VMOFS to track the mapping between inode and object, and a hypervisor manages the storage and deduplicates file objects and stores them into underlying object storage. Deduplication is achieved at a per-object level to reduce the dedup table size. This work is still under development, and no experimental results were presented.

Characterizing Large Scale Workload and Its Implications

Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, and Sam H. Noh, Dankook University, University of Seoul and Hongik University

This project analyzes storage workloads from a collection of traces from various cloud clusters. The observations are (1) workloads with large data are not sensitive to cache size; (2) cache allocation should be dynamically tuned due to irregular cache status changes; and (3) to achieve high cache hit ratio, one might use a policy that quickly evicts blocks with large request size and hit counts in a special period. The next part of this project will be to apply these observations to design and implement a cache service for virtual machine clusters.

Automatic Generation of I/O Kernels for HPC Applications

Babak Behzad, Farah Hariri, and Vu Dang, University of Illinois at Urbana-Champaign; Weizhe Zhang, Harbin Institute of Technology

The goal of this project is to automatically extract I/O traces from applications and regenerate these workloads to different scales of storage systems for performance measurement or testing. To fulfill such a goal, the workload generation systems have long workflows. First, it extracts traces from multiple levels of the I/O flow—application, MPI-I/O, and POSIX I/O levels—and from multiple processes. The second step is to merge these traces and understand the dependency between I/O operations presented in the traces. To construct such a dependency, it needs to understand semantics of I/O operations, which is mostly inferred from MPI-I/O library calls. For operations without clear dependency, timestamps are used to decide the order. The last step is to divide the traces, and generate binaries to replay the traces or simulate the workloads.

Poster Session II

Summarized by Qian Ding (qding8@gmail.com)

VMOFS: Breaking Monolithic VM Disks into Objects

Shesha Sreenivasamurthy and Ethan Miller, UC Santa Cruz

Shesha and Ethan propose a solution of efficient file sharing among virtual machines (VMs) through an object-based root file system. They are building a file system called VMOFS, which adopts object-level deduplication to store monolithic VM images. The hypervisor layer in the system encapsulates the OSD layer and controls the communication between file system and the OSDs via iSCSI. VMOFS is still under development so there is no evaluation at the current stage.

Characterizing Large Scale Workload and Its Implications

Dongwoo Kang, Seungjae Baek, Jongmoo Choi, Donghee Lee, and Sam H. Noh, Dankook University, University of Seoul and Hongik University

Dongwoo Kang presented work about characterizing a group of recently released storage traces and explained their observations and possible implications. The traces the team used were from MSR-Cambridge and FIU. Their first finding was that such traces were not sensitive to cache size when using a simple LRU cache but have much variation on the change of cache hit ratio and I/O size. They also found long inter-reference gaps (IRG) from the traces and provide that as a reason for low cache sensitivity. They propose two ways to improve the hit ratio: dynamic cache allocation and evicting short IRGs in the cache. They also propose ways for optimizing cache utility on a virtualized storage environment.

MicroBrick: A Flexible Storage Building Block for Cloud Storage Systems

Junghi Min, Jaehong Min, Kwanghyun La, Kangho Roh, and Jihong Kim, Samsung Electronics and Seoul National University

Jaehong Min presented the design of MicroBrick for cloud storage. The authors tried to solve the diverse resource requirement problem, such as balancing the computing and storage resource in cloud services. Each MicroBrick node adopts flexible control for both CPU-intensive and storage-intensive configuration through a PCIe switch. Thus, when the cloud system is composed of MicroBrick nodes, they can use a software management layer for autoconfiguration for different resource requirements. The preliminary evaluation of MicroBrick shows competitive results by running cloud computation (e.g., wordcount) and sort programs.

OS and Storage Interactions

Summarized by Kuei Sun (kuei.sun@utoronto.ca)

Resolving Journaling of Journal Anomaly in Android I/O: Multi-Version B-Tree with Lazy Split

Wook-Hee Kim and Beomseok Nam, Ulsan National Institute of Science and Technology; Dongil Park and Youjip Won, Hanyang University

Wook-Hee Kim began the talk by reminding us that although Android is the most popular mobile platform to date, it has severe I/O performance bottlenecks. The bottlenecks are caused by the "journaling of journal anomaly" between ext4 and SQLite, which is used by many applications. Kim gave a stunning example where one insert of 100 bytes resulted in nine random writes of 4 KB each. Currently, the database calls fsync() twice: once for journaling and once for insertion. Kim and his colleagues proposed to obviate the need for database journaling by implementing a variant of multi-version B-tree named LS-MVBT.

Kim presented several optimizations made to LS-MVBT based primarily on the characteristics of the common workloads. Lazy split seeks to reduce I/O traffic during node split by simultaneously garbage collecting dead entries so that the existing node (aka lazy node) can be reused. However, if there are concurrent transactions and the dead entries are still being accessed, then a workaround is needed. Their solution is to reserve space on each node so that new entries can still be added to the lazy node without garbage collection. To further reduce I/O traffic, instead of periodic garbage collection, LS-MVBT does not garbage collect unless space is needed. Next, Kim showed that by not updating the header pages with the most recent file change counter, only one dirty page needs to be flushed per insertion. He pointed out that this optimization would not increase overhead during crash recovery because all B-tree nodes must be scanned anyway. Lastly, they disabled sibling redistribution from the original MVBT and forced a node split whenever a node became full. Kim showed that this optimization actually reduced the number of dirty pages.

In their evaluation, Kim showed that LS-MVBT improves performance of database insertions by 70% against the original SQLite implementation (WAL mode). LS-MVBT also reduces I/O traffic by 67%, which amounts to a three-fold increase in the lifetime of NAND flash. LS-MVBT is also five to six times faster than WAL mode during recovery. Lastly, LS-MVBT outperforms WAL mode unless more than 93% of the workload are searches. At the end of the talk, Kim's colleague demonstrated a working version of their implementation, showing the performance improvement in a simulated environment.

Journaling of Journal Is (Almost) Free

Kai Shen, Stan Park, and Meng Zhu, University of Rochester

Kai Shen started by arguing that journaling of journal is a violation of the end-to-end argument and showed that adding ext4 journaling to SQLite incurs a 73% slowdown in their experiments. Existing solutions require substantial changes to either the file system or the application. The authors proposed two simple techniques.

Single-I/O data journaling attempts to minimize the number of device writes on the journal commit's critical path. In this case, data and metadata are journaled synchronously. During their experiments, they discovered a bug with ext4_sync_file(), which unnecessarily checkpoints data to be overwritten or deleted soon. The problem with the data journaling is the large volume written due to the page-sized granularity of ext4. Therefore, they proposed a second technique called file adaptive journaling, which allows each file to have a custom journaling mode. They found the solution effective for the journaling of journal problem. In particular, write-ahead logs would prefer ordered journaling due to little metadata change, and rollback logs would prefer data journaling due to heavy metadata change. In their evaluation, Shen showed that their enhanced journaling incurs either little to no cost.

Theodore Wong (Illumina, Inc.) asked what would happen if file system journaling were not used. Shen explained that protection of both file system metadata and application data still would be necessary because an untimely crash may cause the file system to become inconsistent. Because the database only protects its own data in a file, an inconsistent file system may cause the database file to become inaccessible. Kim commented on the fact that enterprise databases usually skip over the file system and operate directly on raw devices. However, lightweight databases may still prefer running on top of file systems for simplicity. Peter Desnoyer (Northeastern University) wanted to know whether these changes could affect the opportunity for inconsistency. Shen replied that as long as the failure model is fail-stop, then all of the guarantees would still apply.

Checking the Integrity of Transactional Mechanisms

Daniel Fryer, Mike Qin, Kah Wai Lee, Angela Demke Brown, Ashvin Goel, and Jack Sun, University of Toronto

Daniel Fryer began his talk by showing us that corruptions caused by file system bugs are frequently catastrophic because they are persistent, silent, and not mitigated by existing reliability techniques. The authors' previous work, Recon, ensures that file system bugs do not corrupt data on disk by checking the consistency of every transaction at runtime to prevent corrupt transactions from becoming durable. Because Recon performs its checks at commit time, it requires the underlying file system to use a transaction mechanism. Unfortunately, Recon does not detect bugs in the transaction mechanism. Their solution is to extend Recon to enforce the correctness of transaction mechanisms.

Recon already checks the consistency of transactions by verifying that metadata updates within a transaction are mutually consistent. To enforce atomicity and durability, Recon needs to also be able to catch unsafe writes and prevent them from reaching disk. Fryer defined two new sets of invariants, atomicity and durability invariants (collectively called location invariants), which govern the integrity of committed transactions. These invariants need to be checked on every write to make sure that the location of every write is correct. Fryer walked us through two types of transaction mechanisms, journaling and shadowing paging, as well as their respective invariants. Next, he presented a list of file system features that are required for efficient invariant checking at runtime.

Fryer and his team implemented location invariants for ext3 and btrfs by extending Recon. They had to retrofit ext3 with a metadata bitmap to distinguish between data and metadata, which is required to detect unsafe overwrites to metadata blocks. To evaluate the correctness of their implementation, they corrupted file system writes of various types to simulate bugs in the transaction mechanism and successfully caught most of them. The ones they missed did not affect file system consistency. Finally, they showed that adding location invariants to Recon incurs negligible overhead.

Keith Smith (NetApp) asked what could be done after a violation was detected. Fryer responded that while optimistically delaying a commit is plausible, what they've done at the moment is to return an error since their first priority is to prevent a corrupting write from reaching disk. Ted Ts'o (Google) wanted clarification on why losing some writes did not affect correctness during the corruption experiments. Fryer explained that the particular implementation of the file system was suboptimal and was checkpointing some metadata blocks unnecessarily because future committed versions of those blocks exist in the journal. Therefore, losing those writes was inconsequential. Harumi Kuno (HP Labs) was baffled by the fact that checking both consistency and location invariants resulted in better performance than just checking consistency alone. Fryer believed that it was simply due to an insufficient number of trials.

OS and Peripherals

Summarized by Matias Bjørlin (mabj@itu.dk)

DC Express: Shortest Latency Protocol for Reading Phase Change Memory over PCI Express

Dejan Vučinić, Qingbo Wang, Cyril Guyot, Robert Mateescu, Filip Blagojević, Luiz Franca-Neto, and Damien Le Moal, HGST San Jose Research Center; Trevor Bunker, Jian Xu, and Steven Swanson, University of California, San Diego; Zvonimir Bandić, HGST San Jose Research Center

Dejan Vučinić began on a side note by stating that the next big thing isn't DRAM, because of its high energy utilization, referring to the previous FireBox keynote. He then stated the motivation for his talk by showing upcoming non-volatile memories and their near DRAM access timings. He explained how they each compete with DRAM on either price or latency and showed why PCM has fast nanosecond reads but microsecond writes. He explained that to work with PCM, the team built a prototype, exposing it through a PCI-e interface.

Dejan then showed how PCI-e communicates with the host using submission and completion queues. When a new request is added to the queue, a doorbell command is issued to the PCI-e device. When received, the device sends a DMA request to which the host returns the actual data request. The request is processed, and data with a completion command at the end is sent. The handshake requires at least a microsecond before any actual data is sent. To eliminate some of the overhead, they began by removing the need for ringing the doorbell. They implemented an FPGA that continuously polls for new requests on the memory bus. Then they looked at how completion events take place, which the host could poll for when the data is finished. However, instead of the host polling, they show that completion can be inferred from the response data by inserting a predefined bit pattern in DRAM. Data is stored in DRAM from the device and should be different; thus, they could infer when all data packets have been received.

Dejan then showed that using their approach they achieve, using a single PCI-e lane, a 1.4 μs round-trip time for 512 bytes. Even with these optimizations, there continues to be large overhead involved, and thus the fundamental overhead of the communication protocol should be solved to allow PCM and other next-generation memories to be used efficiently.

Brad Morrey (HP Labs) asked why they didn't put it on the DRAM bus. Dejan replied that there is a need for queues within the DRAM. The queues are needed to prevent stalls while waiting. He wants to have it there in the future, but PCM power limitations should be taken into consideration. Peter Desnoyers (Northeastern University) asked if they had a choice on how PCI-e on PCM could evolve and what would they do? Dejan answered that it could be used with, for example, hybrid memory cubes. Finally, Ted Ts'o (Google) noted that the polling might be expensive in power. What is the power impact? Dejan said that it is very low, as you may only poll during an inflight I/O.

MultiLanes: Providing Virtualized Storage for OS-Level Virtualization on Many Cores

Junbin Kang, Benlong Zhang, Tianyu Wo, Chunming Hu, and Jinpeng Huai, Beihang University

Junbin Kang began by stating that manycore architectures exhibit powerful computing capacity and that virtualization is necessary to use this capacity. He then continued to argue paravirtualization versus operating system virtualization and ended by comparing the many layers of a traditional virtualization stack with a stack using containers. Containers are simpler in their architecture, but they expose scalability issues within the Linux kernel.

To solve the scalability issues, Junbin presented MultiLanes. The data access for containers are partitioned using a partitioned Virtual File System (pVFS) abstraction. pVFS exposes a virtualized VFS abstraction for the containers. In turn, the pVFS eliminates contention within the host VFS layer and improves locality of the VFS data structures. pVFS communicates with the host using a container-specific virtualized block driver (vDrive). The driver takes care of submitting the I/Os from the container to the host system. This allows each container data partition to be split and thereby avoid contention on host VFS locks. Junbin then explained the internal structure of the virtualized driver.

They evaluated their solution using both micro and macro-benchmarks on the file systems ext3, ext4, XFS, and btrfs, with microbenchmarks being metadata operations and sequential writes. For all of them, the operations scale significantly better with additional containers. The macrobenchmarks consist of varmail, fileserver, and MySQL, showing similar performance improvements. Finally, they discussed the overhead of their solution and show it to be negligible in most cases. However, excessive block remapping did have a cost during high throughput.

Kai Shen (University of Rochester) asked whether MultiLanes adds a large memory footprint for each of the channels they create. Junbin said that their approach is complementary to previous approaches. Yuan (UCSC) noted that for a large number of containers using XFS the performance was much better than for ext4. The answer from Junbin and Theodore Ts'o, the ext4 maintainer, was that it depends on the kernel version and other work. Further discussion was taken offline.

Linux FAST Summit '14: 2014 USENIX Research in Linux File and Storage Technologies Summit

San Jose, CA
February 20, 2014

Summarized by Rik Farrow

The Linux FAST Summit took place shortly after FAST '14 had finished. Red Hat offices in Mountain View provided a classroom that sat 30, but the room was full to the point that some of us were sitting in folding chairs at the front or back. Ric Wheeler (Red Hat) moderated the workshop.

Unlike the one Linux Kernel Summit I attended, the focus of this event was strictly on file systems and related topics. Another difference was that—instead of having mainly industry in attendance making requests of kernel developers—file system researchers, mostly students and some professors, were there to make requests for changes in how the kernel works.

The workshop began with Ric Wheeler encouraging people to submit changes to the kernel. He also explained the kernel update process, where a release candidate will come up and be followed by multiple RCs that, outside of the first two RCs, should only include bug fixes. Someone asked about rude answers from Linus Torvalds, and James Bottomley (Parallels) responded that sending in patches with new features late in the RC process is a common way to get an angry response from Linus. Also, the larger the patch set, the less likely it will be accepted. Ted Ts'o (Google) pointed out that just getting a response from Linus is a big deal and suggested seeing who is responding to such a response and who is being ignored. He also said that large intrusions to core infrastructure are less likely to be accepted. Christoph Hellwig (freelance) pointed out that changes to the core of the kernel are harder to validate and less likely to be accepted, which makes it very hard to get very high-impact changes made.

Ethan Miller (UCSC) wondered who would maintain the code that a PhD contributes after they graduate, and Christoph Hellwig responded that they want the code to be so good that the candidate gets a job supporting it later. Ric Wheeler commented on "drive-by" code drops, and said those are fine; if there is broad community support for that area, like XFS or ext4, people will review and support the change. James Bottomley said that it is important to be enthusiastic about your patch. Ted Ts'o chimed in saying that even if your patch is not perfect, it could be seen as a bug report, or some portion of it might be successful. James added that Google recruiters look for people who can take good ideas and turn them into working implementations.

Jeff Darcy (Red Hat) asked where they find tests for patches, and James said they could look at other file systems. xfstests go way beyond XFS, and the VM crew has lots of weird tests.

Ethan Miller launched a short discussion about device drivers, wondering who maintains them, and James Bottomley said that any storage device that winds up in a laptop will be accepted into the kernel. Ted Ts'o said that there were lots of device drivers with no maintainer, and James Bottomley suggested that Fengguang Wu (Intel) runs all code in VMs for regression testing. Andreas Dilger (Google) pointed out that his is just basic testing. James said that SCSI devices might stay there forever, or until someone wants to change the interface.

Ric Wheeler suggested covering how the staging tree for the kernel works. James explained that there are about 300 core maintainer trees, and kernel releases are on a two- to three-month cycle. Ted Ts'o said that RC1 and RC2 are where patches go in, with patches accepted to RC3 being rare, and that by RC7 they better be really critical patches, as that is just before a version release. Ted also suggested that if you are a commercial device builder, you want to test your device with RC2 to see if changes have broken the driver for your device.

Erez Zadok (Stony Brook) asked for a brief history of the memory management (MM) tree. Andrew Morton (Linux Foundation) explained that MM had become a catchall tree where stuff also falls through. Andrew collects these patches and pushes them up to Linus.

After a short break, Ric Wheeler opened the discussion about shingled drives. Ted Ts'o had suggested the *;login:* article by Tim Feldman (Seagate) and Garth Gibson (CMU) [1] as good preparation for this part of the summit. Briefly, Shingled Magnetic Recording (SMR) means that written tracks overlap, like shingles on a roof. These tracks can still be read, but writing must occur at the end of a band of these shingled tracks, or a band can be completely rewritten, starting at the beginning. Random writes are not allowed. Vendors can make SMR drives that appear like normal drives (managing the changes), partially expose information about the drives (restricted), or allow the operating system to control writing the drives (host-aware).

Ted Ts'o explained that he had proposed a draft interface for SMR on the FSDEVEL list. The goal is to make the file system friendly to having large erase blocks like flash, which matches host-aware SMR behavior. If the device is restricted, you still need the operating system to be aware that it is an SMR drive. Ted suggested that this could be part of devmapper (if part of the OS) and could begin as a shim layer.

Erez Zadok said that a group at Stony Brook had been working with Western Digital and had received some SMR drives with a firmware that does more and more, along with a vanilla drive. One of his graduate students reports to Jim Molina, CTO of Western Digital. Erez wanted to share what they've learned so far with the Linux file system community.

First, the vendors will not let us put active code into drives. They will provide a way of knowing when garbage collection (GC) is about to start, and some standards are evolving. Vendors are conscious of the desire for more visibility into the 500k lines of code already in drives.

Because the drives come to Erez Zadok preformatted, Ric Wheeler wondered what percentage remains random, as opposed to shingled, bands. Erez said that the vendors wanted less than 1% of SMR drives as random (traditional tracks) because of the economics involved. They have already done some work using NULLFS with the drives, and Jeff Darcy said he had experimented with using a shim in the device mapper. Ted Ts'o pointed out there was a real research opportunity here in building a basic redirection layer that makes a restricted drive appear like a plain device.

James Bottomley thought that anything they wrote wouldn't last long, as the vendors would move the code into the drive. Eric Reidel (Seagate) objected, saying that they need to determine the boundary between the drive and some amount of software. That boundary needs to take full advantage of the technology, covering the limitations and exposing the benefits. Eric encouraged people to think about this envelope of some hardware and some exposed interfaces.

Someone suggested using XFS's block allocation mechanism. Ted Ts'o said that if we could solve multiple problems at once with Dave Chinner's block allocation scheme, it would give us a lot of power. Both ext4 and XFS have block allocation maps that keep track of both logical and physical block addresses, but keeping track of physical block addresses relies on getting information back from drives.

Sage Weil (Inktank) wondered how many hints the drivers would need to send to drives; For example, block A should be close to B, or A will be short-lived. James Bottomley mentioned that most people assume that the allocation table is at the beginning of the disk. Erez Zadok replied that if we can produce a generic enough abstraction layer, lots of people will use it. It could be used with SMR, but also with raw flash.

Error messages are another issue. Erez said that some drives produce an error if the block has not been written before being read, and someone else said that the driver could return all zeroes for initialized blocks. Erez said that currently, writing to the first block in a zone automatically resets all blocks in that zone, but is that the correct behavior? Bernard Metzler (IBM Zurich) wondered whether it's best to expose the full functionality of this storage class, as was done with flash by Fusion-io. In 10 years, non-volatile memory (NVM) will replace DRAM, so should we treat NVM like block devices or memory?

The mention of NVM quickly sidetracked the discussion. Christoph Hellwig suggested that addressing NVM should not be an either-or decision, while Ric Wheeler suggested that the user base could decide. James Bottomley wondered about error handling of NVM. Someone said they had tried using NVM as main memory, but it was still too slow, so they tried it via PCI. And they did try mmap, and the performance was not very good. Christoph responded that what they have called mmap needs to be fixed. The idea is to allow writes directly to a page in memory and use DMA for backing store.

James Bottomley pointed out that putting NVM on the PCI bus causes problems, because it makes it cross-domain. Ric Wheeler said this was a good example of the types of problems we need to know more about in that vendors can't talk about what they are doing, but then kernel developers and researchers don't know how to prepare for the new technology. Ted Ts'o said that he knows that Intel is paying several developers to work on using NVM as memory, and not on the PCI bus.

Erez Zadok said that this would be the first time we had a byte-addressable persistent memory, with a different point in the device space. He hoped it would not wind up like flash, where there are very limiting APIs controlling access.

After another short break, two students from the University of Wisconsin, Vijay Chidambaram and Thanumalayan Sankaranarayana Pillai, along with Jeff Darcy (Red Hat), addressed the group. They have a shared interest in new ways to flush data from in-memory cache (pages) to disk, although for different reasons. Vijay started the discussion by outlining the problem they had faced when needing to have ordered file system writes. The normal way of forcing a sync is via an fsync call on a particular file handle, but this has side effects. Sage Weil asked if they were trying to achieve ordering in a single file, and Vijay said yes. They had added a system call called osync to make ordered writes durable and had modified ext4 and were able to do this in a fairly performant way. Christoph Hellwig said that there really was no easy way of doing this without a sync, and that no one actually had written a functioning fbarrier (write data before metadata) call. Vijay said that they had created a new mode for a file, and Ted Ts'o asked if, when they osync, it involves a journal command. Vijay responded that the osync wraps the data in a journal command but does not sync it.

Jeff Darcy, a lead programmer for Gluster, said that he just wants to know what ext4 has done, because they need the correlation between user requests and journal commits. Christoph said they could easily do that in the kernel because each time they commit, there is a log sequence number. They could wait for the return from disk and use the log sequence number to implement osync. Ted said that it's the name that's the problem, like a cookie to identify the osync write. Jeff said that getting a cookie back that can be used to check if a write has been committed would be enough for their purposes.

Vijay stated that their main concern was knowing when data has become durable (not that different from Darcy's concern). Kai Shen (Rochester) said he liked the idea, but that it's not easy to use in comparison with atomicity. Kai had worked on a paper about msync, for doing an atomic write to one file. Ric Wheeler pointed out that Chris Mason (btrfs lead) has been pushing for an atomic write, which has not reached upstream (submitted for a kernel update) yet, but is based on work done by Fusion-io. Vijay complained that with ext4, you have metadata going to the journal and data going to the disk, but they need a way of doing this atomically. Sage Weil pointed out that there is no such thing as a rollback in file systems. Jeff Darcy said that all distributed databases have the same problem with durability.

Thanumalayan finally spoke, saying that they had discovered people doing fsync after every write and had been searching for patterns of fsync behavior themselves. When they shared the patterns they had uncovered with application developers, the developers' convictions about how things work were so strong they almost convinced him. The room erupted in laughter. On a more serious note, Thanumalayan asked, if he does a number of operations, such as renames, do they occur in order? Andreas Dilger said they don't have to, but get bunched up. Ted Ts'o mentioned that an fsync on one file implies that all metadata gets sync'd in ext4 and XFS, but not zfs or btrfs. However, that could change, and we might add a flag to control this behavior later.

Ric Wheeler liked this idea and suggested having a flag for async vs. fsync behaviors and being able to poll a selector for completion. Christoph Hellwig thought this wouldn't buy you much with existing file systems, but Ric thought it was worth something. Ted Ts'o said they could add a new asynchronous I/O type, and Christoph said that AIO sync has new opcode, IOCB_CMD_FSYNC, exposed to user space, that is not "wired up." Ethan Miller said it would be nice if you could request ordered writes. Greg Ganger (CMU) explained that there's a good reason why databases use transactions, and that they had tried doing this, and it's really difficult.

Erez Zadok commented that it's interesting to listen to the comments, as he has done 10 years of work on transactional storage, worked with umpteen PhDs on theses, and so on. The simplest interface to expose to users is start transaction/end transaction. To do this, you must go through the entire storage stack, from drivers to the page cache, which must be flushed in a particular order. Once he managed to do all that, he found that people didn't need fsync. Vijay piped up with "that's what we've been trying to do," and Sage Weil said they had tried doing this with btrfs and wondered if they ever got transactions to work. Erez replied that they had gotten transactional throughput faster than some, but Thanumalayan jumped in to say they had run into limits on how large a transaction could be. Vijay said that if you start writing, then use osync—it works like a transaction.

Ric Wheeler then commented that this is where kernel people throw up their hands and say show us how easy this is to do. Ted Ts'o wondered why it couldn't be done in user space (FUSE), and Ric replied that they need help from the kernel. Sage Weil had tried making every transaction checkpoint a btrfs snapshot, so they could roll back if they needed to. Jeff Darcy said that if you need multiple layers working together, you will have conflicting events. Thanumalayan just wants a file system with as much ordering as possible. He was experimenting with what happens to a file when a crash occurs, and said that POSIX allows a totally unrelated file to disappear. Christoph Hellwig shouted out that POSIX says nothing about crashes and power outages, and Jeff agreed that POSIX leaves this behavior undefined.

Jeff claimed he would be happy if he could get a cookie or transaction number, flush it with waiting, and select on that cookie for completion. Greg Ganger suggested he actually wanted more, to commit what he wants first. Vijay explained that the NoSQL developers they've talked to don't need durability for all things, but for some things, and that they have code, used for an SOSP 2013 paper, as an example. Jeff thought that NoSQL developers make assumptions about durability all the time, and about how fsync works. Ted Ts'o thought that all they needed to do was add new flags, or perhaps a new system call. Christoph Hellwig pointed out that there is sync_file_range, for just synchronizing file data within the given range, but not metadata, so it isn't very useful.

Kai Shen suggested that perhaps fsync should work more like msync, which includes an invalidate flag that makes msync appear atomic. Christoph said that older systems didn't have a unified buffer, so you had to write from the virtual memory system to the file system buffers. Ric Wheeler mentioned user space file systems, and Sage Weil suggested that it would be nice if you could limit the amount stored in the caches based on cgroup membership. James Bottomley stated that the infrastructure is almost all there for doing this with cgroups already.

Sage brought up another problem: that when a sync occurs, the disk gets really busy. Because of this, they (Inktank) actually try to keep track of how many blocks might be dirty so they can guess how long a sync might take. Jeff Darcy concurred, saying that they do something similar with Gluster. They buffer up as much as they can in the FUSE layer before pushing it into the kernel via a system call. James said that the Postgres people want the ability to manage write-ahead as well. Christoph clarified this, saying that they want to use mmap for reads, but control when to commit data (write) when it changes. They don't want the kernel storing the data on its own, but rather they want a backing store in place. This would be like mmap private.

James Bottomley said that the problem is related to journal lock scaling, when you want a lock for each subtree. You want to make the journal lock scale per subtree. Ted Ts'o pointed out that the last paper at FAST [2] covered this very topic. James pointed out that this would be a problem for containers, which are like hypervisors with only one kernel (like Solaris has in some form).

Kirill Korotaev (Parallels) then went to the front of the room to introduce another topic: FUSE performance. Most people think FUSE is slow, said Kirill, but they have seen up to 1 GB/s in real life. After that, bottlenecks slow more performance gains. To get past that, they need some interface to do kernel splicing and believe that would be quite a useful interface. Copying using pipes is very slow because pipes use mutex locks, and pipes don't work with UNIX sockets. What they basically need is the ability to do random reads of data with these buffers and to send them to a socket. Kirill also questioned why there's a requirement that data must be aligned in memory. Andrew Morton said that was a very old requirement for some devices, and James Bottomley said that's why there is a bounce buffer for doing alignment under the hood.

Kirill wanted to revive IOBuff, where they could attach to pages in user space, then send them to sockets. James pointed out that except for DirectIO, everything goes through the page cache, and moving data from one file to another is hard. Ric Wheeler asked if the interface was doable, and Kirill said that they think it is. Andrew Morton wondered if this was different from sendfile, and Sage Weil thought perhaps splice would work as well, but Kirill responded that neither work as well. Sage thought the problems could be fixed, but Kirill ended on the note that if it were easier, it would have been done before.

George Amvrosiadis (University of Toronto) introduced the topic of maintenance and traces. For durability, storage systems perform scrubbing (background reads) and fsck for integrity, but today those things need to be done online. The issue becomes how to do this without disturbing the actual workload. For scrubbing in btrfs, you get an upper bound on the number of requests that can be processed during scrubbing. Ric Wheeler, who worked at EMC before moving to Red Hat, asked how often do you want to scrub, and how much performance do you want to give up, and for how long.

George Amvrosiadis wanted to monitor traces and then use the amount of activity to decide when to begin scrubbing during what appeared to be idle time. Ric Wheeler said that while he was at EMC, he saw disks that were busy for years, and the only "idle time" was when the disks performed self-checks. Andreas Dilger asked if there was an idle priority, and George said they wanted to do this for btrfs. The problem is that all requests look the same, as maintenance requests look like other requests. Ric responded that they need a maintenance hint. Ted Ts'o said you'd need to tag the request all the way down to the block device layer, orthogonal to priority. George then stated that all they want to do is optimize when to schedule scrubbing. James Bottomley said they already have a mechanism for increasing the priority of requests, and perhaps they could also support decreasing priority. Ric replied that this sounded like the "hints" stuff from last year's Linux Filesystem Summit. If the file system supports it, although we can't guarantee it, we can at least allow it, so this sounds easy.

Ethan Miller wondered if you have devices with built-in intelligence, do you really want both the device and the kernel doing scrubbing? Ric answered that you want to scrub from the application down to the disk, checking the entire path. George Amvrosiadis then asked if scrubbing could cause more errors or increase the probability of errors. Ethan said that scrubbing has no effects; it's just reading. But Eric Reidel pointed out that all disk activity has some small probability of causing a problem, like smearing a particle on a disk platter. These probabilities are small, compared to the MTBF for a drive.

George Amvrosiadis still wanted some hint from the file system that work was about to begin. This brought up a tangent, where Ric Wheeler pointed out that batching up work for a device has caused problems in the past. Andreas Dilger explained that if just one thread were writing, you could get 1000 transactions per second, but if the application were threaded, and two threads were writing, the rate would go down to 250 per second. The problem occurred because the file system would wait a jiffy (4 ms at the time) trying to batch writes from threaded applications.

Greg Ganger said that there is a temptation to do all types of things at the SCSI driver level so it can do scheduling, and Ric Wheeler responded that the storage industry has been looking at hints from the file system for a long time. James Bottomley replied that the storage industry wanted hints about everything, a hundred hints, and Ted Ts'o added, not just hints, but 8–16 levels for each hint.

George Amvrosiadis then asked about getting traces of read/write activity, and Ethan Miller agreed, saying that they would like to have interesting traces shared. Ric Wheeler said that people have worked around this using filebench, and Greg added that it would be nice to have both the file and block level trace data.

Erez Zadok, the co-chair of FAST '15, said that USENIX is interested in promoting openness, so for the next FAST, when you submit a paper you can include whether you plan on sharing traces and/or code. Christoph Hellwig said just include a link to the code, and Ted Ts'o added that could be done once a paper has been accepted. Ethan Miller said that anonymization of traces is really hard, and they had experience working with NetApp on wireshark traces for a project in 2013. Erez pointed out that HP developed a standard, the Data Series format, and also developed public tools for converting to this format. He continued saying that EMC plans on releasing some traces they have been collecting for several years, and that a past student, Vasily Tarasov, had spent a week at their datacenters collecting statistics.

Several other topics were covered during the final hour: RDMA, dedup, and scalability, but your correspondent missed this in order to catch a flight to SCaLE 12x in Los Angeles.

I did appreciate getting to watch another summit in progress, and was reminded of evolution. The Linux kernel evolves, based on both what people want, but even more on what contributors actually do. And, as with natural evolution, most steps are small ones, because changing a lot at once is a risky maneuver.

References

[1] Tim Feldman and Garth Gibson, "Shingled Magnetic Recording: Areal Density Increase Requires New Data Management," *;login:* vol. 38, no. 3 (June 2013): https://www.usenix.org/publications/login/june-2013-volume-38-number-3/shingled-magnetic-recording-areal-density-increase.

[2] Junbin Kang, Benlong Zhang, Tianyu Wo, Chunming Hu, and Jinpeng Huai, "MultiLanes: Providing Virtualized Storage for OS-level Virtualization on Many Cores": https://www.usenix.org/conference/fast14/technical-sessions/presentation/kang.