Conference Reports

In this issue:

10th USENIX Conference on File and Storage Technologies (FAST '12)

San Jose, CA
February 14–17, 2012

Opening Remarks and Best Paper Awards

Summarized by Rik Farrow (rik@usenix.org)

Bill Bolosky began the conference with the statistics. FAST 2012 set lots of records: largest number of submissions and accepted papers, lowest acceptance rate, and largest number of attendees. Bill wondered if it was just that the economy is getting better, or that the conference is that popular.

They also tried something new this year: short papers that are refereed the same way as longer papers. Bill then showed an image representing the words found in paper subjects and abstracts. Obvious words, such as “storage” and “system,” were most prominent, followed by “file,” “data,” “deduplication,” “flash,” and “performance” (approximately—check out the video). “Cloud” is still a tiny word, but Bill expects that will grow.

Jason Flinn took over and described the first Test of Time award, for ideas that appeared at FAST over 10 years ago; it was presented to Sean Quinlan and Sean Dorward for “Venti: A New Approach to Archival Storage.” Next, he announced the Best Paper awards: “Recon: Verifying FS Consistency at Runtime,” by Daniel Fryer et al., and “Revisiting Storage for Smartphones,” by Hyojun Kim et al.

Keith Smith of NetApp and Yuanyuan (YY) Zhou of UCSD will be the chairs of FAST '12.

Implications of New Storage Technology

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

De-indirection for Flash-based SSDs with Nameless Writes

Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Yiying Zhang presented her research on a new "Nameless Writes" interface to solid state disks (SSDs). SSDs generally include a Flash Translation Layer (FTL) that maps logical to physical addresses. This allows the device to perform wear-leveling behind an opaque address space. However, this indirection incurs mapping table space cost and performance overheads. Zhang proposed eliminating these overheads by largely removing the address translation table and storing physical data addresses directly in the file system. In the proposed interface, file systems do not specify a logical address when issuing a write. Instead, the SSD acknowledges completed write requests with the data's physical address.

Realizing a nameless writes interface required addressing a number of problems. Writes are always placed in new physical locations, which forces metadata modifications to cascade address updates to new physical references up the length of the file system tree. To address this problem, Zhang uses a traditional write interface for file system metadata, including traditional on-device address translation. Since file systems generally store far more data than metadata, the cascading updates can be eliminated while preserving most performance and cost advantages. Physical address migration is also a challenge. With nameless writes, SSDs move data beneath running file systems, just as they do today. This requires that callbacks be sent to the file system informing it of any planned moves, while a temporary mapping table in the SSD ensures that requests are routed appropriately.

To evaluate their system, Zhang and her team created an SSD emulator, ported ext3 to the nameless writes interface, and evaluated against a page-mapped FTL, a hybrid FTL, and a nameless-writes FTL. The effort required 4360 lines of code changes to ext3. Their SSD emulator operates as a pseudo-block device and stores results in memory. They found that their nameless writes indirection mapping table required only 2%–7% of the metadata overheads and performed up to 20 times better on a workload of random writes.

Following the talk, Geoff Kuenning from Harvey Mudd College asked the community why we don't simply write a file system that is designed specifically for the SSDs, and remove the FTL entirely. Zhang replied that she thought the device should control its own wear leveling. Margo Seltzer asked if the space savings from removing the FTL mapping table is offset by the temporary mapping tables. Zhang explained that the overall metadata requirements are much smaller. Keith Smith from NetApp asked whether the authors had considered a richer interface, an idea that Zhang thought was promising. Ethan Miller from UCSC followed up on Smith's question to note that an object interface would provide the same benefits, even though it moves much of the management from the file system to the device.

The Bleak Future of NAND Flash Memory

Laura M. Grupp, University of California, San Diego; John D. Davis, Microsoft Research, Mountain View; Steven Swanson, University of California, San Diego

Laura Grupp presented her team's ominously titled paper, "The Bleak Future of NAND Flash Memory." Their goal is to project the evolution of NAND-based flash into the year 2024 to determine if the current reliability and performance will be derailed by technical limitations. Their findings are mixed. By some metrics, flash will continue to improve, but in other ways it will decline.

It is widely understood that current NAND-based flash drives are fast and reliable but have a relatively high cost-to-capacity ratio. Moving forward, capacity will no doubt increase, but with current processes and technologies, the increased density will incur increased error rates and decreased performance. To anticipate the future of NAND flash, Grupp and her team combined measurements of modern flash architectures with projected trends in manufacturing to model the capacity, latency, and throughput of flash going into the future.

Grupp explained that capacity will increase with the bit density of each cell. Current technologies include single-level cells (SLCs), which store a single bit; multi-level cells (MLCs), which store two bits; and triple-level cells (TLCs) which store three bits but are not really triple-level cells (they have eight levels). Cell size decreases by scaling, following Moore's Law. Current processes are between 25 nm and 34 nm, with industrial working groups predicting 6.5 nm by 2024. These factors suggest a 43-fold increase in capacity over that period. To test performance, Grupp and her co-authors used an in-house testing system to analyze 45 flash chips from six companies with a variety of bit densities and manufacturing processes. Fitting the results to an exponential curve suggests a twofold increase in latency for every order of magnitude increase in density. In concluding, Grupp noted that we can improve density and cost, but performance and reliability will decline.

Nauman Rafique from Google asked why the authors consider the provided scenario to be bleak. Grupp replied that consumers are accustomed to technologies improving, but we will not see this with flash. Michael Jadon from Radian Systems was optimistic that future precision improvements in voltage measuring would lower SSD latencies, but Grupp reiterated that the model only tracks current trends and does not include assumptions about future discoveries. David Rosenthal from Stanford University added that there is insufficient manufacturing capacity for flash to completely replace magnetic disks anyway, and that many of the limitations discussed apply to other technologies, such as Memristor. Abhijit Paithankar from VMware asked if the authors studied power consumption, but they had not considered this extensively. Kirk McKusick asked how the memory lifetime changes as we move to MLC and TLC. The author referred him to the paper, saying "It's a dramatic decline." TLC will only survive 500 program-erase cycles per block.

When Poll Is Better than Interrupt

Jisoo Yang, Dave B. Minturn, and Frank Hady, Intel Corporation

Jisoo Yang explained how the next generation of NVRAM will see interrupt overhead as a major source of latency. His work seeks to quantify the costs and to reduce them.

Conventionally, disks use a hardware interrupt to notify the scheduler when an I/O operation completes. The scheduler correspondingly wakes the thread waiting on that operation. This implies that between requesting and completing the operation, the requesting thread loses its context, freeing other threads in the application to do work. With a very low latency device, Yang argues, the overheads of handling this asynchronous I/O will start to dominate. It may be more efficient for the CPU to directly poll the device for completion. To test the claim, the team experimented with prototype hardware. They measured a DRAM-emulated PCIe SSD using the new NVM Express interface. They found that traditional interrupt-driven I/O had a 6.3 microsecond latency, while polling had only a 4.4 microsecond latency. Further, the asynchronous nature of interrupt-driven I/O left only 2.7 microseconds for an application to make any progress, limiting the benefit of asynchronous I/O.

John Groves of Dell Storage asked whether it would be better to have a polling loop in place of the entire interrupt handler. Yang clarified that each CPU in their implementation has a dedicated polling loop. Yan Li from UC Santa Cruz asked how the results should be interpreted in light of the preceding talk, "The Bleak Future of NAND Flash Memory." Yang responded that he expects the next-generation processes to move away from NAND and to make significant improvements in performance. Kai Shen from the University of Rochester noted that even with polling, OS overhead is substantial. He wondered if it is worthwhile to remove the OS completely, as is done today with GPUs. Yang said it was an interesting and likely effective approach, but that it would break the block interface, which has value. Steven Swanson from UC San Diego asked if the authors see potential benefit in increasing concurrency by using a polling thread with multiple outstanding requests. Yang believed the results would depend on the application logic. In some cases it might benefit, so it's a scenario worthy of further consideration.

Back It Up

Summarized by Yiqi Xu (yxu006@cs.fiu.edu)

Characteristics of Backup Workloads in Production Systems

Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu, EMC Corporation

Fred Douglis started by highlighting some special characteristics of a backup system, e.g., it stores the data using aggregation instead of small files. Other characteristics include that backup data is replicated and that the backup data on a weekend usually has full 100 GB tar-type, while workdays have incremental 1 GB tar-type. He pointed out that analysis of primary storage abounds, while there is little characterization of backup systems. Their work can validate past design decisions using more extensive data and provide data for future analyses. The work is also motivated by the fact that it is predicted that there will be eight exabytes of data on disk-based, purpose-built backup appliances by 2015. Their two-pronged analysis covers a study of both breadth and depth, with statistics from over 10,000 systems and using detailed metadata traces from several production systems storing almost 700 TB of backup data.

Fred compared backup file size to primary storage file size. The former is orders of magnitude larger, so that traditional optimizations do not work for backups. Backup files also have many fewer files and directories, as well as flatter hierarchy because of many files per directory, and backup systems also use catalogs. The weekly churn is around 20%, so the system should be able to reclaim data on a regular basis. That's why deduplication helps. Primary data deduplication is reported to be 3x–6x, while backup data is >60x dedupe for some, 384x max. He went on to sensitivity analysis of chunk size and cache size. With many different kinds of data sets, he proposed merging chunks to analyze deduplication rates across a range of chunk sizes without having to access the whole content. They used content-defined merging and considered the overhead of metadata with smaller chunks. The rule of thumb is 15% better deduplication rate for each smaller power of 2 in chunk size, but about 2x the metadata. The best deduplication chunk size is 4 KB, and 8 KB considering maintenance and cleaning. For caching, they proposed replaying traces with varying cache sizes and reported results on the warm cache. The results show that for writes, chunk-level LRU caching needs large chunks to be effective for writes and that container-level LRU caching works well. Read cache behavior is similar but for much larger caches, due to data caching.

Fred covered the related work on deduplication and data characterization and concluded that high churn means throughput must scale with primary storage capacity growth. Backup systems are very different from primary systems. They need high locality and deduplication for hit rate high performance; 8 KB is a sweet-spot chunk size.

Dutch Meyer asked if 8 KB chunk average is an effective average they would measure or a statistical average they expect to get (it makes a difference if chunks break on zeros very often). Fred said that they are actually pretty close, that their system doesn't do anything special about zeros, and that if all zeros on a block define a block boundary, then it's going to cause a problem. Primary storage often has many files whose last chunk is smaller than the rest. If we get TBs of data on one file, then the last chunk is just noise. Dutch Meyer did set chunk boundary on zeros when he did his work and found it interesting that EMC doesn't. Dutch said it's really dependent on what the average means, and they took the discussion offline. Geoff Kuenning asked if data is quite anonymized and can they have it. Fred replied that they cannot make any promises and cannot join in a repository, at least for the foreseeable future, but they welcome interns to join them and have access to the data and work on it. Someone asked whether storage speed depends on chunk size. The answer was no. Arkady Kanevsky from Dell asked, if database churn is very quick in the dataset, is that characteristic or anomalous? Fred reviewed the slides and said it's an indication of how long the user would keep the backup, and it's really a choice of the user. John Groves asked if they are chunking with bins and Fred answered yes.

WAN Optimized Replication of Backup Datasets Using Stream-Informed Delta Compression

Philip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu, EMC Corporation

Philip Shilane started his presentation by showing the demands and challenges in improving offshore replication performance. Afterwards he talked about his idea and demonstrated an example of deduplication, with delta compression sketches matching similar chunks. He compared his work by searching through several sketch index options (full index, partial index, and stream-informed cache) and analyzing their advantages and disadvantages. He further exemplified his idea by animating a discovery of similarity in data. Philip compared the two approaches, replication with deduplication and replication with deduplication and delta compression. He discussed the properties of stream-informed delta compression, its pros being fast compression of data and small memory footprint, and cons being dependence on locality and cache, and some resource cost. The data set he used is very different from the previous one: multiple months' backups with varying sizes. The results from delta compressions are shown on multiple datasets, compared with a full index simulator. The results show that two super features are better than an index with one. More features do not necessarily improve compression, because of a fixed cache size. The results also show that one feature compression is within 14% of using an index. As a result, delta improvement is from 1.8x to 3.1x and the effective network throughput is 1–2 orders of magnitude faster than the old approach without delta compression.

Overhead and limitations were also discussed, with more space requirement per chunk and more CPU and I/O consumption on the source and destination. The sketching is also claimed to slow down the writes for non-duplicates by 20% and scales linearly with the number of streams at the destination. They also discussed compression loss because of shared caching size and the real case results from customers. Philip concluded his talk by listing the related works and stating that delta locality closely matches deduplication locality for backup datasets; in addition, the work has low cost and good scalability, and it allows customers to protect twice as much data by moving it across a WAN. Cristian Ungureanu from NEC asked why 3.1x compression improvement results in 1–2 orders of magnitude in network throughput. Philip answered that it's due to the performance variation in non-delta compressions.

Power Consumption in Enterprise-Scale Backup Storage Systems

Zhichao Li, Stony Brook University; Kevin M. Greenan and Andrew W. Leung, EMC Corporation; Erez Zadok, Stony Brook University

Zhichao Li claimed that disk backup is a prime candidate for power management, but there is no previous power measurement research and so assumptions are often made that disks will dominate power. The authors measured four enterprise backup controllers and two kinds of enclosures. They used a power meter for accurate measurement while excluding other factors such as networking, cooling, and internal subcomponents. First, Zhichao presented numbers for idle power consumption—when deduplication is being performed, CPU and RAM cost power. He used watts per TB to measure the different models and found that newer controller models are more efficient. The case for enclosures is similar. He concluded that deduplication saves space, because it saves hardware such as the controller/enclosure and networking devices. It also reduces disk I/Os. Zhichao went on to disk power management—spin down/power down of disks has limited savings on power consumption. Other components in the enclosure drain more power. Disk spin-down at scale also demonstrates that in order to save power, many enclosures in a controller need to spin down their disks. He then looked at power proportionality in the controller. The results show that power varies more by model than by workload, because the controller consumes more power than needed. And in the enclosure the percentage power increase is less than the workload change, which proves again that controller/enclosures consume more power than disks.

Zhichao's conclusions were: (1) controller/enclosures are power hungry, (2) current systems are not power proportional, and (3) new hardware is more efficient. Future work will focus on aged backup, primary storage, CPU, and RAM power consumptions when built-in sensors are available.

Brent Welch from Panasas asked how realistic it is for the controller to populate 50 enclosures, because the controller also has to do RAID to protect drives. At what point does the deduplication controller spend more time on RAID rebuilds than on decompression? Will they really recommend that customers load 50 enclosures? Zhichao replied, it depends on whether the customer wants larger capacity with lower power cost, and how much customers are willing to pay.

File System Design and Correctness

Summarized by Yiqi Xu (yxu006@cs.fiu.edu)

Recon: Verifying File System Consistency at Runtime

Daniel Fryer, Kuei Sun, Rahat Mahmood, TingHao Cheng, Shaun Benjamin, Ashvin Goel, and Angela Demke Brown, University of Toronto

Awarded Best Paper!

Daniel Fryer presented work on an online filesystem-checking layer between the file system and block layers. He showed us that metadata is important while fragile in file system consistency. And metadata is more error-prone because of file system bugs and will cause data loss. Current solutions rely on the correctness of file systems such as journals, checksums, and RAID. Offline checking is often used, but it is slow, requires taking the file system offline, and produces error-prone repair.

Daniel's team proposes making sure that every update results in consistency. However, consistency properties are global and may require a full scan run; furthermore, fsck at every write is not possible. Thus, fast, local consistency invariants are introduced to keep data consistent before it becomes persistent. Daniel demonstrated an example in ext3, where the block bitmap and the block pointer should agree with each other. He further explained that the check happens on the transaction boundary just before the commit block reaches the disk. The design depends on the interpretation of metadata and comparison with the old version, without depending on the agnostic file system. He proposed an interface to invoke for different types of file systems. The write cache is used for delaying the update, and the read cache is for storing the hot cache of the read metadata.

Daniel discussed the evaluation of the implementation, detection effectiveness, and performance overhead. They simulated metadata error corruption by injecting wrong metadata before it was written. The errors they catch are inclusive of all the errors found by e2fsck except for a special flag that isn't being used in ext3. Cache size also affects one of the benchmarks evaluating throughputs, because the 128 MB cache size takes away some cache from the system.

Wenguang Wang from Apple asked what happens after catching an inconsistency; is stopping all I/O an option? Yes, they hold and fail stop. Several other options are possible, including stopping all writes, remounting read-only, taking a snapshot and continuing, and micro-booting the file system or kernel. Ethan Miller from UC Santa Cruz asked what kind of issue to expect when delayed commit is implemented in the FS. Daniel responded that ext3 also group commits with 5–10 thousands of blocks at a time. Keith Smith from NetApp asked for advice for future file system writers to make checking easy. Daniel said they like Btrfs, with back pointers that require less data to track; he also recommended figuring out and writing consistency problems in a declarative language like SQL. Someone asked if this kind of delay is tolerable in synchronous, production workloads with increased latency for commit block; they took the discussion offline. Atul Adya from Google asked if they considered applying this technique to other applications such as distributed file systems. Daniel said that they thought about this and will probably find transaction models and distributed invariants. Chris from Nebula asked why they do this work in the block layer. He maintains a subset of this kind of code for xfs within the file system. Daniel admitted that it is more practical for real systems this way, but placing the function on the block layer also has the benefit of taking it out of the kernel and placing it with the hypervisor if they don't trust the operating system, and that filesystem format doesn't change quickly.

Understanding Performance Implications of Nested File Systems in a Virtualized Environment

Duy Le, The College of William and Mary; Hai Huang, IBM T.J. Watson Research Center; Haining Wang, The College of William and Mary

Duy Le started by pointing out an inadequacy in the investigation of impacts of nested file systems. Existing literature exists around I/O scheduler, storage allocation, and virtualized FS on the virtual machines. However, assumptions made on one layer of file system may hurt two-layer schemes. The authors combine six file systems (ext2, ext3, ext4, reiserfs, xfs, and jfs) as host and guest file systems to find the best combination with varied I/O behavior and interactions.

In their setup, they partitioned the physical disk into equal partitions and formatted six of them to six different file systems, which in turn used a flat file to act as disk image for a virtual block device. The last partition was used as a direct block device for baseline measurement.

The results show that guest file system and host file system choice are bi-directionally affecting each other in performance. While writes are more critically affected by the additional layer, read sometimes can achieve even better performance. Latency is sensitive to nested file systems. Duy then zoomed in on some specific combinations of guest and host file systems for detailed analysis on reads and writes. He showed some findings, including the effectiveness of guest file system block allocation, I/O scheduler's effectiveness on the guest file system, and journaling performance impact. Finally, he listed five pieces of advice for file system choice/tuning for different workloads and circumstances.

Erez Zadok from Stony Brook University asked why they did not shuffle the host file systems on each of the partitions. This could result in zone-constant angular velocity, which is said to have up to 25% marginal variance on performance. Duy replied that they have demonstrated a less than 5% performance difference and determined that it is a negligible factor in the evaluation, since 42 combinations will cost a lot more in time. Zadok then asked whether it will be different when they put different kinds of workloads/access patterns on the partitions. John Groves from Dell asked whether the container file is pre-allocated (yes) and did they use direct I/O to avoid upper-level I/Os going to page caches of the underlying file system (yes). Ric Wheeler from Red Hat commented that NOOP is often used in the upper level in a virtualized environment. Duy said that guest CFQ / host deadline was the best combination they found, so they tended to use this setting. Dutch Meyer (University of British Columbia) asked whether images are raw (yes) and can the findings/approaches be generalized at the disk management layer. The topic was taken offline. Was disk flushing disabled for accurate measurement? They made sure all caches got flushed.

Consistency Without Ordering

Vijay Chidambaram, Tushar Sharma, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

Vijay Chidambaram pointed out that crash inconsistencies are caused by ambiguity about logical object identity. He showed a crash scenario in which two files claim a shared block. He then introduced No-Order File System (NoFS), which uses back pointer-based consistency, where owners of objects are found through the back pointer; the important assumption is that object and back pointer are written atomically. He revisited the crash scenario to show how using a back pointer works. He then showed how allocation structure consistency is maintained. The creation of new objects can proceed without complete allocation information. The validity bitmap is used to track checked objects. He elaborated the case with a scenario determining allocation information. Implementation details include two threads responsible for metadata and data scan in the background. Note that scheduling of scans can be configured while idle or periodically.

In evaluating NoFS, Vijay answered three questions: is it robust? how much is the overhead? and does the background scan lower performance? They put a pseudo device driver between the file system and disk to discard writes to selected sectors to simulate crash. As a matter of fact, NoFS detected all the inconsistencies, and orphan structures were reclaimed. There proved to be minimal overhead in terms of throughput, and ext3 demonstrates lower throughput in cases where there are order points in writes. However, scan reads interleaved with file system I/Os really affect throughput, and accesses on non-verified objects have penalties, such as stat. Vijay concluded that trust is implicit, and removing it is key to robust, reliable storage systems.

Peter Macko from Harvard asked about the memory overhead of this approach. Vijay replied that one extra block for the bitmap is acceptable. Bill Bolosky from Microsoft Research asked how big the file system is. Vijay answered that it's a 50 G partition with 1–5 GB of data. Bill responded that a 3 TB surface scan takes 4–5 hours. How does this system work when it's close to full? Does it wait a long time for writing (finding a free block)? He mentioned the work he was doing on distributed file systems, which are often full. Ashvin Goel from the University of Toronto asked if this approach works for all disks. Vijay answered, as long as the back pointer is atomically written together with data block on the disk, it can be achieved on the device. Someone asked where the back pointer is stored. Vijay answered that in the implementation it is inside the data block. But when out of band area can be used, they can store it there. Keith Smith from NetApp commented that if a file system grows big enough, it will not be able to handle the extra block. He asked if files created on disk can be in the same order as created in the applications. Vijay said no, since it's not the consistency they provide. Dave Anderson from Seagate Technology recommended that Vijay apply this technique to other problems. Vijay said yes, it's applicable to hierarchy problem domains. Brad Morrey from HP Labs asked about the extra CPU cycles this approach introduces. Vijay answered that there is no noticeable increase, because the check occurs between the disk and memory. Someone from Seoul National University asked how the back pointers are removed when deleting files. Vijay said their approach is lazy deletion. He talked about having version number consistency if necessary. Vijay said that only bitmaps are kept in memory and they don't keep any structures, so there is no need to free many objects.

Flash and SSDs, Part I

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

Reducing SSD Read Latency via NAND Flash Program and Erase Suspension

Guanying Wu and Xubin He, Virginia Commonwealth University

Guanying Wu presented his research on making the program-erase cycle of NAND flash memory suspendable. To rewrite a page of NAND flash memory, the drive must complete a program-erase cycle, which first erases the page and then reprograms it. The erase operation is performed with a long pulse of erase voltage to expel electrons from the cells. The subsequent programming requires a series of charges in which a short pulse is attempted, then tested. This programming is retried with successively higher voltage pulses until it is successful. This process can be 10x–100x slower than a read, which only requires measuring the cell voltage. The problem is that a program-erase cycle blocks out all reads to the chip where the block exists, causing higher latency.

To improve read performance, Wu and his team developed a method for making the program-erase cycle suspendable. This allows read requests that arrive during a lengthy program-erase operation to be quickly fulfilled and the rewrite to be resumed later. This suspension may occur at different points in the program-erase cycle, each requiring a different mechanism. If suspension is needed during a program operation, suspension occurs between program pulse and verify operations. During the erase cycle, the erase operation is stopped and the duration remaining is noted. The erase can continue when the interrupting read request completes. In evaluating the system, Wu found that write latency increased by a few percent, which he considered trivial. Read latency decreased by 50% or more. He concluded that suspending the program-erase cycle for reads is a feasible solution, and one that significantly improves read performance.

Umesh Maheshwari from Nimble Storage noted that this approach seems to work best for lightly loaded systems, and Wu agreed. Dave Anderson (Seagate Technology) asked if flash lifetimes are degraded because of the extra wear involved in restarting program-erase cycles, but Wu had not performed that experiment at the time of the presentation.

Optimizing NAND Flash-Based SSDs via Retention Relaxation

Ren-Shuo Liu and Chia-Lin Yang, National Taiwan University; Wei Wu, Intel Corporation

Ren-Shuo Liu presented a technique for optimizing flash performance by opportunistically relaxing the requirements for longterm data retention. He noted that there are two main reliability specifications in flash: bit error rate and data retention time. The latter specifies how long the data should stay durable on stable storage. Error correcting codes (ECCs) are used to ensure that both criteria are met, but as flash density increases, the raw reliability degrades. This forces manufacturers to slow down writes to mitigate the worsening bit error rate and to strengthen ECCs.

However, data retention time of one year is overly conservative for the majority of data. Liu's team proposes a retention-aware FTL that initially relaxes the data retention specification to two weeks, and if data is not overwritten in one week, it can be reprogrammed with the stronger retention policy. The result is that long-lived data is eventually given the longer retention policy, but data overwritten within a week can use a weaker ECC to speed up writes by a factor of two.

To evaluate the approach, Liu used 11 workloads gathered from Microsoft Research Cambridge (MSR-C) and synthetic workloads, including TPC-C and Hadoop benchmarks, to estimate the average lifetime of a block of data. As a point of reference, in the MSR-C workload 86% of writes are overwritten in less than one hour. These workloads were evaluated on disksim 4.0 and SSDsim, using the retention-aware FTL design. The results suggest that a 2 to 5.7-fold improvement in performance is possible.

Several in attendance, including Sam Noh from Hongik University and Dave Anderson, asked Liu to clarify the durability assurances of long-lived data. Liu explained that background processes always convert long-lived data to normal retention mode, so there is no longterm relaxation of durability requirements. Another participant asked if any effort has gone into understanding the effects of the garbage collector. Liu responded that only writes originating from the guest use the weaker ECC. Data movement due to garbage collection always uses full ECC. Liu was also asked if the same performance improvement could be had if the controller was made more powerful, to accommodate stronger ECC, but that conversation was taken offline. Geoff Kuenning from Harvey Mudd College asked Liu to clarify the methodology around measuring the cleaning that occurs due to long-lived data being moved to a high retention cell. Liu assured Kuenning that the cleaning process was included in the simulation.

SFS: Random Write Considered Harmful in Solid State Drives

Changwoo Min, Sungkyunkwan University and Samsung Electronics; Kangnyeon Kim, Sungkyunkwan University; Hyunjin Cho, Samsung Electronics; Sang-Won Lee and Young Ik Eom, Sungkyunkwan University

Changwoo Min presented a new file system for solid state disks (SSDs) that's designed to address two fundamental limitations of these devices—random write performance and limited lifespan. As SSD technology matures, lifespan is a concern because each bit added to a memory cell decreases the number of accurate rewrites by an order of magnitude. Meanwhile, random write workloads significantly degrade performance and further shorten the lifespan of SSD.

Min described SFS, which is a file system specifically designed to remedy these problems. The file system employs a log-based design, which is suited to the unique characteristics of SSDs. By writing in a log structure and carefully grouping requests to match the size of an erase block, SFS effectively transforms random writes on the SSD into better-performing sequential writes and removes most internal fragmentation. When writing, it groups data according to hotness, so that future updates are less likely to require moving otherwise cold data. As a metric for hotness, SFS tracks write count and divides by the age of the block. It groups blocks together using an iterative refinement technique inspired by k-means clustering.

Min's team evaluated their system on three classes of SSD, using two synthetic workloads—TPC-C and a workload collected by UC Berkeley—and compared the results to ext3 and Btrfs. The results showed that SFS requires up to 6.1x fewer erase operations. The system is effective at making segment fullness bimodal, with more segments being either completely full or completely empty. As future work, Min is applying the file system to magnetic disk-based storage. His early investigations show some "promising results."

Min was asked if indirection at the file system level could cause additional write amplification. He answered that this is not what they have seen. In practice, total block erase counts are lower. Seungjae Baek from the University of Pittsburgh noted that grouping according to hotness has a long history and asked about comparisons to other schemes. Min directed Baek to comparisons in the paper. Umesh Maheshwari from Nimble Storage asked how the designers matched their segment size to that of the flash drive, and was particularly concerned about misaligned blocks. Min acknowledged that misalignment degrades performance, but the researchers had selected the file system parameters used by directly measuring the erase block size of their flash drives. Did Min's team have any plans to productize the file system? They were still discussing that possibility. Finally, one attendee asked whether the design depends on deferring writes until a full erase block is available, and cited sync operations in TPC-C as an example. Min responded that in sync operations, the system works as best effort.

Poster Session I

Summarized by Dulcardo Arteaga (dulcardo@gmail.com)

Only posters that were not represented by papers are summarized here. See static.usenix.org/events/fast12/poster.html for PDFs and descriptions of all posters.

CSPE: Cloud Storage Provisioning Decided by Rate of Return and Workload Characteristics

Jianzong Wang, Rui Hua, Changsheng Xie, Jiguang Wan, Yanjun Chen, Peng Wang and Weijiao Gong, Wuhan National Laboratory for Optoelectronics

This project presents a model that evaluates current workload on a cloud and its tendency to determine the benefit of purchasing/leasing new disks. They used the Internal Rate of Return (IRR) model used in economics to solve the "purchase or not" problem. They also used one module to detect workload peaks and another to trace the workload.

Reliable Energy-Aware SSD-based RAID-6 System

Mehdi Pirahandeh and Deok-Hwan Kim, Inha University

This project presents an approach for periodic estimation of reliability and energy consumption and a model of RAID6 that saves energy. The idea is converting pages into packages to reduce the amount of work during writes. Their evaluation shows that there is improvement in the energy consumption.

InnoDB DoubleWrite Buffer as Read Cache Using SSDs

Woon-Hak Kang, Gi-Tae Yun, Dong-In Shin, Yang-Hun Park, and Sang-Won Lee, Sungkyunkwan University; Bongki Moon, University of Arizona

Woon-Hak Kang presented this work to extend and move the double write buffer of InnoDB to an SSD to exploit the capacity and low latency of this kind of device. Besides the functionality of writing dirty pages to guarantee atomicity, they propose using the double-write buffer as a cache for random reads, consequently improving the performance of reading and for writes. The evaluation shows a significant improvement in performance for reads but not for writes when comparing the use of HDD to SSD.

Mitigating the Network Impact in Large Scale DFSs

Gustavo Bervian Brand and Adrien Lébre, Ecole des Mines de Nantes

This project evaluates the performance of different distributed file systems based on different topologies, comparing when data servers/metadata servers are located behind a

WAN/LAN. This evaluation attempts to demonstrate that there is a need to include the factor of topology when designing a distributed file system.

Their experiments compared a variety of topologies with different numbers of nodes, and their performance, based on different configurations, varies considerably, due to the overhead in the network traffic.

vPFS: Bandwidth Virtualization of Parallel Storage Systems

Yiqi Xu, Dulcardo Arteaga, and Ming Zhao, Florida International University; Yonggang Liu and Renato Figueiredo, University of Florida; Seetharami Seelam, IBM T.J. Watson Research Center

Yiqi Xu proposed vPFS, which adds to existing parallel file systems the ability to differentiate I/O requests from different applications, then meet per-application quality of service. This approach was implemented on top of PVFS, which is a user-level distributed file system. A proxy was created to intercept the I/O traffic and tag it according to the applications, and a scheduling algorithm is applied at that point to meet the quality of service.

Experiments show that using different applications with different QoS can meet application requirements without generating overhead in the I/O.

OS Techniques

Summarized by Daniel Fryer (dfryer@cs.toronto.edu)

FIOS: A Fair, Efficient Flash I/O Scheduler

Stan Park and Kai Shen, University of Rochester

Stan described the increasing adoption of flash-based storage, and how there's been very little work in I/O scheduling for flash. In particular, synchronous writes have been a bottleneck for I/O, and they continue to be on flash devices. Then he described the characteristics of flash devices that distinguish them from disks, particularly the lack of seek latency, the large erase granularity, the need to erase before write, and variations between vendors. He then gave an example of why "fairness" in flash I/O scheduling matters: during conflicting reads and writes both response time and variance increase greatly. Each vendor's devices react differently. This was also demonstrated for requests issued in parallel—some devices give better results than others.

This led to the development of FIOS, with fairness as a first-class concern. They try to use fairness but also efficiency by exploiting I/O parallelism, and also use I/O anticipation (delaying I/O briefly) to help achieve this. The approach of FIOS is timeslice management: give tasks equal amounts of time to access the device (possibly non-contiguously). Timeslices are grouped into epochs; an epoch ends when all tasks have spent their timeslice or there are no pending I/O requests. Because reads are faster than writes, reads are penalized more by interfering writes. The authors introduce interference management by servicing reads quickly and delaying writes until the reads complete. This minimizes the opportunity for interference.

To exploit I/O parallelism while still respecting their timeslice management, the authors require some cost accounting method for parallel requests. They have two models: a linear cost model, and probabilistic fair sharing. The linear cost model calculates I/O cost as a function of request size, calibrated by times to service reads and writes of different sizes. The probabilistic fair sharing model tries to estimate the amount of concurrency occurring. They used the linear cost model for the results in the rest of the talk.

Anticipatory I/O was used on disks to improve performance hits due to deceptive idleness. It's not necessary for flash performance, but they use it for fairness. Deceptive idleness can cause an epoch to end early, robbing some process of the remainder of its timeslice. Also, a write issued immediately before a group of read requests is bad. The question is how long to wait, without wasting valuable I/O time. Their default is to wait for half of an average request service time.

They evaluated fairness by measuring the I/O slowdown ratio—the amount a request's response time is degraded compared to running alone. For *N* tasks, proportional slowdown means that each task should experience no more than a factor of *N* slowdown. They show that the NOOP, CFQ, and SFQ(D) schedulers slow down reads dramatically, while writes are faster than proportional; FIOS achieves fairness, with both reads and writes faster than proportional slowdown. Quantum-based scheduling is fair, but relatively slower. Stan showed how FIOS also performs well under asymmetric read/write loads. Running SPECweb and TPC-C simultaneously showed that FIOS maintained fairness under real workloads. Finally, performance on a low-power CompactFlash system showed better read efficiency than the other schedulers and fair write performance.

Stan reiterated that fairness was their primary concern. I/O anticipation is important for fairness, even though it's not important for pure throughput. The I/O scheduler must be robust in the face of differing flash architectures. The authors believe that the FIOS approach to fairness might also be applicable to other domains such as virtual machines and the cloud.

Geoff Kuenning (Harvey Mudd) asked what their definition of fairness was on the slide showing asymmetric I/O. Their measure was equal latency—how much each task is slowed down. Geoff thought that this was a matter of opinion, but didn't want to push the matter. Someone from Google asked whether they planned to look at ticket-based schedulers, where tickets are issued proportional to I/O sizes, to give them parallelism without anticipation. Stan explained that these policies aren't specific to FIOS—FIOS was the artifact that came out of looking at these policies—and that they could try a ticket-based approach. Vasily Tarasov (Stony Brook) wanted to know whether they ran experiments with different priorities assigned to tasks. The authors had not, but Stan suggested that they could do scheduling within each priority class, or hand out different-sized timeslices.

Shredder: GPU-Accelerated Incremental Storage and Computation

Pramod Bhatotia and Rodrigo Rodrigues, Max Planck Institute for Software Systems (MPI-SWS); Akshat Verma, IBM Research—India

Pramod Bhatotia started with a fundamental problem: given that the total volume of data is growing rapidly, how can we efficiently store and process it all? One major technique is to eliminate redundancy. Redundancy elimination is expensive, however, and is a three-step process. First, a file is broken into chunks, then the chunks are hashed, and finally the hashes are matched to establish whether or not a duplicate exists. "Content-based chunking," introduced in SOSP '01, uses a sliding window over a file rather than fixed chunks. This can keep boundaries stable under small insertions or deletions in the data. Unfortunately, it is very CPU intensive, which can be a bottleneck.

Content-based chunking throughput on a multicore machine is about 0.5 GBps, but about 1 GBps with a standard GPU-based design. This is a 2x improvement, but still not good enough—the target I/O bandwidth they're trying to support is 2.5 GBps! The reason for the performance gap is because GPUs are designed for compute-intensive workloads, not data-intensive workloads.

Buses run from host memory to CPU, from the CPU to GPU (PCI), and inside the GPU, which is divided into a set of multiprocessors with local memory and a global pool of shared memory. The CPU can only access this global memory, and not the private memory of the compute units. First, data is transferred to device memory, then the CPU launches threads on the GPU, which loads data from global GPU memory into its local memory for fast access, and eventually pushes results back to the host.

There are several scalability problems here. The cost of data transfer across the PCI bus is comparable to the time spent in the chunking kernel. Shredder pipelines data transfer by dividing GPU global memory into two portions, and loads one while chunking is being executed on the other. The second problem is memory access conflicts on the device itself. For speed, data needs to be fetched from global GPU memory into the local processing unit memory. Global memory is divided into interleaved banks, and threads accessing the same bank simultaneously cause a conflict, leading to serialization. Shredder's solution is to coordinate threads so that they cooperate to fetch data for a task from separate banks, leading to parallel bank access. Then the threads can work in isolation on device local memory.

Shredder was implemented using C++ and CUDA, and benchmarked on an NVIDIA Tesla c2050 hosted on a 12-way Xeon. The basic GPU approach achieved 1 GBps. Using pipelined CPU->GPU transfers, this can be improved to 1.75 GBps. Finally, using the coalesced threads loading local processing unit memory, they achieve 2.25 GBps. Pramod then presented a case study on incremental MapReduce, where some input has changed; they want to recompute along the path from this changed input using the other unchanged intermediate results. The problem is that using fixed-size chunking would throw all the chunk boundaries off if data is inserted or deleted, so they use content-based chunking to partition data before running the MapReduce process.

Someone from EMC BRS asked what their baseline multicore performance was, since OpenSSL gets 350 MBps on a single core for SHA1. Did they need all 12 cores to get 500 MBps? It seems slow for a multicore, and it was suggested that they could get the same performance by tuning their CPU implementation. Pramod disagreed, but discussion was deferred. Brent Callaghan (Apple) wanted to know whether they did hashing in the GPU as well as the chunking. Pramod clarified that they are only finding chunk boundaries, although in theory they could do the hashing as well. Someone from UC Santa Cruz wondered what the difference was between this and scientific computation, since there has been work done on using GPUs for scientific workloads. Pramod differentiated the two by suggesting that scientific applications are often N^3, N^4 while chunking is linear, so it's all about transfer bandwidth.

Adding Advanced Storage Controller Functionality via Low-Overhead Virtualization

Muli Ben-Yehuda, Michael Factor, Eran Rom, and Avishay Traeger, IBM Research—Haifa; Eran Borovik and Ben-Ami Yassour

Avishay noted the need for new functionality in storage controllers (e.g., deduplication or compression) that has already been implemented elsewhere. The conventional approach has been to port the functionality from its original environment onto the storage controller itself. This has the advantage of low overhead but incurs a high engineering and maintenance cost. Another approach is to perform the function on a separate machine, which he calls the "gateway" approach. This avoids the cost of porting the software, but incurs a runtime performance cost as well as the additional cost of the gateway hardware.

A third way is a hybrid approach: running a VM on a storage controller. Unfortunately, virtual machines have a bad reputation for overhead. Storage controllers are special-purpose devices with finely tuned resource control. Virtual machines provide a large number of features, not all of which are needed on a storage controller—they need the fault isolation and the separate environment, but they don't need resource-sharing, the ability to overcommit, or migration. So they thought that perhaps they could customize virtual machines to make them suitable for use on storage controllers.

Avishay defined external communication as the I/O between client and VM; internal communication is the communication between the VM and the controller. In their approach, the I/O interfaces are directly assigned to the VM, although servicing interrupts and I/O completions require a hypervisor context switch. To communicate between the VM and the controller they use a virtual I/O block device built on top of shared memory, but it also requires hypervisor switches to handle the interrupts and I/O completions.

Their main approach to avoiding the latency of the hypervisor is polling. They run a polling thread on the guest VM which polls the NIC, avoiding the need for an interrupt. The block request is put into shared memory by the VM; the host detects this request by its own polling thread. When the request on the controller is complete, the host process puts the completion in shared memory, where it is detected by the original polling thread on the VM, finishing the I/O path with no interrupts. They also statically allocate CPU cores and memory, since they can establish resource usage parameters beforehand. The VM is configured to poll when idle instead of sleeping, since they don't need to share with other VMs. They minimize memory management overhead by backing the VM's memory with HugePages.

To benchmark, they configured two servers, each with a pair of quad-core 2.9 GHz Xeons and 16 GB RAM. One server functioned as a load generator, the other served as an emulated storage controller. They compared their VM-based solution to a "bare metal" implementation with four cores assigned to the host. Storage was emulated by an 8 GB RAM disk to avoid the I/O bottleneck of a physical disk.

Their first evaluation is response latency during a ping flood. They show that with no polling, the bare metal solution takes 24 ×s, but the VM uses 89 ×s. On the other hand, with polling enabled both solutions take 21 ×s. On the Netperf benchmarks, guest and host polling show the best performance among the configurations they tried, except on TCP receive throughput, which they claim is because no real work is being done. They calculate that in the optimized case, they add only 6 ×s of latency to a random 4k synchronous write. They improve read throughput by a factor of 7 and write performance by 6 times. Finally, they present the incremental improvements of each of their optimizations, showing that after the work they've done they match the bare metal performance at the end. When they cut the controller down to four cores again, they take a performance hit because of the competing polling threads, but they can tweak thread priorities and affinities to overcome this.

Their conclusion is that virtual infrastructure can be used with near zero performance overhead. This provides the benefits of the high performance and low hardware cost of native integration combined with the shorter time to market and simpler development of the gateway approach.

Lakshmi Bairavasundaram (NetApp) wanted to know, if cores were assigned statically, how would they deal with situations where VMs were being supplied by multiple vendors? Avishay responded that you only need to configure it once, when you figure out what software you are deploying on your controller. Dutch Meyer of UBC asked if the presenter could comment on virtual storage appliances. Avishay said that his understanding was that they run extra functionality outside of the controller.

Mobile and Social

Summarized by Swapnil Patil (svp@cs.cmu.edu)

ZZFS: A Hybrid Device and Cloud File System for Spontaneous Users

Michelle L. Mazurek, Carnegie Mellon University; Eno Thereska, Dinan Gunawardena, Richard Harper, and James Scott, Microsoft Research, Cambridge, UK

In this talk, Michelle Mazurek presented a new file system for mobile/home networked devices through the use of new hardware components and a combination of storage system techniques. The goal of this file system, called ZZFS, is to provide spontaneous data access with good trust and control over data storage. Michelle first presented a user study from traces in the LiveMesh and Dropbox service; this study was driving the design of their ZZFS system. Key observations of this study include: (1) users are busy and want spontaneous response from the system, (2) users do not know their data needs a priori, and (3) users place/organize their data in a planned and reasoned manner.

Because battery life is a key concern on many mobile devices, ZZFS relies on an existing hardware component, the Somniloquy NIC, that provides on-demand network interface card wakeup with some on-board flash. One hardware assumption ZZFS made was that users rely on broadband connections at home with weak 3G-based connections on mobile devices.

ZZFS used a combination of well-studied storage systems techniques, including flat namespace metadata service, policy-driven I/O director service, and I/O offloading for effective power management. The authors built a prototype of their ideas and evaluated the performance of ZZFS's design for spontaneous and ad hoc data access.

Margo Seltzer (Harvard) asked about the few random slow requests in write latency. Michelle said the problem was probably with the WiFi router in the experimental setup. Jason Flinn (Michigan) said that centralized storage has advantages, but he agreed with Michelle' about it being hard to trust and asked for thoughts on how that could be improved. Michelle said that all trust-based issues are more about the experience; several companies have had bad experiences with user data and trust—this affects people's attitudes toward using centralized systems. John Berry (Riverbed) asked about the cache coherency policies in the system: for example, updates to the shared music repository. Michelle answered that if a device is on, the updates are sync'd and serialized. John asked whether there could be races in the middle of a song. Michelle replied that ZZFS relies on the Everest system at MSR (published few years ago) which uses serialization through the primary copy of the data.

Revisiting Storage for Smartphones

Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu, NEC Laboratories America

Awarded Best Paper!

Nitin presented work on the performance of storage systems in smartphones. Mobile devices are becoming increasingly diverse in hardware, software, and application ecosystem. Most existing work has studied network and CPU performance; much work has also been adopted by system developers to make better use of these two resources. The authors studied the performanceof a suite of popular apps on a Google Nexus phone with Android OS. Although both the hardware and the systems software were commodity, the authors patched the OS with some measurement and monitoring extensions. To measure the storage I/O behavior, this work used different storage media devices (i.e., SD cards); this enhanced the study to be more device agnostic.

Several lessons emerged from this work. For SD cards, the results showed high disparity between random and sequential I/O performance; the device specifications are "bloated," because vendors report sequential speeds instead of slow random I/O speeds. Furthermore, the authors also observed that the storage device performance has not improved as much as the network speed performance over the past few years. In terms of applications, it was observed that although there are only half as many random writes as sequential writes, most apps write sufficient random data that the application performance is adversely affected. Application performance is also dependent on the quality of the storage media. In terms of systems software, applications target their writes either to a file system (in-memory, cached file system) or to a SQLite database. Both FS and DB behave differently; in particular, the synchronous writes used by many apps cause the DB writes to be much slower than FS writes.

Geoff Kuenning (Harvey Mudd) said that an Apple I/O study showed that most apps do a lot of synchronous I/O. Since you make similar observations, is it just that plain stupid apps are the problem? Nitin said that the common theme is the presence of App-OS modularity and interface boundaries—which is not good in all cases, particularly when performance is a victim of that modularity. Eno Thereseka (MSR) asked, does it really matter if storage is the bottleneck in end-to-end experience? Nitin replied that as the users of the phones and apps, they found that app performance is highly variable depending on how one uses the apps and phones; since all measurement is at the user level, they captured as much end-to-end performance as possible using black-box phones.

Serving Large-scale Batch Computed Data with Project Voldemort

Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah, LinkedIn Corp.

Roshan Sumbaly presented their large-scale batch processing system built on a key-value store. LinkedIn runs large Hadoop applications that need to bulk-load massive amounts of data in a system that is online and is processing active requests. LinkedIn relies on the Voldemort key-value store that was inspired by Amazon's Dynamo paper. In this work, LinkedIn developers extended Voldemort to overcome performance degradation due to index creation and mutation.

Extensions to Voldemort included incremental bulk loading, data error minimization and mitigation, and ease of use through configuration management. Two existing approaches, including a Hadoop-based insertion tool and multi-cluster deployment, failed due to performance interference and management complexity. Instead, the authors used Hadoop's parallelism and fault tolerance to build an intermediate table construction phase that relied on easy rollback using versioned datasets. They also added simple rebalancing protocols to drive changes in the data-to-memory ratio to mitigate bottlenecks due to memory utilization. Results from their production system show that LinkedIn's extensions help run large data pipelines while maintaining sub-5 ms latency for active users.

Konstantin Shvachko (eBay) asked how rollback worked with new versions. Roshan said that rollback is more than just symlink repointing. It also involves closing an mmaped index, updating the current versions, and then updating symlinks. Rollbacks are worst-case scenarios to handle production problems. Dan Peek (Facebook) asked, why not have one large ring instead of many small rings? Roshan replied, because there is a chance that rebalancing work may increase for a single ring.

Work-in-Progress Reports (WiPs)

Summarized by Michelle L. Mazurek (mmazurek@cmu.edu)

Generating Realistic Datasets for Deduplication Analysis

Vasily Tarasov and Amar Mudrankit, Stony Brook University; Will Buik, Harvey Mudd College; Philip Shilane, EMC Corporation; Geoff Kuenning, Harvey Mudd College

Research and industry have developed many different deduplication protocols. Comparing them is difficult because evaluation depends so heavily on the dataset used. What is needed is a benchmarking dataset that is large, realistic, versatile, easy to distribute, and with parameters that are easy to tune. This work attempts to develop such a dataset by observing and emulating the way that real file systems mutate over time.

Disk-Failure Injection Framework for Fault-Tolerant Systems Research

Yathindra Naik, Mike Hibler, Eric Eide, and Robert Ricci, University of Utah

It is important to understand how modern complex storage stacks will behave in the face of disk failures. This work builds a framework for injecting disk errors for testing. The framework should be realistic, controllable, repeatable, scalable, and scriptable. Using Emulab, the presenters model delayed I/O, corrupt reads and writes, and sector errors. In progress: more realistic failure models that reflect the realistic frequency and distribution of these errors. A prototype will be available soon.

DS-RAID: Efficient Parity Update Scheme for SSDs

Jaeho Kim and Jongmin Lee, University of Seoul; Jongmoo Choi, Dankook University; Donghee Lee, University of Seoul; Sam H. Noh, Hongik University

Current SSDs provide low reliability, a high error rate, and a limited erase count, with multi-level cells exacerbating the problem. Current approaches to applying RAID5 (striping) to SSDs have limitations related to small writes and the inability to write new data until a stripe becomes full. Parity pages must be written too frequently, increasing wear. This work uses dynamic striping to solve these problems. Stripes are constructed based on arrival order, containing non-consecutive block numbers. Sub-stripe parity is used for write requests smaller than the stripe size. Evaluation shows DS-RAID results in fewer extra reads and writes, with less cleaning cost, than standard RAID5.

The Peril and Promise of Shingled Disk Arrays (How to Avoid Two Disks Being Worse Than One)

Quoc M. Le, JoAnne Holliday, and Ahmed Amer, Santa Clara University

Shingled disks promise to increase storage density for disk drives, but must be used carefully because updates to written tracks may overwrite neighboring tracks. This work evaluates the behavior of shingled disks when used in array configuration or when faced with heavily interleaved workloads from multiple sources. There are three evaluation workloads: pure (sequential workloads), striped (interleaved workloads), and dedicated (one workload per disk). As might be expected, more interleaving results in more disk activity as bands are relocated. Proper use of shingled disks may therefore require rethinking traditional disk array layouts.

A Unified Object Oriented Storage Architecture

Andy Hospodor, Ethan Miller, Rekha Pitchumani, Yangwook Kang, and Darrell Long, University of California, Santa Cruz; Ahmed Amer, Santa Clara University; Yulai Xie, Huazhong University of Science and Technology

This work presents a unified storage architecture based on object-oriented storage. The goal is to decouple metadata from data, allowing management of objects rather than blocks. This architecture can apply to a range of devices including magnetic, optical, SSD, tape, and even shingled disks. The presenters suggest that such an architecture should be designed from scratch, rather than attempting to extend SCSI. The architecture will provide typical methods such as read and write, as well as new methods including find, append, replicate, merge, and sort. An OO storage device should use a publish-subscribe model to allow operating systems to access these methods.

Challenges in Long-Term Logging and Tracing

Ian F. Adams and Ethan L. Miller, University of California, Santa Cruz

Long-term logs and traces are important, as some trends in system use aren't apparent in the short term. We have good tools for capturing log and trace data, but not for maintaining it over the long term: it's too much data, there are occasional hiccups in collection, and log formats change. The presenters propose periodically transforming older data to coarser resolution, so it takes less space and is easier to work with, while keeping fine-grained logs for particularly interesting events. They also suggest annotating logs and traces to indicate events like maintenance, nodes or processes going down, etc. Another idea is to combine traces with snapshots of system state, so you can use the trace as a record of the change between snapshots. Finally, it's important to periodically check for format consistency and note anomalies and problems, so that log parsers don't break or, even worse, fail silently when processing a lot of log data.

Dynamic Block-level Cache Management for Cloud Computing Systems

Dulcardo Arteaga, Douglas Otstott, and Ming Zhao, Florida International University

Block-level network storage is used in cloud systems to provide VM storage and allow fast VM migration as well as VM availability. However, as cloud systems increase in size, there are scalability problems. This work uses dynamic, block-level client-side caching to improve performance at scale. This approach exploits data locality in VM data access, while supporting dynamic and flexible cache configuration. Each host contains one cache, which all the VMs on that host share. Within this cache are virtual caches for each VM so they don't interfere with each other. When the VM migrates, the virtual cache is flushed. Preliminary results show improved throughput using the IOzone benchmark.

CASE: Exploiting Content Redundancy for Improving Space Efficiency and Benchmarking Accuracy in Storage Emulation

Lei Tian and Hong Jiang, University of Nebraska—Lincoln

Storage benchmarking is very sensitive to content, with the same operations on different content potentially inducing very different performance results. As a result, benchmarks must retain data content. CASE aims to provide flexible, space-efficient, timing-accurate, and content-aware storage emulation for benchmarking. CASE is implemented using data deduplication over fixed-size chunks. Preliminary results indicate that CASE saves up to two orders of magnitude in storage space.

Trusted Storage

Anjo Vahldiek and Eslam Elnikety, MPI-SWS; Ansley Post, Google; Peter Druschel and Deepak Garg, MPI-SWS; Johannes Gehrke, Cornell; Rodrigo Rodrigues, MPI-SWS

Storage is complex, involving millions of lines of code, operating systems, file systems, drivers, etc. This complexity means vulnerability to bugs, viruses, and operator errors that threaten integrity, confidentiality, and durability. The presenters developed a trusted storage architecture that enforces user-provided policies for application objects like files. Policies may be based on user ID, hardware or software configuration, quota, time, or location, and govern the conditions under which objects can be read, updated, or deleted. Signed certificates attest to the object's properties, policies, and access history. Overhead for the implementation is expected to be below 3%.

Accelerating Data Deduplication by Exploiting Pipelining and Parallelism with Multicore or Manycore Processors

Wen Xia, Huazhong University of Science and Technology and University of Nebraska—Lincoln; Hong Jiang, University of Nebraska—Lincoln; Dan Feng, Huazhong University of Science and Technology; Lei Tian, University of Nebraska—Lincoln

Deduplication is important for storage efficiency, but the process of chunking and fingerprinting data is time-consuming and CPU-intensive. The presenters propose P-Dedupe, which exploits parallelism and pipelines to avoid this computation bottleneck. P-Dedupe divides the data stream into multiple sections that can be chunked and processed in parallel, with the boundaries between sections requiring special processing to account for the sliding windows used in chunking.

High-Throughput Direct Data Transfer Between PCIe SSDs

Jun Suzuki, Masato Yasuda, Masahiko Takahashi, Yoichi Hidaka, Junichi Higuchi, Yoshikazu Watanabe, and Takashi Yoshikawa, NEC Corporation

Data reallocation and backup are examples of data being transferred between devices without modification. Currently, this transfer must traverse main memory of the server hosting the I/O devices; this link can become the bottleneck. The presenters propose DirectConnect, a method to transfer this data directly, using memory in a PCIe-to-Ethernet bridge as an intermediate buffer for DMAs of the source and destination devices. Prototype evaluation shows high throughput even when server bandwidth is narrow.

Grouping Data for Faster Rebuilds: The Art of Failing Silently

Avani Wildani and Ethan Miller, University of California, Santa Cruz

In big systems with erasure coding for reliability, rebuild time after failure is inevitable and slow—up to six hours to rebuild a 300 GB disk. The goal of this work is to reduce the impact of rebuild by striping intelligently. Data is grouped into access groups that correspond to real-life working sets for applications, users, or projects. Striping these access groups strategically can ensure that a rebuild halts progress for only a few users or projects rather than all of them—one project must rebuild all of its data, rather than rebuilding some data for each of many projects. Ongoing work includes evaluation with probabilistic fault injection, modeling correlated failures, and measuring overall impact of rebuilds.

Toward an Economic Model of Long-Term Storage

Daniel C. Rosenthal, University of California, Santa Cruz; David S.H. Rosenthal, Stanford University Libraries; Ethan L. Miller and Ian F. Adams, University of California, Santa Cruz; Mark W. Storer, NetApp; Erez Zadok, Stony Brook University

People want to store their content indefinitely, but want to pay for that storage up-front rather than on a continuing basis. The cost of indefinite storage is difficult to predict, depending on future events ranging from regular disk replacement to natural disasters. The presenters use Monte Carlo modeling to simulate hypothetical futures for a storage system. They calculate tradeoffs between cost and the likelihood of data survival, in an attempt to value the endowment needed to preserve data.

Emulating a Shingled Write Disk

Rekha Pitchumani, University of California, Santa Cruz; Yulai Xie, Huazhong University of Science and Technology; Andy Hospodor, University of California, Santa Cruz; Ahmed Amer, Santa Clara University; Ethan L. Miller, University of California, Santa Cruz

Shingled disks can more than double disk capacity, but, due to their architecture, random writes may destroy data. In particular, in-place overwrites can be destructive. Research and development of how to best manage shingled disks is hindered because they are not yet available for testing. The presenters' goal is to emulate shingled disks by providing a device driver that mimics their operations. The driver uses a mapper that maintains knowledge of which tracks are overwritten by which writes, so reads to overwritten tracks return the overwritten rather than the original content. Future work includes adding the ability to report physical geometry. The emulator can be useful even after shingled disks become available as a platform for consistent and controllable testing. The emulator is currently being evaluated and will be released soon.

Cloud

Summarized by Daniel Fryer (dfryer@cs.toronto.edu)

BlueSky: A Cloud-Backed File System for the Enterprise

Michael Vrable, Stefan Savage, and Geoffrey M. Voelker, University of California, San Diego

Michael introduced BlueSky by stating that since many services are moving towards the cloud, they wanted to explore the idea of a network file server backed by cloud storage. Existing cloud storage acts like another level in the storage hierarchy, but with different characteristics. The interface usually only supports writing complete objects, but if one supports random reads, latencies are high enough that the additional penalty of random access doesn't matter. Privacy becomes more of a concern, and since billing for storage is usually by quantity used, deleting unused data is important.

Their approach to providing a cloud-backed NFS service avoids modifying the client NFS/CIFS stack. Instead, they implemented a caching proxy server. The proxy can provide lower latency, perform write-back caching, and encrypt before forwarding requests to the cloud storage service. On the back-end, they use a log-structured file system. When writing, each segment is uploaded by the proxy all at once and is stored as an object in the cloud. For reads, they take advantage of the ability to do random access on the content of these segments. There is also a garbage-collecting log cleaner process, which can run on the proxy or on a compute node in the cloud.

To maintain confidentiality, the log-cleaning process does not need to have the encryption key for the file system, which can remain safely on the proxy. They do need to make some metadata available, so they structure their metadata as a four-level tree. The top two levels are the log checkpoints and the inode map, which locate the most recent versions of inodes in the log. These levels are unencrypted. Below these are the inodes and data blocks, which do have encrypted contents. Michael then presented a diagram of the proxy architecture. At the front-end are NFS or CIFS interfaces to handle client requests. Since they do writeback caching, they write to the local disk before replying to the client to announce that a write is durable. Once they have accumulated a log segment's worth of data, they can encrypt it and use cloud-specific back-ends (S3, WAS) to store the log segment in the cloud.

Their design is predicated on high-bandwidth connections to the cloud service provider. One of the major problems is latency, which is partly a function of location. Measuring performance with varying object sizes and amounts of concurrency, they showed that 32 concurrent connections could saturate a 1 Gbps link.

To benchmark their system, they ran a kernel source unpack, checksum, and compile process. Michael compared a local NFS server, a purely remote NFS server, BlueSky with a warm cache, and BlueSky with a cold cache. They also evaluated cache hit ratios and the effect they had on client performance. With about a 50% hit rate, they were able to keep read latencies within 2x or 3x of the purely local solution. They could write at local speed until the proxy ran out of disk space for logging, at which point they were limited by bandwidth to the cloud. Michael then presented a final benchmark, based on SPECsfs2008. BlueSky performed similarly to the local NFS system with unconstrained network bandwidth; with a constrained network it scaled to about 90% of the local throughput before dropping off and becoming erratic. They also found that while fetching full segments was helpful for the compile benchmark, it had a negative impact on SPECsfs.

Based on S3's pricing model for bandwidth and operation counts, they calculated the costs of BlueSky. The main point was that by aggregating writes into log updates and by allowing random reads, they decreased usage costs dramatically.

Brent Callaghan (Apple) asked whether they had thought about multiple proxies accessing the same data store; he also wanted to know if they had thought about backup. Michael explained that they'd thought about some of the issues but hadn't implemented any of their ideas. There are several reasons why you might want to have multiple proxies: for higher scalability or for geographically distributed access. One approach would be to have multiple proxies writing to separate logs in the cloud and rely on some kind of opportunistic concurrency, or maybe implement distributed logging. For backups, as long as you don't garbage-collect all your log segments, you can get information from a previous checkpoint.

Someone from Nimble Storage wondered whether log structuring was worth it, given the complications of cleaning. Why couldn't you just increase throughput with higher concurrency? Michael explained that the major reason is cost; you pay an operation cost on a per-object basis. The cleaning can run in the cloud, so you don't pay transfer charges.

Someone from Red Hat asked what their benchmark NFS server was, because the numbers looked horrible. They were referred to the numbers in the paper, but it was a Linux server with a couple of disks. The numbers might have been different had they used a high-performance storage server. Joe Tucek of HP Labs asked whether they had thought about the different consistency models provided by the different cloud services. Michael replied that, because BlueSky is log-structured, they're not overwriting data in place, so eventual consistency doesn't cause them as much trouble. They don't have different versions of objects; they're either there or not there. If something just isn't there, they could retry, timeout, or report an error to the client.

Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads

Osama Khan and Randal Burns, Johns Hopkins University; James Plank and William Pierce, University of Tennessee; Cheng Huang, Microsoft Research

Osama Khan discussed the rapid growth in the total quantity of stored data, projecting a 44-fold growth over 10 years, particularly in the cloud. With this much data, replication is not a cost-effective means to achieve reliability. Erasure coding is a natural solution to this problem, but with traditional erasure-coding approaches, recovery is a slow process requiring the involvement of many devices. Existing erasure codes are not designed with minimal recovery I/O in mind.

Their solution is to create an algorithm that minimizes the amount of data needed for recovery under any XOR-based erasure code. Before describing the details, Osama presented an overview of the general process of erasure coding, starting with a block of file system data. The encoding is done by a matrix multiplication, and then the result is distributed into stripes after encoding. He gave an example of the type of decoding equation that results from this process. Their algorithm finds a decoding equation for each failed bit while minimizing the total number of symbols needed for reconstruction, given the code generator matrix and a list of failed symbols. They do this by constructing a directed graph, with the weights on the edges representing the number of symbols involved in the equation. The lowest-cost path through the graph minimizes the number of symbols involved. These solutions can be precomputed and stored for later use.

Their second contribution was to address the problem of degraded reads—disks that are temporarily unable to deliver data. To optimize read performance to deal with this, they invented a new class of codes called "rotated Reed-Solomon codes." Standard coding schemes compute different symbols from single rows, whereas rotated codes span multiple rows. This means that each coding disk is using slightly different symbol sets. Osama presented some examples of what kind of access has to be done during failure and how the rotated Reed-Solomon codes require fewer reads.

Jay Wiley asked how their graph-based algorithm compared to Hafner's work using matrix methods. Osama couldn't remember, so they took it offline. If their symbol size was 100—500 MB for performance reasons, Jim Molina of Western Digital wondered, what kind of correction capacity would they have? It was clarified that the block size doesn't affect correction capacity, which is a function of redundancy.

NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds

Yuchong Hu, Henry C.H. Chen, and Patrick P.C. Lee, The Chinese University of Hong Kong; Yang Tang, Columbia University

Patrick noted how outages or vendor lock-in makes depending on a single cloud provider for storage risky. The obvious solution is to take advantage of multiple-cloud storage, using a proxy to stripe data across the clouds using an MDS encoding scheme where any K out of N clouds can reconstruct the original data. Repair would then involve downloading all the data from the functioning clouds to determine what to write to a new cloud. This could incur a high repair cost due to bandwidth usage equivalent to the size of the whole dataset.

Their system, NCCloud, applies the idea of "regenerating codes" to the problem of repair in bandwidth-constrained situations. Regenerating codes aim to reduce the amount of data needed to perform reconstruction of a failed node by selectively downloading portions of the data stored on each node, where the nodes themselves may perform some computation on the data during the reconstruction process. Up to this point, regenerating codes have primarily been studied from a theoretical perspective. To keep NCCloud simple, they would like to avoid any computation on the storage nodes.

NCCloud relies on their implementation of a functional minimum-storage regenerating code (F-MSR). Reconstruction is based on random linear combinations of existing chunks. Unlike a "systematic" code, they don't keep the original data around, but only the linearly combined code chunks. This makes actual decoding expensive. They propose F-MSR for rarely read long-term archival applications. One challenge that arises is ensuring that after reconstruction, the MDS properties of the original encoding are preserved and that any subsequent repair will preserve properties as well. F-MSR reduces repair bandwidth cost by 25%. They compare NCCloud with F-MSR to Reed-Solomon–based RAID-6. F-MSR has higher response time during writes, due to encoding overhead, which they expect will be masked by network latency unless *N* is very large. Reconstruction time is lower, due to less bandwidth use. In summary, NCCloud realizes an implementation of a regenerating code, which preserves storage cost but uses less repair traffic.

Someone asked what the odds were of losing data from a single cloud provider, much less two. Patrick argued that there are many new cloud providers, and we can't guarantee that they are all equally reliable or available. The questioner said that he thought that the math was really interesting, but he didn't think the economics of it made sense.

Poster Session II

See static.usenix.org/events/fast12/poster.html for PDFs and descriptions of all posters.

A Little Bit of Everything

Summarized by Doowon Kim (dwkim@cs.utah.edu)

Extracting Flexible, Replayable Models from Large Block Traces

V. Tarasov and S. Kumar, Stony Brook University; J. Ma, Harvey Mudd College; D. Hildebrand and A. Povzner, IBM Almaden Research; G. Kuenning, Harvey Mudd College; E. Zadok, Stony Brook University

Vasily Tarasov described what their traces look like. In general, a timestamp is a common field, but other fields depend on what events are traced. The authors used block traces focusing on operation, I/O size, and offset, but their approach is valid for any trace. Vasily said that there are two main use cases for traces: (1) workload analysis and characterization and (2) trace replay. He mentioned that trace replay has some problems.

Vasily showed why statistics matter. Although traces collected on the same machine and in the same environment might differ on Monday and Tuesday, for example, it's the overall statistical modeling of properties such as I/O rate and read-write ratio that are important to consider in evaluating systems.

He explained their design goals: (1) accuracy, (2) conciseness, (3) flexibility, and (4) extensibility. Vasily explained that the first problem they encountered was that workload can change in the trace over time. He explained the feature functions within a chunk and said that they can put the value into a multi-dimensional histogram. He then said they could generate benchmark plugins and explained what the plugins are. In the evaluation of their work, he argued that the average relative error is less than 10% across all parameters and systems, and there was a 17x–25x size reduction. Vasily then discussed their future work: (1) more accurate parameters, systems, and traces; (2) file system traces; (3) automatic selection of parameters; and (4) operations on models.

Joe Tucek from HP Labs asked about the difference between Monday and Tuesday traces. Vasily replied that many assume the traces will be the same day-to-day, but there may be significant differences in the pattern of use, while the overall load remains the same. They want to develop meta-cases and be able to work from that. Someone from VMware asked about the chunk sizes used in deduplication. Vasily said that he understood the issue, but that they didn't include the information about chunk sizes in the paper. Someone from Microsoft asked if it takes an expert to choose from their library of functions that can create particular traces. Vasily said they had published a tool named Distiller that helps with this with good results. Josh Berry of Riverbed Technology asked if they had looked at latency-dependent workloads. Vasily said that this is a known problem with traces, and they did experiment with adding in random delays or having no delays (infinite speed).

scc: Cluster Storage Provisioning Informed by Application Characteristics and SLAs

Harsha V. Madhyastha, University of California, Riverside; John C. McCullough, George Porter, Rishi Kapoor, Stefan Savage, Alex C. Snoeren, and Amin Vahdat, University of California, San Diego

John McCullough started by explaining provisioning hardware for cluster applications. There are many goals for provisioning, but he focused only on achieving SLA (performance) goals and minimizing cost for a single application while emphasizing storage. He said that the challenge is a very large configuration space, making solving this problem non-trivial. The current state-of-the-art solution is just to apply rules-of-thumb from experience, and use trial-and-error with various configurations. Their goal is to discover what a low-cost configuration is now and what a low-cost configuration will look like in the future. They do this by first measuring "in-the-small" and modeling application performance, in order to predict "in-the-large." He explained scc (Storage Configuration Compiler). If cluster building blocks, an application model, and SLA specification are put into scc, it produces a spectrum of cost for different configurations.

John explained cluster building blocks. Servers have many components, such as cores, RAM, storage, I/O, and network. You also need an application model to use scc. The model has tasks (computation), datasets (storage), edges between tasks and datasets (I/O), and edges among tasks (dependencies). In his example of the model, photo-sharing, there are three datasets: photos, thumbnails, and tags. The related tasks are single operations with known sizes for writing or reading from the datasets. John said that if you use only a hard disk and single core, the cost will be really low. However, if you use a lot of SSDs, the cost will be high. The guiding principle is to meet SLA. He said ILP minimizes cost.

John talked about validation. They built three applications: photo-sharing, product search, and Terasort. He showed that scc meets the SLA at lower cost for all three.

Sanghyun Cho from University of Pittsburgh asked whether the author had considered including costs such as power. John answered that currently they did not include power bills. Fred Douglis of EMC asked whether they can handle very large models and had tried perturbing the inputs. John answered that for larger models they use a gradient descent to pick the best solution. In terms of perturbing, some of that can be done by swapping out parts. Ben Reed of Yahoo! said that this work reminded him of the Starfish project in the database community, as they had both a processing and a working-set model built by profiling. John said he wasn't familiar with Starfish, but they did get feedback from unnamed storage providers. Ben said that just tuning the software configuration made huge differences. John said that this sounds interesting to pursue, but complex. Someone from Google wondered about using scc for a cluster that would have multiple, simultaneous uses. John said they planned to look into that in the future. Randal Burns (Johns Hopkins) pointed out that this is not the way people deploy in the cloud. John said they want to extend their model to support cloud deployments. Rik Farrow asked if scc is available for use, and John replied that he would have to ask his co-authors.

iDedup: Latency-aware, Inline Data Deduplication for Primary Storage

Kiran Srinivasan, Tim Bisson, Garth Goodson, and Kaladhar Voruganti, NetApp, Inc.

Kiran Srinivasan began by explaining the overview and context of iDedup. He said that storage clients are connected to primary storage via NFS, CIFS, or iSCSI, and the primary storage is connected to secondary storage through NDMP or other methods. In this hierarchy, dedupe can save more than 90% in secondary storage. However, primary storage has some unique characteristics. First, performance and reliability are key features. Second, RPC-based protocols are very latency sensitive. Third, only offline dedupe techniques have been developed. He said iDedup is for inline or foreground dedupe for primary storage and has little impact on latency-sensitive workloads. Kiran compared offline dedupe to inline dedupe. He explained why inline dedupe for primary storage is required. He said that this is because provisioning and planning is easier, with no post-processing activities, and allows efficient use of resources. He then explained the key features of iDedup. First, it minimizes inline dedupe performance overheads. Second, it has a tunable tradeoff. Last, it can be combined with offline techniques.

Kiran talked about inline dedupe challenges. First, it has read path challenges. This is because dedupe causes disk-level fragmentation. Second, it has write path challenges, because it produces CPU overheads in the critical write path and extra random I/Os in the write path due to the dedupe algorithm. He then talked about their approach: iDedup provides a solution to read path issues which are dedupe-only sequences of disk blocks, as well as keeping a smaller dedupe metadata as an in-memory cache to write path issues.

Kiran explained that the iDedup architecture has two design-tunable parameters: threshold and dedupe metadata (fingerprint DB) cache size. Kiran then explained the iDedup algorithm and its four phases.

In their evaluation setup, they replayed real-world CIFS traces. Kiran compared iDedup to a system with no iDedup and with full dedupe. They tested three dedupe metadata cache sizes: 0.25, 0.5 and 1 GB. Results showed less than a linear decrease in dedupe saving, and that the ideal threshold is the biggest threshold with the least decrease in dedupe saving. Fragmentation for other thresholds is between the baseline and threshold 1. CPU utilization demonstrated a larger variance compared to the baseline, but the mean difference was less than 4%. Finally, Kiran showed that the result of latency impact for longer response times is larger than 2 ms.

Margo Seltzer asked about the result graph (in Figure 7) comparing the deduplication ratio to the minimum sequence threshold. Kiran answered that they chose to use 4 as the best threshold. Margo then asked about prior work done on inline deduplication at UCB. Kiran answered that this work was similar, but completely redone. Joseph Glider of IBM Almaden Research asked whether their traces include content information. Kiran said that they used content hashes. Glider then asked whether they consider the age of an entry before ejecting it from the cache. Kiran answered that they do not use LRU. The policy they use is based on hashes for blocks that have been previously used for deduplication. Vasily Tarasov (Stony Brook) asked if they preserve the dedup information when moving from primary storage to secondary. Kiran answered that they didn't but it was a good idea.

Flash and SSDs, Part II

Summarized by Doowon Kim (dwkim@cs.utah.edu)

Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems

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

Yongseok Oh explained that hybrid storage systems benefit from combining SSDs and HDDs. One of the important characteristics of flash-based SSD is that it maintains over-provisioned space (OPS). Typical SSDs have a fixed OPS size in which optimal size is unknown, so one of their goals was to determine the optimal size of OPS. According to Oh, as OPS increases, the performance cost of garbage collection (GC) decreases but the cache miss rate increases. Overall, the performance is going to be bad, but optimal OPS size can produce the best performance possible.

Oh presented various cost models and then moved into an explanation of his evaluation setup. He used a hybrid storage simulator and flash cache layers (FCLs). OP_FCL shows near-optimal performance, and optimal performance depends on workload characteristics. OP-FCL dynamically adjusts cache spaces according to workloads. Considerable OPS is used to lower garbage collection cost. Most caching space is used to maintain read data. Optimizing the lifetime of the flash is left as future work.

Umesh Maheshwari of Nimble Storage said the paper was very interesting and he thought it is very useful in real SSD developing. Then he asked whether read cost's dependence on garbage collection cost was their assumption or their experience. Oh answered (with his advisor's help) that in reality, the read should not be affected by garbage collection, but that their assumption was valid because they experienced that. Someone from Google wondered how a cache policy like LRU or FIFO affects the cost model. Oh answered that they did not look at other replacement policies. In this case they used LRU, but, overall, the cost model will not be changed.

Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling

Sungjin Lee and Taejin Kim, Seoul National University; Kyungho Kim, Samsung Electronics, Korea; Jihong Kim, Seoul National University

Lee said that flash-based SSDs are becoming an attractive storage solution for enterprise systems, but poor write endurance results in the limited lifetime of SSDs hampers wideradoption of SDDs. The SSD lifetime is determined by the number of bytes that can be written to the SSD and the number of bytes written per day. Mobile phones and desktop PCs that are not write-intensive can achieve the required lifetime, but with write-intensive workloads such as on enterprise servers, a reasonable lifetime cannot be guaranteed. Flash has a self-healing capability that increases flash lifetime related to the logarithm of the time between erasures. Current schemes such as reducing WAF and incoming write traffic can improve overall SSD lifetime but cannot guarantee the required SSD lifetime. Static throttling limits the maximum throughput of SSDs but is also likely to throttle performance uselessly and to underutilize the available endurance.

Lee introduced Recovery-Aware Dynamic Throttling (READY). READY's design goals are to guarantee the required SSD lifetime, minimize average response times, and minimize response time variations. READY consists of three modules: the write demand predictor, the throttling delay estimator, and the epoch-capacity regulator. The write demand predictor can predict future write traffic for throttling by exploiting cyclic behaviors of enterprise applications. Throttling delay should match future write demand, increasing if demand exceeds epoch capacity, decreasing if demand falls short of capacity, and remaining unchanged if demand equals capacity. The epoch-capacity regulator can throttle write performance by applying the same throttling delay to every page write and increasing a throttling delay later to reclaim the overused capacity.

The team evaluated four SSD configurations: NT, ST, DT, and READY. Results showed that NT cannot guarantee the required SSD lifetime, READY achieves a lifetime close to five years, and ST and DT exhibit a lifetime much longer than five years. NT exhibited the best performance, and READY performed better than ST and DT while guaranteeing the required lifetime. Finally, READY showed shorter response time variations than ST/DT, and ST exhibited the most significant response time variations. Future work involves implementing READY in a real SSD platform and supporting latency-aware performance throttling.