Conference Reports

In this issue:

FAST '11: 9th USENIX Conference on File and Storage Technologies

San Jose, CA
February 15–17, 2011

Opening Remarks and Best Paper Awards

Summarized by Rik Farrow (rik@usenix.org)

The FAST '11 chairs, Greg Ganger (CMU) and John Wilkes (Google), opened the workshop by explaining the two-day format. Past workshops extended two and one-half days, but by leaving out keynotes and talks, they fit the same number of refereed papers, 20, into just two days. There was also a WiPs session and two poster sessions with accompanying dinner receptions.

The Best Paper awards went to Dutch Meyer and Bill Bolosky for *A Study of Practical Deduplication* and to Nitin Agrawal et al. for *Emulating Goliath Storage Systems with David*.

The total attendance at FAST '11 fell just three short of the record, set in 2008, with around 450 attendees.

Deduplication

Summarized by Puranjoy Bhattacharjee (puran@vt.edu)

A Study of Practical Deduplication

Dutch T. Meyer, Microsoft Research and the University of British Columbia; William J. Bolosky, Microsoft Research

Awarded Best Paper!

Dutch Meyer gave a lively presentation of this paper, winner of a Best Paper award, about deduplication research in the form of a study of the files involved. Meyer started by saying that dedup is a good idea, but he is not sold on it because of fragmentation and loss of linearity against the potential storage savings. Storage space being cheap, this is counter-intuitive. He said that dedup is important in cases where the tradeoff achieved is high. This is what this paper focused on. At Microsoft Research, the authors were able to recruit 875

people to investigate their file system states once a week for four weeks, yielding around 40 TB of data.

Analysis of the files revealed that the median file size is 4K and that the files are linear. However, a lot of the bytes are concentrated in a few files, mostly of type .iso (disk images) and .vhd (virtual hard disks). He also talked about the different kinds of dedup techniques out there, such as whole file, fixed chunk, and Rabin fingerprinting. Applying these techniques, they were able to achieve 70% storage saving with Rabin, 60% with fixed chunk, and 50% with whole file. With support for sparse files, another 8% savings were achieved. Further analysis showed that there are two classes of files—small at 10 MB and emerging class of big 8–10 GB files of type .iso, .vhd, .null, etc. They observed that Rabin works well on the big files, such as .vhd. Hence the authors suggested a mixed technique of doing whole file dedup on most files and Rabin on some special files, to reduce the gap of 20% between whole file and Rabin performance. Meyer also announced that they are working to release the dataset.

During the questions, Michael Condict (NetApp) pointed out that people usually care more about the amount of space remaining than about space saved. Assar Westerlund (Permabit) wondered about lower performance than expected for 4K Rabin, and Dutch answered that this might be workload specific, with more bytes in .vhd than earlier, making whole file dedup gains harder. He speculated that techniques to open up the .vhd and examine subfile structures could lead to more gains. Ben Reed (Yahoo) asked about the methodology of the one machine numbers. Bill Bolosky, the co-author, replied that it was for each of the 875 machines. Reed asked whether the dedup numbers were evenly distributed or heavily biased, but the authors had not looked into that. Chris Small (NetApp) had a suggestion for the authors to look into the tradeoff between smaller chunk size for dedups vs. local compression and backup.

Tradeoffs in Scalable Data Routing for Deduplication Clusters

Wei Dong, Princeton University; Fred Douglis, EMC; Kai Li, Princeton University and EMC; Hugo Patterson, Sazzala Reddy, and Philip Shilane, EMC

Fred Douglis presented this paper on the performance in dedup clusters with existing high throughput single-node systems. The requirement for clusters was motivated by situations where backups do not fit in a single appliance, and hence dedup clusters are needed that provide throughput comparable to single node along with load balancing on the cluster nodes. Douglis described how fingerprint lookup tends to be the bottleneck and how cache locality and disk avoidance via containers and Bloom filters can help. He described their approach of using super chunks (a consecutive group of chunks) and deduping node by node at chunk level. This means that the same chunk might appear on multiple nodes, leading to opportunities for balanced storage and scalable throughput, because routing super chunks provides better throughput than routing chunks. He discussed their techniques of stateless (with low overhead) and stateful (with more overhead but better load balancing) routing protocols. In stateless, chunks are migrated when a node exceeds 105% of average disk usage. In stateful, chunks are routed to nodes where it matches the best, but it avoids flooding by adding more new chunks. If no clear winner is found in terms of match, the algorithm defaults to a stateless approach.

He then talked about how contents of each node are tracked using a Bloom filter and their strategy of reducing overhead by sampling with a rate of 1/8. He then described their evaluation to find the best super chunk feature and granularity. They used trace data from a production environment with three large backup sets and five smaller sets with individual data types and one synthetic dataset. They found that a hash of 64 bits works well for small clusters and that bin migration indeed helps to reduce skew and increase effective dedup. Stateful routing was found to offer further improvement. Larger superchunks gave better throughput. He also cautioned that dedup improves after migration but may drop between migration intervals. He talked about the Global Deduplication Array product, which was based on the research presented here. Finally, he talked about future directions: investigating conditions causing data skew, scalability over a broad range of cluster sizes, and supporting cluster reconfiguration with bin migration.

During questions, Michael Condict (NetApp) asked if bin content was accounted for to maximize performance. Douglis replied that they did it in the simulator, but in practice they did not notice sufficient improvement. Geoff Kuenning (Harvey Mudd) said that the probability of troublesome chunk boundaries with Exchange was low, and asked if the authors had fiddled with the hashing algorithm and chunk boundaries to get around that problem. Douglis replied that if the masks were changed, Exchange could work better but other cases might be hit. If they were to start from scratch, a whole chunk hash instead of a first few bits might have worked but at large cluster sizes, a hash is only one of the factors that have to be accounted for. Keith Smith (NetApp) asked about hardware cost tradeoff and said that perhaps bigger nodes could perform better than a cluster. Douglis said that was a possibility. Finally, Christian (NEC) wondered if the tradeoff of 8% decrease in dedup was worth the 50% increase in throughput for routing by chunk and super

chunk. Douglis replied that they were more concerned about throughput, so it was acceptable in their case.

Specializing Storage

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

Capo: Recapitulating Storage for Virtual Desktops

Mohammad Shamma, Dutch T. Meyer, Jake Wires, Maria Ivanova, Norman C. Hutchinson, and Andrew Warfield, University of British Columbia

Jake Wires introduced himself as a member of the storage team of XenSource (now acquired by Citrix) for about five years and currently in Andrew's group at UBC, focusing on how virtualization will affect storage. Then he introduced some interesting predictions from Gartner, one being that by 2013, 40% of desktops will be virtualized. He claimed that he was not trying to argue the accuracy of these predictions but did believe that virtualization is trying to take over the world. To demonstrate this trend, he cited some companies investigating and deploying virtual desktop infrastructure (VDI). For example, UBC tried to deploy 2000 virtual desktops this year. Administrators love VDI because it makes it easy for them to manage desktops centrally while reducing hardware and maintenance costs. And users will embrace it if VDI provides a familiar working environment and comparable performance. Jake then turned to the topic of how VDI changes storage.

He introduced the typical architecture of storage in VDI. A central storage system is used to store disk images for virtual machines, and gold images are used to create a virtual disk with copy-on-write. Such architectures enable sharing of gold images and fast cloning. To facilitate system upgrades, user directories are isolated from system directories in practice. In order to identify what can be done to improve the storage system in the VDI, they did a workload analysis in UBC. They profiled 55 Windows machines for one week by installing drivers which can capture file- and block-level operations. In total, they got 75 GB of compressed logs. After that Jake showed the day-to-day activity and analyzed some peaks. An important observation is that the fraction of accesses to user data is rather small. Most of the accesses are to system directories or metadata updates. He also showed that 50% of the data was rewritten within 24 hours and that the average divergence was 1 GB through a week.

His discussion turned to improving VDI scalability. The main finding from their analysis is that the life of written data is short, and virtual machine hosts usually have local disks which are not used. By using local disks as a persistent cache, VDI can coalesce a significant portion of writes to the central storage system. They also observed that reads to system files can be shared with multiple virtual machines within a host. Implementing local persistent caching which supports both write-through and write-back policies did a good job of eliminating duplicate reads. He also introduced the idea of a multi-host preloader which, because duplicate data is shared among multiple hosts, enables shared data to be multi-casted to all interested hosts. And, finally, the differential durability mechanism enables users to specify different synchronization policies for different directories. For example, user data uses write-through to provide a strong consistency while the page file is totally eliminated from being written back to the central storage system. Jake then provided a micro-benchmark of the preloader, showing that it does eliminate duplicate reads effectively. He also showed how they evaluated their system with trace replay. Write-back caching and differential durability eliminate I/O operations significantly.

Steve Kleiman from NetApp asked whether the timestamp was updated for each write, since it would double the number of writes. Dutch Meyer, who was sitting in the auditorium, said that it was turned on. Steve Byan from NetApp asked whether they noticed any drive scan activities. Jake said that they found some and also found some defragmentation activities. Keith Smith (NetApp) encouraged people from other companies and universities to ask questions and asked why the write-back window of 10 minutes was used and whether it came from stream analysis. Jake replied that they did analyze the data trace and found 10 minutes is a sweet point to coalesce the writes. A researcher from VMware asked about the overhead of cache mechanism, especially of memory. Jake replied that their cache was located in disk, not in memory. She asked about the CPU overhead of the cache mechanism. Jake replied that their mechanism did introduce a little overhead by looking up the cache in the local disk.

Exploiting Half-Wits: Smarter Storage for Low-Power Devices

Mastooreh Salajegheh, University of Massachusetts Amherst; Yue Wang, Texas A&M University; Kevin Fu, University of Massachusetts Amherst; Anxiao (Andrew) Jiang, Texas A&M University; Erik Learned-Miller, University of Massachusetts Amherst

Mastooreh Salajegheh began by introducing small embedded devices. Although these devices are small, the market for them is huge, so it is important to solve the challenges they pose.

The flash for these devices is small and embedded in the same chip as the CPU, but since the voltage requirements for the CPU and flash are different, they should ideally use separate power lines. In actual practice, however, they share

the same power line, which results in excessive energy consumption.

Mastooreh said that the goal of their research is to reduce the energy consumption for embedded storage. Traditionally, there are two ways to set the voltage for embedded systems: pick the highest voltage, which results in a huge energy consumption, or dynamically scale the power, which results in complex design. They suggested another approach—to write to flash with low voltage. However, there are occasional failures with this approach, and she gave an example. Then she introduced three factors which may influence the error rate: operating voltage level, hamming weight of data, and wear-out history. She showed their testbed and the error rates for different settings. With different operating voltages, the error rate is nearly 100% at 1.8V, but it drops sharply when the voltage changes to 1.85V. The error rate reaches 0 well before 2.2V, which is the voltage suggested for the microcontroller. She also showed that the heavier the hamming weight, the smaller the error will be, since there are fewer 1s which need to be changed to 0s.

By designing an abstract model of flash memory and introducing the accumulative behavior of the flash memory, they developed several ways to cope with partial failure. One is in-place writes which repeatedly write to the same location. The idea is to recharge the cell repeatedly until it is written correctly. She illustrated the error rates for different voltages with different number of sequential in-place writes. The effect of in-place writes did show up when the number of writes was increased. The second approach is multiple-place writes. The basic idea is to write to multiple locations so that reliability is increased. She also showed the results of this approach.

After this, she compared energy consumptions for three operations—RC5, retrieve, and store—at different voltages: 1.8V, 1.9V, 2.2V, and 3.0V. The RC5 and retrieve operations required less energy than the store operation. She hypothesized that CPU-bounded workloads, which have fewer store operations, should save energy. Then she introduced their synthetic application, which reads 256 bytes, does an aggregation of these bytes, and writes the result back. For this workload, their scheme saves as much as 34% energy. At the end of her talk, she suggested two further improvements: store the complement of a data item if it has a light hamming weight; use a mapping table to map frequently used numbers to numbers which have heavier hamming weights.

A researcher from EMC was confused by the distinction between 1.8V and 1.9V and asked why 1.8 is a useful number and not the higher 1.9. Mastooreh answered that they were trying to find the voltage that worked best and were not interested in using a voltage where an error did not happen. William Bolosky from Microsoft Research suggested that for in-place writes, it might be possible to read the data back and check to see whether it is written correctly before a second write. Mastooreh agreed and said this is what they have done and that measurements reflect the savings from this feedback system. Another researcher was curious about whether the authors had investigated the temperature factor for error rate, since high temperatures improve flash memory reliability while low temperatures result in a higher error rate. Mastooreh replied that they did consider the temperature factor and did see a decreased error rate when they increased the temperature. She admitted that they had not put the chip in a fridge, but they expected the error rate to be increased at low temperatures. A researcher from EMC asked whether it is difficult to do a read after write to verify whether the data was correctly written. Mastooreh said that it is easy for flash memory (especially the smaller ones) to check the result of write operations. This researcher pointed out that the worst bound of rewrite may take more energy than what would be saved. Mastooreh agreed and said that it depends on the voltage the chip is working on. Another researcher asked whether the authors had any ideas about predetermining the number of in-place writes for applications. Mastooreh answered that it depends on the chip, and if the application developers tell them what error rate they want, they can set parameters for them.

Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory

Shivaram Venkataraman, HP Labs, Palo Alto, and University of Illinois at Urbana-Champaign; Niraj Tolia, Maginatics; Parthasarathy Ranganathan, HP Labs, Palo Alto; Roy H. Campbell, University of Illinois at Urbana-Champaign

Shivaram Venkataraman began with the scalability wall of DRAM, which prevents it from scaling for future computer architectures. There are several new memory technologies such as Phase Change Memory (PCM) and Memristor, non-volatile memory devices in which the access time is projected to be in 50–150 nanoseconds, which is much nearer to DRAM access time than Flash's. With non-volatile memory replacing traditional storage architectures DRAM and disk, only a single copy of data will exist in the system. They called such storage architectures single-level stores.

Shivaram demonstrated that it is not easy to maintain the consistency of a binary tree based on these new stores. To deal with such challenges, he introduced "consistent and durable data structures," defining four properties for such structures: versioning, restoration, atomic change, and no processor extensions. He discussed how they implemented

a consistent and durable B-tree based on these guidelines. They chose B-tree because it is a complex data structure and is widely used in file systems. In addition to keeping a data value in a node, they introduced start and end version numbers to keep version information. Then he walked through lookup and insert/split operations and explained how version number is used and how end version is calculated.

He then explained how they incorporated their consistent and durable B-tree into the key-value store system, Redis, producing their system, Tembo. It is rather easy to integrate their consistent and durable B-tree into other systems; in their case, it only required modifying 1.7% of Redis. They evaluated their system at three levels: micro-benchmarks, end-to-end comparison, and real cluster deployment. They found that for small value size such as 256 bytes, a hashtable with logging performs better than their consistent and durable B-tree. But when the data size becomes 4K, their B-tree achieves higher throughput. For end-to-end comparison, they used the Yahoo Cloud serving benchmark, Cassandra. Tembo outperformed Cassandra in-memory by 286%. He also mentioned additional details in their paper, such as deletion algorithms and analysis for space usage.

Edward Denali (UC Berkeley) asked about the advantages of the authors' approach when compared with transaction memory. Shivaram acknowledged that transaction memory is a way to provide consistent updates but it is too heavy-weight for certain applications. A researcher from NetApp asked whether the garbage collector will produce more garbage. Shivaram replied that it will not, since they can safely remove all versions before the last consistent version. Margo Seltzer (Harvard) asked how their version compared to B-trees of 30 years ago. Shivaram said they didn't want to include old data. Seltzer persisted, pointing out that multi-versioning concurrency control is really a very old technique. Shivaram replied that even for a single update, they can go back and safely recover, do rollbacks. Seltzer responded that multi-level concurrency could do this 30 years ago.

Ethan L. Miller from UCSC found this work very interesting and asked whether they did an error test in the middle of the flush process. Shivaram said it is one part of their current work to improve system robustness. A researcher from Stanford asked whether their key-value system handles node failure when they compared their system with other key-value systems. Shivaram replied that they replicate data to multiple nodes, using consistent hashing, but did not perform any experiments with failures. Michael Condict from NetApp pointed out that for read-only portions of programs and memory-mapped files, there is only one copy of data either in RAM or disk. So these are also a form of single-level store and there's no need to wait another two years for new memory technologies.

Flash

Summarized by Puranjoy Bhattacharjee (puran@vt.edu)

CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives

Feng Chen, Tian Luo, and Xiaodong Zhang, The Ohio State University

Feng Chen described SSDs, discussed their merits, and highlighted the concern of limited lifespan caused by limited program/erase (P/E) cycles. He talked about the prevalence of redundancy-based solutions for wear-leveling in Flash Translation Layers (FTL), but noted that prior work has been inconclusive regarding the benefits of this approach. Chen presented a formula for endurance of a flash-based system: Endurance = (CxS)/(VxE), where C = P/E cycles, E = Efficiency of FTL designs, V = Write volume per day, and S = available space. Thus, Chen pointed out that flash endurance can be increased if V is reduced and S increased. The insight here is that data duplication is common, and deduplication can help reduce V. In addition, coalescing redundant data will help increase S, thus leading to higher endurance.

Chen talked about the challenges in implementing such a scheme, such as availability of only block-level information and lack of any file-level semantic hints for deduplication. To overcome these, CAFTL uses a hash function and fingerprinting to identify duplicate blocks. However, the authors observed that most fingerprints are not duplicates; hence they need to store the fingerprints for the ones which are most likely to be duplicates. The other challenge is for reverse mapping from PBAs to LBAs (since there is an N-to-1 correspondence between the logical and physical block addresses because of deduplication); Chen described their solution, VBA (virtual block address). He also talked about an acceleration method of sampling of pages for hashing based on the idea that if a page in a write is a duplicate page, the other pages are also likely to be duplicates. He discussed various sampling strategies and talked about their content-based scheme, where the first four bytes of a page are selected for sampling. He then talked about the evaluation based on an SSD simulator for desktop and office workloads and on TPC-H benchmark, leading to a dedup rate of 4.6% to 24% and space savings of up to 31.2%.

This was a popular talk and there were a lot of people queuing up to ask questions, but only a few got to ask because of time constraints. Assar Westerlund (Permabit) asked about the estimate of storing overhead information on the flash and

RAM. Chen replied that it is on the order of 10s of MB for 32 GB flash drives. He said that other techniques, such as the FTLs described in the next paper, could be used. In addition, he said that SSD manufacturers maintain that the most significant production cost is flash chips. So, adding more RAM could be a solution. Peter Desnoyers (Northeastern U.) talked about SANForce controllers which appear to be doing dedup similar to the CAFTL scheme. Chen said that they had not looked at it, but agreed it was a nice idea and said that runtime compression could provide more space savings, at the cost of more complexity. Raghav (Penn State) asked how old VBA values are updated when a new unique value comes in pointing at the same VBA. Chen answered that there is an offline deduplication method and it is done when the flash is idle.

Leveraging Value Locality in Optimizing NAND Flash-based SSDs

Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam, The Pennsylvania State University

Aayush Gupta presented the authors' look at different dimensions of value locality and how to develop a new kind of SSD called CA-SSD based on that. He discussed temporal and spatial localities but focused on another form of locality, called value locality, where certain content is accessed preferentially and can be exploited by data dedup using content-addressable storage. He discussed out-of-place updates and loss of sequentiality of writes, challenges which need to be overcome. However, this analysis works only if real workloads exhibit value locality. To quantify this, he talked about a new metric the authors introduced, value popularity, which indicates the potential of a workload to be value-local. They found that 50% of writes have only 8% unique values, so real workloads are indeed value local. He then described the design of their system, which added a hash co-processor to prevent that from being the bottleneck, and also added persistent storage with battery-backed RAM. Simply stated, when new content arrives, compute the hash and lookup if it is new or existing; if existing, update the data structure, otherwise write the page.

To implement this, Gupta talked about maintaining two tables, iLPT (inverse logical to physical table) and iHPT (inverse hash to physical table), in addition to LPT and HPT. He then described the metadata management policy used in CA-SSD. Since three extra tables might not fit in RAM, they exploited the property of temporal value locality. They store only a few entries in the HPT and iHPT, and discard using an LRU (least recently used) policy. This raises the possibility of affecting the dedup rate but not the correctness of the policy; in any case, experimentally they did not observe any dedup degradation. Gupta then discussed their evaluation strategy using Web workloads. He said that workload writes and garbage collection writes were both reduced, leading to improved lifetimes.

Someone asked how their work differed from least popular value replacement work published earlier. Gupta said they were not aware of this. Jonathan Amit (IBM) wanted examples of value locality in real applications. Gupta pointed out spam deliveries to all users in a mail workload. Mark Lillibridge (HP) and Gupta then discussed the different kinds of localities. Mark also pointed out the need for a distinction between static and dynamic deduplication, and Gupta agreed. Dongchul Park (U. Minn) asked about the recovery procedure when power is cut down before all the tables can be updated, and Gupta replied that they roll back.

Reliably Erasing Data from Flash-Based Solid State Drives

Michael Wei, Laura Grupp, Frederick E. Spada, and Steven Swanson, University of California, San Diego

Michael Wei discussed the reasons why SSDs pose problems for erasing data with confidence. SSDs are new devices, and the existing erasing techniques are tailored for hard disks. The presence of FTLs poses another problem, since the OS is not aware of the actual data layout on the flash drives. There is the problem of incorrect implementation of FTLs, because there are many manufacturers. Since FTLs are cheap and easier to disassemble/reassemble, Wei contended that someone could steal data overnight from an SSD. Wei then presented some background on sanitization, following which he described their custom hardware platform to validate sanitization efficacy of various techniques and reported the results. Built-in commands were unreliable, since one disk reported data as erased when it was not. The technique of crypto-scramble encrypts data online but at the same time makes drives unverifiable, so Wei said they could not vouch for this technique. Finally, with software overwrites from various government standards, Wei said that sometimes two passes of overwrites were required; at other times, even 20 passes were not enough, thus rendering this technique unreliable as well.

Wei then discussed their technique of scrubbing—an enhancement to FTL for single file sanitization. He found that flash devices can be made to program pages in order, with some restrictions for single-page overwrites to work. With multi-level cells (MLCs), however, there is a limit on the number of scrubs possible, called the scrub budget. Three modes of scrubbing are possible: immediate, background, and scan. Wei then discussed the different sanitization levels achievable with the different modes. It was found that SLC

scrubbing can be done in a couple of seconds, but MLCs take longer.

Nitin Agarwal (NEC) wondered if scrubbing causes disturbances in nearby cells. Wei said that scrubbing works by writing a single page to all zeroes and that it works fine for SLCs and for some MLCs. As a follow-up, Agarwal asked whether they had any idea what a reasonable scrub budget was. Wei said that this varied from manufacturer to manufacturer, but that they had not tested how many exactly. He added that they had not seen any corruption on adjacent pages. Keith Smith (NetApp) suggested integrating their approach throughout the file system stack to avoid problems where file systems did not store new data in place of the old data, resulting in stale blocks. Wei agreed that this was one way to improve the process, but said they did not have time to modify the file system. Michael Condict (NetApp) asked if the authors had explored techniques which would not involve increasing the error rates, such as immediate block copies. Wei said this would result in too much latency. Finally, Peter Desnoyers (Northeastern U.) pointed out that hardware to steal data from flash drives is available for a thousand dollars or so. In addition, he wanted to know the block size, but Wei did not know. Peter also wondered what hardware supportsthe authors would ask for if they could and Wei had suggestions for switching mechanisms in the FTL to use the three methods of sanitization along with new command sets.

The Disk Ain't Dead

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

A Scheduling Framework That Makes Any Disk Schedulers Non-Work-Conserving Solely Based on Request Characteristics

Yuehai Xu and Song Jiang, Wayne State University

A work-conserving scheduler is one in which disk requests are scheduled immediately, while non-work-conserving schedulers instead choose to delay some requests when doing so may lead to a better schedule. Despite the potential delay, the tradeoff is sometimes attractive because it can better eliminate costly seeks in magnetic disks. Current non-work-conserving schedulers, like the Linux anticipatory scheduler, take advantage of process information to predict the locality of future requests, but that information is not available in some environments. In response to this deficiency, Song Jiang presented a new scheduling framework based exclusively on observed characteristics of incoming requests.

Their approach is called stream scheduling. The authors observe that requests can be grouped into streams based on locality. A stream is defined as a sequence of requests for which the judicious decision is to wait for more requests. Streams can be identified as sequences of requests where intra-stream locality is stronger than inter-stream locality, and which grow as requests within a certain range of each other are queued. Streams are ended when the time allotted to them expires, an urgent request arrives, or the stream is broken by a request that does not exhibit locality. Song also showed the evaluation of a Linux prototype stream scheduler, which was shown to have a factor of 2 or 3 performance increase in scenarios where multiple independent workloads are run in parallel.

Arkady Kanevsky of VMware asked how well stream scheduling scaled with number of streams. Song acknowledged that the wait times could grow long but said that in such a scenario, other non-work-conserving schedulers (such as AS and CFQ) would have problems as well. Currently, there's no general solution for this problem, but Song suggested that a high QoS requirement might still be met using multiple disks, as different disk heads could then serve different streams. Vasily Tarasov from Stony Brook University asked how stream scheduling scaled to multiple queues. Song answered that stream scheduling shows an advantage as the number of clients increase relative to the number of queues. However, an implementation artifact in NFS led them to measure this effect using multiple NFS servers, as opposed to multiple threads on the same server.

Improving Throughput for Small Disk Requests with Proximal I/O

Jiri Schindler, Sandip Shete, and Keith A. Smith, NetApp, Inc.

Jiri Schindler presented his work on proximal I/O, a disk access pattern for small updates to data sets that require both good sequential read performance and frequent modification. Existing disk layouts require in-place updates for efficient sequential access, while small updates are better served by log-style appends. Unfortunately, these two layouts are at odds. Log-style appends fragment on-disk layout, which makes serial reads inefficient. This type of mixed workload is extremely challenging to system designers and occurs in a number of practical workloads, such as when data is written in small transactions, then read in bulk for business intelligence purposes.

Schindler and his team developed a new approach that uses a flash-based staging buffer for writes, which are then committed to disk in batches. The writes are placed near related data to provide the required locality for reads, and the flash-based staging area provides the sufficient I/O density to be able to retire multiple user I/Os in a single disk revolution. Proximal I/O exploits hard disk drive mechanisms that allow for scheduling multiple writes per revolution within

a span of hundreds of thousands of logical blocks of the disk interface. This combination of leveraging magnetic disk behavior characteristics and a small amount of non-volatile memory performs well and is well suited to file systems with no-overwrite semantics such as the write-anywhere method used by NetApp.

Schindler's performance evaluation of proximal I/O demonstrated that the flash could be small—just 1% of the total active working set. He also showed results against an aged file system and a 90% full disk. Despite these disadvantages, proximal I/O was able to provide RAID-1–like performance on a RAID-4 disk configuration that provides parity.

Someone asked how proximal I/O compared to free block scheduling (FAST '02). In free block scheduling, updates could be made to any free block, while proximal I/O employs a minimal staging area to keep the costs of flash low. Schindler concluded that the two techniques were useful in different applications. Aayush Gupta (Penn State) asked if flash lifetimes have been considered. Schindler replied that while writes were random from the application's perspective, they may be placed anywhere on a device, so the aging of flash memory will occur evenly.

FastScale: Accelerate RAID Scaling by Minimizing Data Migration

Weimin Zheng and Guangyan Zhang, Tsinghua University

Guangyan Zhang presented this research that attempts to accelerate RAID scaling, where a new disk is added to an existing RAID set. FastScale works on RAID-0, RAID-01, RAID-10, and RAID-1. Zhang's team started with several goals. While scaling a RAID set, they needed to ensure that data was spread across the disks uniformly, so no hotspots would be created. They also sought minimal data migration, so that the scaling could be done quickly. Finally, they needed to maintain fast addressing, to not impair the overall performance. To meet these requirements, they developed a new approach to migration and on-disk layout.

To reduce the cost of migration, FastScale groups blocks that must be migrated to new disks into groups that can be moved together. This allows multiple blocks to be read with a single I/O request. Blocks are then arranged in memory to form sequential sets of blocks that can be written linearly as well. In addition, by separating the migration process into phases which copy data, and phases which replace existing data, FastScale is able to lazily update the state of migration information. This limits the number of metadata writes required to ensure consistency in the event of a failure.

With a simulator, Zhang showed an 86% shorter RAID scaling time as compared to SLAS, a scaling approach proposed in 2007. In addition, the lower overhead of FastScale was shown to result in lower latency during the block migration process.

Bill Bolosky from Microsoft Research noted that during reorganization, the block address space in the RAID appeared to be made less linear. He asked if sequential I/O performance might be harmed by the approach. The author believed that the 64K block size was large enough to avoid many seeks, but Bolosky disagreed. He suggested they try the approach with a larger block size. Keith Smith from NetApp asked why the authors had not extended the technique to work with RAID-4, which he believed was similar in difficulty to what the team already supported, unlike RAID-5. The discussion was eventually taken offline, where Zhang noted that the percentage increase of RAID-4 scaling with FastScale would be smaller, because there was a constant overhead required to compute parity information.

Wednesday Poster Session

First set of posters summarized by Vijay Chidambaram (vijayc@cs.wisc.edu)

Improving Adaptive Replacement Cache (ARC) by Reuse Distance

Woojoong Lee, Sejin Park, Baegjae Sung, and Chanik Park, Pohang University of Science and Technology, Korea

Sejin Park presented a new block replacement algorithm for second-level caches. The Adaptive Replacement Cache (ARC) algorithm dynamically balances recency and frequency. However, it doesn't take into account the reuse distance of I/O requests and hence doesn't perform well for second-level caches. The authors propose an enhancement to ARC called Reuse-Distance Aware ARC (RARC). RARC maintains a history buffer over all I/O requests and uses a sliding window on the history buffer in order to determine which I/O blocks are kept in the recency queue. The size of the sliding window is equal to the size of the recency queue in RARC.

Email-based File System for Personal Data

Jagan Srinivasan, EMC; Wei Wei, North Carolina State University; Xiaosong Ma, Oak Ridge National Laboratory and North Carolina State University; Ting Yu, North Carolina State University

Xiaosong Ma proposed engineering a highly available storage service based on email services such as Gmail, building upon the reliability of cloud storage. The authors create the abstraction of virtual storage disks on top of email accounts and employ RAID techniques such as space aggregation, data striping, and data replication. They have implemented the

Email-based File System (EMFS), which exports a subset of the POSIX interface and is built on FUSE. They evaluated its performance on the Postmark benchmark. Results show that the performance of EMFS is comparable to that of commercial solutions such as JungleDisk.

Solid State Disk (SSD) Management for Reducing Disk Energy Consumption in Video Servers

Minseok Song and Manjong Kim, Inha University, Incheon, Korea

Minseok Song proposed a scheme for using solid state disks to reduce disk energy consumption in video servers. The zip distribution of block requests allows the use of small SSDs as effective caches for the disks, thereby allowing the system to run the disks at a lower speed. This enables the video servers to reduce their power consumption without a drastic reduction in performance. This raises the question of which videos should be cached on the SSD. The authors formulate this as an integer linear problem, with the solution minimizing energy consumption. Simulation results show up to 33% reduction in power consumption when a 256 GB SSD is used.

Object-based SCM: An Efficient Interface for Storage Class Memories

Yangwook Kang, Jingpei Yang, and Ethan L. Miller, University of California, Santa Cruz

Yangwook Kang pointed out that using SCMs in the storage hierarchy today leads to the choice between major changes to the file system and suboptimal performance. The authors aim to solve this problem using an object-based model, enabling integration of a range of storage class memories into the file system and drop-in replacement of different SCMs. The object-based model also enables performance optimization for each SCM, since more information is known about the I/O requests in this model. The authors have implemented a prototype in Linux and evaluated it using the Postmark benchmark. Results show significant performance improvements when the object-based model is used.

Latent Sector Error Modeling and Detection for NAND Flash-based SSDs

Guanying Wu, Chentao Wu, and Xubin He, Virginia Commonwealth University

Guanying Wu explained that the increasing density of NAND Flash SSDs leads to higher probability of latent sector errors. Latent-sector errors in NAND Flash are not well understood, due to the limited population of SSDs in the field. The authors propose a new model for latent sector errors in NAND Flash SSDs that takes into account the write/erase cycles in NAND flash, retention, and electrical disturbances. The authors also propose an optimized disk scrubbing strategy based on their comprehensive model, which uses different policies such as localized and staggered scrubbing.

Polymorphic Mapping for Tera-scale Solid State Drives

Sungmin Park, Jaehyuk Cha, and Youjip Won, Hanyang University, Korea; Sungroh Yoon, Korea University, Korea; Jongmoo Choi, Dankook University, Korea; Sooyong Kang, Hanyang University, Korea

Sungmin Park proposed a scheme to manage the mapping information for tera-scale solid state drives. For such large drives, the size of the mapping information itself is on the order of gigabytes and new techniques to manage the mapping table in SRAM are required. The proposed mapping scheme, polymorphic mapping, exploits the spatial locality of requests to store only part of the complete mapping information in SRAM, thus reducing the space needed for the mapping. This is done at different granularities from the block to the page. The authors evaluate their system using several real-world workloads. The results show that polymorphic mapping outperforms DFTL in most cases.

DBLK: Deduplication for Primary Block Storage

Yoshihiro Tsuchiya and Takashi Watanabe, Fujitsu Limited

Yoshihiro Tsuchiya presented the Deduplication Block Device (DBLK), a primary-storage device with in-line block-level deduplication. DBLK uses an innovative multi-layer Bloom filter (MBF) in order to index the data in the system. The MBF works like a binary tree, with a Bloom filter present at each node. The system uses techniques such as bitwise transposition in order to optimize the Bloom filter access. Micro-benchmarks show that DBLK's performance is comparable to the base RAID system, and that using the MBF to index metadata leads to reduced latency.

Implementing a Key-Value Store based MapReduce Framework

Hirotaka Ogawa, Hidemoto Nakada, and Tomohiro Kudoh, AIST, Japan

Hirotaka Ogawa presented SSS, a MapReduce system based on distributed key-value stores. SSS seeks to eliminate the shuffle and sort phase of MapReduce using the properties of the key-value stores, while enabling intuitive access to the MapReduce data. Building SSS on distributed key-value stores makes Map and Reduce tasks equivalent, enabling combinations of multiple maps and reduces in an individual workload. The authors evaluated their prototype using the wordcount benchmark. They demonstrated performance comparable to Hadoop. The results also show that an optimized version of SSS is almost three times faster than Hadoop.

Hot and Cold Data Identification for Flash Memory Using Multiple Bloom Filters

Dongchul Park and David H.C. Du, University of Minnesota—Twin Cities

Dongchul Park pointed out that current schemes for identifying hot data in flash memory focus only on the frequency of data access, ignoring the recency of the data access. The authors present a new classification algorithm, window-based direct address counting (WDAC), which also takes into account the recency of data access. Multiple Bloom filters are used in order to efficiently capture recency and frequency information. The key idea is that each Bloom filter uses a different recency weight to capture fine-grained recency. Experimental results show that WDAC improves performance by 65%, with lower overhead and memory requirements.

Second set of posters summarized by Deepak Ramamurthi (scdeepak@cs.wisc.edu)

An FCoE Direct End-to-End Connectivity Scheme

Michael Ko, Li Ke, Wang Yuchen Wu Xueping, Yang Qiang, Liu Lifeng, Meng Jian, and Yang Qinqin, Huawei Symantec Technologies Co., Ltd.

Michael Ko explained how the forwarder becomes a bottleneck in a Fibre Channel over Ethernet (FCoE) network, when it is required to mediate all data and control information, including those between FCoE end nodes. The authors propose the introduction of a fabric login wherein each end node is assigned a unique Fibre Channel identifier. All data plane functions between the end nodes use this identifier to communicate directly, thereby bypassing the FCoE forwarder. This eases the load handled by the FCoE forwarder. As native FCoE devices become commonplace, the forwarder is no longer a time-critical component and can be implemented in software.

What makes a good OS page replacement scheme for Smart-Phones?

Hyojun Kim, Moonkyung Ryu, and Umakishore Ramachandr, Georgia Institute of Technology

Hyojun Kim explained the need for OS-level support for better management of low-end flash storage used in smartphones. The authors present data to show that write request ordering is a key factor in the performance of flash storage, with sequential writes offering better performance than random writes. They present two write ordering-aware page replacement algorithms—Sorted Clock and Sorted Clock-WSR (Write Sequence Reordering)—and quantitatively show that they perform better than replacement policies that do not respect write ordering.

Accelerating NFS with Server-side Copy

James Lentini, Anshul Madan, and Trond Myklebust, NetApp, Inc.

James Lentini presented a new NFS server-side file copy interface that does away with the two-step approach by first fetching the file from the server and then writing it again to the destination location on the server. This single message contains the source and destination information, which saves a lot of client-side and network resources. Applications of such a mechanism include VM backup, cloning or migration, and file backup restoration.

Cluster Storage System, RAID and Solid State Drive Emulation with David

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

Leo Arulraj presented interesting extensions to David, a lightweight storage system emulator that currently allows the emulation of large disks by much smaller disks. Incorporating complex RAID setups into the emulator and on-the-fly generation of parity is one possible extension. Extending David to emulate SSDs is not trivial, because of the difference in latencies of I/O from an SSD and a disk.

Tamias: a privacy aware distributed storage

Jean Lorchat, Cristel Pelsser, Randy Bush, and Keiichi Shima, Internet Initiative Japan, Inc.

Jean Lorchat presented a distributed storage system that enables users to share data while giving them fine-grained control over the sharing process. The authors use Tahoe-LAFS, an open source storage system, as a foundation for their system, and build in additional features such as capabilities and user identification. They achieve capabilities through public-key cryptography. They maintain a directory of public keys of friends and use this information for access control.

Mercury: Host-Side Flash Caching for the Datacenter

Steve Byan, James Lentini, Luis Pabón, Christopher Small, and Mark W. Storer, NetApp, Inc.

Steve Byan and his team propose Mercury, a system that uses flash storage as a cache in a datacenter environment. With increasing amounts of flash being integrated into shared VM servers, they can be used for storage purposes. But using them as primary storage would break the shared storage model of a datacenter. The authors have built a block-oriented write through cache with this flash storage. Their results show that this leads to a very healthy reduction in mean I/O service time as well as in the number of requests sent to the server.

DADA: Duplication Aware Disk Array

Yiying Zhang, University of Wisconsin—Madison; Vijayan Prabhakaran, Microsoft Research

Yiying Zhang presented DADA, a duplication-aware disk array which identifies and retains the duplicated blocks on primary storage systems. This additional knowledge is put to use in different ways. Disk background activities like scrubbing are made faster by scrubbing only a unique block among duplicates and skipping the rest. Recovering only unique blocks improves RAID reconstruction time and hence improves availability. She also explained some of the avenues explored in choosing a representative block among the duplicates, which included a simple random selection policy to more complex heuristics based on region density of unique blocks.

MAPL: Move-anywhere proximate layout

Peter Corbett, Rajesh Rajaraman, Jiri Schindler, Keith A. Smith, and Pelle Wåhlström, NetApp, Inc.

Keith Smith presented MAPL, a journaling file system that tries to preserve the physical sequentiality of logically related blocks of data. By achieving this, the file system is able to provide good performance even for sequential-read-after-random-writes workload. MAPL allocates regions of the disk to a file. Space is over-provisioned for a file so that a portion of the file, called the *snapshot reserve,* is used to accommodate logically overwritten blocks of data in a snapshot. Once a region fills up, the snapshot data is copied out in bulk to a different region of the disk. Good performance is achieved by a sequential read of a full region for actual random reads. The cost of the copy-out of snapshot data is also amortized.

Scaling Well

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

The SCADS Director: Scaling a Distributed Storage System Under Stringent Performance Requirements

Beth Trushkowsky, Peter Bodík, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson, University of California, Berkeley

Beth Trushkowsky started her talk by introducing the elasticity to scale up and down provided by clouds and turned to talk about her work on how to dynamically allocate storage resource for workload changes. The challenge for this is that they must satisfy service level objectives (SLO). Then she showed a trace of Wikipedia workload which shows that hits are rather bursty. If they were to replay it on a system with 10 storage servers, the hotspot would affect one of those 10 servers. For this hypothetical example, the system would need to be over-provisioned by 300% to handle a spike of this magnitude on any data item. So it is important to leverage the elasticity from cloud storage.

Beth then introduced SCADS, an eventually consistent key/value store that has three features: partitioning, replication, and add/remove servers. All the data is kept in memory, which avoids the latency of disk accesses. With SCADS, it seemed quite straightforward to use classical closed-loop control for elasticity. But she illustrated that closed-loop control cannot handle oscillations from a noisy signal well, and smoothing can result in a long delay to react to spikes. In order to solve these problems, the authors proposed a new control framework called model-predictive control (MPC). This framework uses per-server workload as a predictor of upper-percentage tile latency and needs a model to map the workload to SLO violations. The MPC knows the current state of the system and can determine actions to be taken to meet constraints. Beth then showed how to transition from the classic closed-loop control to the MPC model. The upper-percentage tile latency is replaced with a workload histogram, and the controller refers to the performance model to decide actions. The authors built the performance model by benchmarking SCADS servers on Amazon's EC2. Using this performance model, the MPC can decide when and where to migrate the data, but it still does not know which piece of data is hot and should be migrated. So, they introduced the idea of finer-granularity workload monitoring. The benefit from this approach is that hot data can be identified and moved back and forth much more quickly. With both the performance model and fine-grained monitoring, it can determine the amount of data to be moved, what the hot data is, and when to coalesce servers. Beth also pointed out that they used replication for prediction and robustness; she referred interested audiences to read their paper for more details.

Next she illustrated how their system decides the amount of data and the idle server to be moved. Then she introduced the experiment setup and workload profiles for Hotspot and Diurnal. The aggregate request rate appeared to be flat, but the per-bin request rate increased sharply at the beginning for the hot bins. For the other 199 bins, the per-bin request rate remained almost constant. Their system dynamically replicated hot data among more servers, which succeeded in keeping the 99% tile latency almost constant.

Fred Douglis from EMC asked whether it is possible to use past behavior to predict future workload. Beth replied that the experiment they showed today involved observing the workload and reacting directly to it but that it is possible to use yesterday's workload trace as input to the controller. Brent Welch (Panasas) asked about the cost model of moving data around the servers. Beth agreed that if moving data

is too costly, you might need to trade off the decision about scaling up or scaling down. Brent continued to point out that the cost models of SLO and moving data are different and encouraged Beth to look into that model in the future. Margo Seltzer (Harvard) said that from their experience, they can see a stable behavior for minutes or hours but the variability of day-to-day behavior is huge. She asked whether the authors have noticed this phenomenon and how this variance will affect their system's behavior. Beth replied that they did several things to deal with this. Although they did not evaluate the performance on different days, they built their model on different days and their experiments ran successfully. They used replications to improve the stability of their system. Margo asked what factor of replication they used. Beth replied they used a replication factor of 2 in their result. Xiaodong Zhang from OSU said moving data is expensive and caching is more effective. Beth replied that their system keeps data in memory and functions as if it were adding another cache.

Scale and Concurrency of GIGA+: File System Directories with Millions of Files

Swapnil Patil and Garth Gibson, Carnegie Mellon University

Swapnil Patil started his presentation by noting that checkpoints in supercomputing require each thread to write states periodically into a new file. As a result, it requires huge directories. He also pointed out that future massively parallel applications and diverse small file workloads may need huge directories. File systems today provide parallel access to the file data but not to the metadata. So they decided to build a scalable and parallel file system directory subsystem. Their goal is to support high file insert rates in directories and complement existing cluster file systems by providing POSIX-like semantics. Swapnil illustrated how small and large directories are stored on three servers: the smallest directory is hosted by a single server, while the largest directory is split into three partitions based on a hash scheme, and each partition is stored at one server. Then he discussed how the GIGA+ client and server components work and interact and showed that GIGA+ can achieve 98K file creates/s, which far exceeds the HPCS goal of 32K. He also compared the scalability of GIGA+ with HBASE and Ceph, and GIGA+ has much better scalability than both of them.

With the observation that directories start small and most remain small, directories in GIGA+ start with one partition on one server. When the number of files in a directory reaches a threshold, the hash space is split into two equal halves and half of the files are moved to a second server. It continues this split repeatedly when the directory size keeps increasing. The uniqueness of the split process in GIGA+ is that splits can happen concurrently at multiple servers without any synchronization. Swapnil described the performance of GIGA+ during the incremental splitting phase as dropping once until all the servers are used, then scaling linearly. He also mentioned that interested readers should look at the paper for additional details.

Assar Westerlund (Permabit) pointed out that for checkpoints, the directories do not shrink. He asked whether their system will rebalance for shrinking. Swapnil replied that currently they do not consider shrinking, but it is possible to deal with this. Servers can shrink their directories and the client mapping will be updated lazily. Thomas Schwarz from UCSC pointed out that Ceph's performance stats cited in the paper were five years old and that Ceph has been actively developed and has many improvements; why didn't the authors compare the scalability of the latest version with GIGA+ directly? Swapnil said that by default, Ceph does not support distributed directories. They tried to run it but, due to time pressure, they could not complete their evaluation. They do believe the latest Ceph will have better scalability than the old version. A researcher said he understood the goal of this work and asked why not just ask the programmers not to create so many files in one directory. Swapnil replied that some of the applications are legacy and it is better to provide file system support for them. What will the performance be when reading or deleting all the files? Swapnil replied that doing a read of millions of files is interesting, because you will get two-day vacations. A researcher from NetApp asked whether it will cross partitions when renaming a file. Swapnil said it is possible and they can reuse the atomic renaming function provided in the cluster file systems.

AONT-RS: Blending Security and Performance in Dispersed Storage Systems

Jason K. Resch, Cleversafe, Inc.; James S. Plank, University of Tennessee

Jason Resch introduced the basic idea of dispersed storage, which is to computationally massage data into multiple pieces and store them at different locations. It allows owners to reconstruct the original data by using any K out of N pieces. K and N can be arbitrarily chosen. The reliability of a storage system is improved since only K pieces are needed to reconstruct the original data, and it allows for disaster recovery without replication. It also provides for strong security, as discussed later.

The conventional approach to storing data securely is to encrypt it, but this requires users to protect and store their keys. Another problem with this approach is that users will lose their data if they lose their keys, and increasing the reliability of keys by replication opens more opportunities for attacks. In 1979 Adi Shamir and George Blakely inde-

pendently discovered a better way, called Secret Sharing, in which a secret is divided into N shares. You can get the secret with any threshold (K) number of shares. However, you cannot get any information about the secret with fewer than K shares. This provides a high degree of privacy and reliability. Actually, encryption is a specific case of secret sharing, where N = K = 2. However, Secret Sharing has several drawbacks which prevent it from being used for performance- or cost-sensitive bulk data storage. The first drawback is the storage overhead. For Shamir's scheme, it requires N times storage and bandwidth, while for Blakely's method it is even worse: N*K times. The encoding time is another drawback. It grows with N*K. To deal with these two challenges, Michael O. Rabin designed another method, called Information Dispersal Algorithm (IDA), which can achieve efficiency, security, load balancing, and fault tolerance. Storage requirements are reduced to be (N/K) times of original data and (N/K) can be chosen close to 1. However, the security of Rabin's method is not as strong as Shamir.

In 1993, Secret Sharing Made Short (SSMS) was designed by Hugo Krawczyk by combining Shamir's Secret Sharing with Rabin's IDA. It works as follows: first, the input is encrypted with a random encryption key; the encrypted result is then dispersed using Rabin's IDA, while the random key is dispersed using Shamir's Secret Sharing. The result of this combination is a computationally secure secret sharing scheme with good security and efficiency.

After this introduction to related work, Jason discussed their new scheme, called AONT-RS. It combines Ron Rivest's All-or-Nothing Transform with Systematic Reed-Solomon encoding. The security and efficiency properties are similar to SSMS. However, it has another four properties: faster encoding, protected integrity, shorter output, and simpler rebuilding. In the All-or-Nothing Transform it is trivial to reverse the transformation once one has all the output, but it is very difficult to invert without all of the output. So by combining an All-or-Nothing Transform with Reed-Solomon, a computationally secure secret sharing scheme was achieved. Jason then showed a complete view of their system. AONT is used to pre-process the data and IDA is used to create N slides. Without a threshold number of slides, it is impossible to recreate the original data. If there are some bit flips in the encrypted packets, then you will get a different hash value and a different key to decrypt the packets. You can easily tell this from the decrypted canary value.

The observed performance and the expected performance derived from their performance model for AONT-RS, Rabin, and Shamir were shown. The authors implemented two versions of AONT-RS, optimized for performance and security, by using different ciphers and hash functions. Jason described a real-world deployment of their system for the Museum of Broadcast Communications which spans three power grids.

Mark Lillibridge (HP Labs) asked why not encrypt the data and only distribute the key with secret sharing, since keys are much smaller. Jason replied that in order to provide high availability and reliability of the data, they had to ensure the security of both the keys and the data. If the data is stored in a single or unsafe storage system, it may be corrupted or deleted, leaving you unable to restore the data. Bill Bolosky (Microsoft Research) asked why they used a secure hash in their AONT and suggested CRC might be enough. Jason said that secure hash ensures the integrity of their data and agreed that if AES were used to do encryption, the CRC could be enough to do the hash. Assar Westerlund (Permabit) suggested that it might be more helpful if the authors could present their threat model more clearly and pointed out that with their scheme, it would be possible for the storage providers to get together to figure out the information. Jason acknowledged this and said they had described their threat model in their paper. Their threat model is for enterprise storage which wants to maintain control of their data and does not want to recover data from node failures. Organization managers do not need to worry about encryption keys which can be taken and used by their employees.

Making Things Right

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

Emulating Goliath Storage Systems with David

Nitin Agrawal, NEC Laboratories America; Leo Arulraj, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Awarded Best Paper!

Leo Arulraj presented David, an emulator for evaluating new storage systems in this Best Paper award winner. Arulraj explained that researchers face a dilemma in trying to measure future storage systems with current hardware. David is designed around the observation that most benchmarks measure performance but require only valid, not exact, file content. The emulator works by storing metadata but throwing away and regenerating fake data content as needed. In doing so Arulraj showed how they could accurately emulate a 1 TB disk using an 80 GB disk, or a multiple disk RAID with only RAM.

A key requirement of the system was being able to accurately classify blocks as data or metadata. Two such methods were provided. For ext3, David uses implicit classification, where the stream of requests is monitored and knowledge of the file

system's operation is used to make the correct classification. For Btrfs, an explicit classifier was shown, which passes hints through the file system stack. In addition to classification, David must also model device behavior. In evaluating the system, Arulraj showed how they could model both device and I/O request queue behavior. Together these techniques allow a file system evaluator to accurately emulate, and thus measure, the performance of a new file system on faster and larger hardware than they have access to. As future work, the team intends to model a storage cluster with just a few machines.

During the question period, Arulraj explained that they must exert great effort to make the correct classification. Nitin Jain from Riverbed Technology asked if file system–level caching complicates block classification decisions. Arulraj explained that no decision will be made about an unclassified block until it can be made definitively. Geoff Kuenning from Harvey Mudd asked if explicit classification would incorrectly identify data blocks written to the journal as metadata if full journaling was enabled. Arulraj explained that they would need to use knowledge of the file system's operation in addition to explicit classification in that case. Vasily Tarasov from Stony Brook University asked about positioning David lower in the stack, below the I/O request queue. Arulraj explained that being above the request queue provides an opportunity to speed up benchmarking through emulation.

Just-in-Time Analytics on Large File Systems

H. Howie Huang, Nan Zhang, and Wei Wang, George Washington University; Gautam Das, University of Texas at Arlington; Alexander S. Szalay, Johns Hopkins University

Howie Huang presented research that addresses the difficulties of quickly generating statistics about a large file system. As a motivating example, Huang imagined a border patrol agent tasked with checking travelers' laptops for pirated media. The analytics engine developed at George Washington University, called Glance, delivers approximate results to aggregate query processing and top-k style queries. This gives quick answers to questions like, "What is the total count of a specific type of document?" or "What are the top 100 files in the file system that meet some criteria?" respectively.

The system works by walking the tree structure of a file system on a random path while evaluating the query. Each random walk constitutes a trial and the expected result of each of those trials is the actual value in the file system. The problem with this approach is that high variance can result in inaccuracy. Huang elaborated on two techniques to decrease variance. In the first, his tool visits all directories high in the tree to eliminate high-level leaf nodes that might end a trial early. In the second, the random walk is expanded to include random subdirectories in a breadth-first search manner. The system was evaluated on an impressive set of traces, including file systems up to 1 billion files, assembled from a Microsoft file system study, a pair of 2 million file traces from Bell Labs, a 2.4 million file NFS trace from Harvard, and a synthetic file system generated by impressions.

The Q&A was lively. Michael Condict from NetApp noted that the accuracy of the results should be independent of file system size, as is true in calculating a margin of error of a poll. Margo Seltzer from Harvard asked how the work compared to online aggregation estimates research in the database community. Bill Bolosky from Microsoft Research noted that many distributions in storage are highly skewed; he pointed to his own work published earlier the previous day, which showed that 40% of bytes were stored in 0.02% of files. Both Bolosky and the author agreed that you would need many more samples to capture distributions such as this.

Making the Common Case the Only Case with Anticipatory Memory Allocation

Swaminathan Sundararaman, Yupu Zhang, Sriram Subramanian, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

The common paths in a file system's code are usually well tested by users and can be considered hardened against bugs. Unfortunately, rarely executed paths such as error handling routines do not see the same degree of use. When this error handling code has bugs, it can cause serious file-system and application-level errors, sometimes even silently corrupting data. Sundararaman's presentation detailed their approach, Anticipatory Memory Allocation (AMA), and how it can ensure that memory allocation errors are handled correctly inside the operating systems in the context of file system requests.

Sundararaman started by showing how more than a half-dozen Linux file systems responded to fault injection during memory allocation. Each showed some potential to arrive at an inconsistent or unusable on-disk state. Sundararaman and his colleagues proposed to use static analysis to determine the location and factors that determine the size of each memory allocation on the request path, then runtime support to preallocate this memory at the start of the request and to draw from this preallocated pool so that the memory allocation does not fail. This has the advantage of easy recovery, because early in request handling there is little or no state to unwind. It also has the effect of centralizing all of the recovery code, so failures need only be handled in one place. At two points in the talk, Sundararaman relayed the project's mantra: "The most robust recovery code is recovery code

that never runs at all." Before closing, the author showed how his team had minimized the memory overheads by reusing objects in the cache and recycling memory in the preallocation pool.

Jiri Simsa from CMU asked how long the semi-automated approach to determining allocation sizes took, and how human error might be handled. Sundararaman explained that the approach took just a few seconds and could be fully automated. He also explained that it does assume that finding bounds for the memory sizes was decidable, based on the reasonable structure of file system code. Nitin Jain from Riverbed Technology thought it would be interesting to see the difference in the number of outstanding requests that can be sustained, given the memory overhead of the approach, and the author agreed. Danilo Almeida from Likewise Software asked how the tool tracks allocations to the associated memory waiting in the preallocated pool. This led to an explanation that tracking allocations happen inside each request and the called memory-allocation function and its input parameters help determine the type and size of object that needs to be returned back to the calling function.

Work-in-Progress Reports (WiPs)

First set of WiPS summarized by Avani Wildani (avani@soe.ucsc.edu)

New Cache Writeback Algorithms for Servers and Storage

Sorin Faibish, Peter Bixby, John Forecast, Philippe Armangau, and Sitaram Pawar, EMC

Sorin Faibish presented a new cache writeback algorithm for servers, storage, and file systems. Their previous work, published in SYSTOR2010, defined the algorithm and demonstrated that it provided a 30% improvement to performance on their system. The goal of this project is to adapt the algorithm to make it usable in a general environment. They want to change the paradigm of using metrics to prevent I/O bottlenecks and stoppage in systems, some ideas for which were presented at LFS 2010.

So far, they have used measurement tools to pinpoint the problem and have written a Linux utility that will be made publicly available. They have identified room for improvement in ext3, ext4, and ReiserFS. Many applications use aggressive caching, but they are slowed by flushing to disk. They would like to solve memory swap, which is a big issue for cache writeback. Their end goal is to completely move memory swap on Linux. Their results show that the cache writeback took 3200 seconds with 8 GB of memory, 4300 with 4 GB, and never finished at 2 GB of memory. This shows that their intuition is correct. They also demonstrated how their method worked in simulation.

OS Support for High-Performance NVMs using Vector Interfaces

Vijay Vasudevan and David G. Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Vijay Vasudevan began by describing how we have a shrinking I/O gap, and yet the methodologies we use have not caught up to the hardware. Historically, the CPU-I/O gap has increased sharply, but recently this has changed because of the move to multi-core processors and the limit of single thread execution. Additionally, NVRAM is very fast and has become widely available. Modern systems are capable of over a million random I/Os per second instead of thousands, but we still use the same technologies to support these systems! Vasudevan defines this problem as the shrinking I/O gap.

You could reduce access latency, but more and more people are increasing device parallelism. Instead, the authors suggest issuing enough concurrent I/Os to saturate devices whose queue depths are increasing over time. Vasudevan introduced a vector operating system interface where independent threads calling files are replaced by a vector of OS calls. Combining threads into a vector limits context switches, and interrupts, and even reduces the path resolution operations if the files are in the same directory. The questions that remain are, "What are the interfaces?" "What is the impact on the latency?" and "What algorithms will best maintain the queue depth?"

During Q&A someone asked, "If operations are sent simultaneously, how do you aggregate?" Vasudevan replied that this is based on the operating system.

PreFail: Programmable and Efficient Failure Testing Framework

Pallavi Joshi, Haryadi S. Gunawi, and Koushik Sen, University of California, Berkeley

Pallavi Joshi discussed failure testing in large, distributed systems composed of commodity machines. In the cloud era, these systems are prevalent and failures in these systems are both frequent and diverse. The state-of-the-art for testing for such failures is testing single failures and testing multiple random failures to see if the system correctly recovers. Tests against single failures are straightforward, requiring one test execution per failure. For multiple failures, however, there are too many potential sequences of failure to test every sequence separately, necessitating sub-sampling.

Joshi proposed improving on the random selection typically used to do this by identifying the failure paths that most need

testing and avoiding retesting failures that lead to the same recovery behaviors. He outlined an example of testing points where the number of test sequences required to provide full coverage is significantly smaller than the combinatorial maximum. Developers can use this to easily identify the code snippets to test failures and define failure coverage.

Don't Thrash: How to Cache your Hash on Flash

Michael A. Bender, Stony Brook University and Tokutek, Inc.; Martin Farach-Colton, Rutgers University and Tokutek, Inc.; Rob Johnson, Stony Brook University; Bradley C. Kuszmaul, MIT and Tokutek, Inc.; Dzejla Medjedovic, Stony Brook University; Pablo Montes, Pradeep Shetty, and Richard P. Spillane, Stony Brook University

Pradeep Shetty presented an alternative to Bloom filters for efficient cache indexing. Bloom filters are common for efficient indexing but do not scale well out of RAM. Bloom filters are used in BigTable, deduplication, network computing, bioinformatics, and a variety of other applications. These organizations handle the Bloom filter scaling issue using techniques such as MapReduce, LFS/merge trees, etc.

There is an underlying issue to merging two Bloom filters. Particularly, it is difficult to recover hashes and rehash them in a larger Bloom filter. It is necessary to reduce random I/Os, but even buffered Bloom filters result in random I/Os. The solution that Shetty proposed is to use compact hashing, abandoning Bloom filters entirely. Compact hashing, a CPU-bound vs. an I/O bound technique, preserves hashes and stores them in increasing order, allowing iteration. This is defined as a "quotient filter." Similar to merging two sorted lists using merge sort, the quotient filter merges serial cache. With its lookahead array, compact hashing has great scaling properties with inserts and deletes occurring at the bandwidth of the disk. The primary negative is that compact hashing takes twice the disk space of an equivalent Bloom filter. Otherwise, compact hashing is 3000 times faster than Bloom filters, 100 times faster than buffered Bloom filters, and has a comparable lookup throughput.

A File System for Storage Class Memory

Xiaojian Wu and A.L. Narasimha Reddy, Texas A&M University

Wu discussed how to design a file system for storage class, byte-addressable memory. A new file system is needed for SCMs because current file systems are not designed for non-volatile memory devices. For one, they have a very large overhead. Typically, this overhead is written off because I/O latency is so high on standard disks. SCMs, however, have significantly faster I/O, so this overhead presents a problem. A memory-based file system will also not suffice for SCMs because memory-based file systems assume that the devices are using a page cache that will be lost on restore.

Their goal is to keep the file system as small and simple as possible to keep overhead down. They propose to do this by utilizing the memory management module in the operating system, removing the block management in traditional file systems, and simplifying the file system design by allocating space contiguously per file. Instead of building over blocks, they build over the memory manager. Their code is 2700+ lines vs. 27000+ lines for ext3, and they have a small overhead compared to memory-based file systems.

Repairing Erasure Codes

Dimitris S. Papailiopoulos and Alexandros G. Dimakis, University of Southern California

The speaker presented a novel application of a new theoretical result in information theory to better rebuild erasure-coded systems. The speaker briefly introduced erasure coding, essentially a method of calculating parity over k data blocks such that any k blocks of your data+parity set can be used to restore the original data. While any MDS code can be used, Reed-Solomon is common.

Assume all n blocks of a particular piece of data are on different nodes. On rebuild, k nodes need to be spun up to reconstruct the original data. The data from these k blocks needs to be sent to a centralized location to calculate the value for the lost disk. However, if the code used is regenerating, the speaker claimed that the rebuild could occur with (n-1)*B/(n-k)*K blocks using a matroid property. For parity blocks, they read more than they send and calculate intermediate results.

Rewriting the storage stack for Big Data

A. Byde, G. Milos, T. Moreton, A. Twigg, T. Wilkie, and J. Wilkes, Acunu

This talk was presented by the Acunu company, and it describes their motivation for rewriting the middle of the storage stack to better suit modern applications and storage hardware. The speaker began by showing an old storage stack, and then pointed out that it has not changed much in many years. While there has been innovation at both ends, representing applications and hardware, there has been little work to redefine the middle.

Applications want new stuff, but they need to work around all the assumptions in the stack because the abstraction given to them is not immediately compatible with their goals. They termed this an impedance mismatch: the abstraction is fundamentally wrong. As a result, some applications built over this outdated stack are slow, unstable, and unpredictable.

Acunu proposes rethinking the stack by throwing away the file system interface and replacing it with a shared memory ring buffer in the stack. This allows a range of improve-

ments such as extremely fast versioning, control of the page cache, cache-oblivious algorithms, the ability to rewrite RAID, etc. Fast versioning means no B-trees, since they are slow on disk. Performance is linear at eight cores, and it is maintained for billions of inserts. They currently have a product that sits above the hardware and provides a memory monitoring stack. They will issue patches for Cassandra and Voldemort so they can run over their existing stack.

Acunu is hiring (http://www.acunu.com/jobs). They will also be in beta soon.

Second set of WiPs summarizerd by Vijay Vasudevan (vrv+usenix@cs.cmu.edu)

T2M: Converting I/O Traces to Workload Models

Vasily Tarasov, Koundinya Santhosh Kumar, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College

When evaluating or benchmarking systems, researchers today commonly generate a workload and monitor output parameters. But benchmarks themselves have a complex set of parameters and languages to express the workload; these benchmarks are becoming more expressive over time. Alternatively, people use I/O traces from real-world workloads, which are typically more credible. Many researchers release these real-world workloads and datasets, and as a result these trace repositories are growing larger.

The key idea of this work, presented by Vasily, is to convert traces to benchmark workload descriptions. Their system takes in traces such as system call traces, file system traces, etc., and generates an intermediate workload model, which hooks into adapters for different benchmark frameworks. These adapters take in the model and write the proper parameters for each output benchmark tool. For example, they have currently used system call traces and hooked it into Filebench.

Skyline-enabled Storage System

H. Howie Huang and Nan Zhang, George Washington University

This talk presented S3, the Skyline-enabled Storage System, designed to automatically manage heterogeneous storage systems by moving files around among devices. Devices have different characteristics (capacity, bandwidth, IOPS), and choosing the right set of files to move can be difficult. When a new large disk is added to a system, one would like to move large files. This request can be expressed as a top-k SQL-like query. If you add flash memory, you have a top-k query with a different scoring function based on the characteristics of the device. The scoring function therefore varies by device.

To efficiently execute such top-k queries, they proposed maintaining skylines. Without knowing the scoring functions, which are based on device characteristics, you can get the answer from the skyline data. Their work-in-progress includes how to identify dimensions of interest, constructing the optimal skylines, and using approximate processing to improve efficiency.

Load-Aware Replay of I/O Traces

Sankaran Sivathanu, Georgia Institute of Technology; Jinpyo Kim and Devaki Kulkarni, VMware Inc.; Ling Liu, Georgia Institute of Technology

Jinpyo began by asking, "Why do we need a load-aware trace replayer?" When traces are replayed on different systems, the characteristics of the system can show a different I/O profile, making performance debugging and analysis more difficult. Jinpyo provided an example of a database program replayed on another system, showing two very different behaviors.

They proposed using a load-based metric called "outstanding I/O requests" (OIO) to evaluate I/O load-balancing algorithms on large-scale cluster systems. Using load-based replay of original traces from a hypervisor and replaying it on a guest OS showed similar, reproducible behaviors across different systems. Their future work includes generalizing to distributed environments and improving replay accuracy.

Flash SSD Oriented IO Management for Data Intensive Applications

Yongkun Wang, Kazuo Goda, Miyuki Nakano, and Masaru Kitsuregawa, The University of Tokyo

Kazuo talked about optimizing flash I/O. One major direction for flash has been to develop log-structured or copy-on-write techniques to avoid random writes. But few have explored whether these techniques achieve the potential performance of the device. The goal of this work is to look at how to optimize I/O to achieve the device's full potential. They identified the performance gap from random writes vs. sequential writes using DB benchmarks, and they found that LFS gives improvement but does not reach the potential of pure sequential writes.

They proposed "eager scheduling" using database checkpoint cues, deferring write requests, and performing write scheduling techniques such as coalescing, LFS-like address conversion, and block alignment, all within a small buffer until the checkpoint. They demonstrated a trace driven experiment showing that even a small buffer improved performance. Simply deferring the writes without the aforementioned write scheduling techniques wasn't as helpful. Their ongoing work is focusing on how more information from the

application and device specifications can improve performance.

A Novel Nested Qos Model for Efficient Resource Usage in Storage Servers

Hui Wang and Peter Varman, Rice University

Server consolidation, bursty workloads, and single-level QoS make performance guarantees hard to achieve. Capacity provisioning provided by burst rate or long-term average are both insufficient: one has low server utilization and the other provides no strong guarantees.

In response, Hui proposed a nested QoS model. Each class is characterized by a traffic envelope and response time limit. If a request falls within that class, it is provided a guarantee of that time limit. The initial results showed that a nested QoS system could reduce the provisioned capacity significantly in terms of IOPS compared to a single-level QoS model, and could provide comparable performance.

Reading from a Write Optimized Parallel File System Under Highly Concurrent Shared File Workloads

Adam Manzanares, Los Alamos National Laboratory; Milo Polte, Carnegie Mellon University; Meghan Wingate and John Bent, Los Alamos National Laboratory; Garth Gibson, Carnegie Mellon University

Adam explained that HPC apps have thousands of clients writing to a single file (an N-to-1 writing pattern), which performs poorly on parallel file systems for a number of reasons. In response, they built an interposition layer atop a parallel file system that converts N-to-1 writes to N-to-N. Doing this improved write performance significantly: an example graph showed a factor-of-50 improvement. But this made subsequent read open times much worse. On opening a file read-only, all processes individually try to reconstruct a file offset map, which significantly degrades performance because each of N clients needs to aggregate N index files, which requires N^2 work.

They explored three different possible solutions. First, they developed an aggregation technique that uses a root process to perform the reconstruction exactly once, then distributing the resulting map to all the processes. While this improves performance over the naive approach, it unnecessarily serializes the work onto one process. This work can be parallelized and distributed over many nodes, which they called the "parallel read" approach. Finally, they noted an opportunity to write the map out to the file system on file close because the data would likely be in memory already, which they called the flatten-on-close approach. In their evaluation, the flatten-on-close was actually the best for read opens, but not great for writes, so the best option for a write-optimized file system was the parallel-read approach.

Their future work focused on several research opportunities: first, a parallel file system that uses decoupled log-structured files presents a unique opportunity to reconsider the downsides of log structured file systems; second, they are investigating a scalable key-value store to hold the file offset map, a feature they think will become important in the exascale era.

OrangeFS: Advancing PVFS

Michael Moore, David Bonnie, Walt Ligon, Nicholas Mills, and Shuangyang Yang, Clemson University; Becky Ligon, Mike Marshall, Elaine Quarles, Sam Sampson, and Boyd Wilson, Omnibond Systems

Michael talked about OrangeFS (a continuation of PVFS), a parallel open-source file system aimed at obtaining high performance. Michael described several goals for OrangeFS. First, they want to move file systems to fault-acceptance rather than just fault tolerance. As systems scale, failures increase. The systems that use parallel file systems exist in environments that require high availability, need replication, etc.

Support for distributed directories: applications often put lots of files in a single directory, a challenging requirement for most parallel file systems. They are changing the granularity of their directory implementation and are using techniques from GIGA+, presented the day before at FAST '11.

Capabilities: assumptions about trust in HPC environments have changed, so security must be improved. Their system uses a capability-based system to verify trust before performing operations.

OrangeFS clients: they are developing for Windows and have a working implementation.

Redundancy: they are duplicating objects between servers, assigning different roles for metadata and data, and working on integrating hierarchical storage, replication for performance, and off-site replication once this is done.

Flash the Second

Summarized by Rosie Wacha (rwacha@gmail.com)

Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance

Yangyang Pan, Guiqiang Dong, and Tong Zhang, Rensselaer Polytechnic Institute, USA

The reliability of NAND flash memory cells degrades over time and data becomes increasingly noisy with bad data. In

order to accommodate noisy media, manufacturers incorporate additional NAND flash memory cells to store error correction codes (ECC). With each program/erase cycle, charge traps are accumulated in the cells, resulting in higher error rates. Especially at the beginning of a device's lifetime, the cells storing ECCs are largely underutilized. Tong Zhang presented a mathematical channel model that treats data retention as a Gaussian process with mean and variance scaling with P/E cycling and time in a power law fashion. Using this model, he introduced two techniques that better utilize the additional flash memory cells.

The first technique improves program speed by adaptively adjusting the voltage range, dV, for each bit. In the early stages, a larger dV is used to speed up the program and introduce some additional tolerable errors. The other technique improves technology scalability by allocating some redundant cells to compensate for defective ones. Because the defect rate in different blocks will be different, individual blocks will have unique P/E limits and wear leveling algorithms will handle this issue. Trace-based simulations were done using the SSD module in DiskSim. The program speed was increased by a factor of 1.5 by varying dV throughout the device lifetime, without violating ECC redundancy. For technology scalability, 30% higher endurance was shown with differential wear leveling compared to uniform wear leveling.

Brent Welch from Panasas asked if they could simulate the reduced lifetime of the device due to writing with higher voltages. Zhang said yes, it's a mathematical model and he could modify the parameters that are based on underlying physics. Another question was whether using redundant cells earlier and with higher voltage than necessary causes additional wear on cells that you would later need for redundancy. Zhang answered that all cells are erased in every P/E cycle anyway and because the erase cycle is a stronger contributor to wear-out, the impact on the overall cycle limit is small.

FAST: Quick Application Launch on Solid-State Drives

Yongsoo Joo, Ewha Womans University; Junhee Ryu, Seoul National University; Sangsoo Park, Ewha Womans University; Kang G. Shin, Ewha Womans University and University of Michigan

Yongsoo Joo presented an application prefetcher for solid state drives (SSDs) called FAST (Fast Application STarter). The amount of time it takes for an application to become responsive upon launch is a critical indicator of user satisfaction. Hard disk drive (HDD) access latency has scaled linearly while CPU, DRAM throughput, and HDD have scaled exponentially. Several software schemes have addressed this issue, including Windows sorted prefetch, which prefetches the set of blocks historically accessed during the launch of each application. They measured a 40% improvement in application launch time using sorted prefetch. This work eliminates seek delay by moving applications to SSDs, resulting in some improvement in application launch time. However, traditional HDD optimizers based on minimizing disk head movement are not very effective on SSDs. Sorted prefetch on SSDs results in a 7% improvement in application launch time.

FAST overlaps CPU computation with SSD accesses by initiating application prefetching first and starting CPU computation when data is available. This is possible because there is determinism in the set of blocks requested by most application launches. FAST is implemented in Linux and it first uses the blktrace tool to determine the blocks necessary to the application, then replays block requests according to the observed launch sequence to generate the prefetcher, and finally executes that prefetcher simultaneously with the original application by wrapping the exec() system call. The challenge in prefetching the LBA from the inode information given by blktrace was solved by building a partial inode to LBA map for each application using a system call log. They measured a 16–46% reduction (average: 28%) in Linux application launch time, with higher reduction found for applications with a higher ratio of SSD to CPU usage and better distribution of SSD access over the launch process. This work could be ported to smartphones, with expected improvement of up to 37% based on measured cold and warm start times of 14 iPhone applications.

Assar Westerlund (Permabit) asked why not use strace to capture all the file-level I/Os such as read() system calls and replay them instead of converting the block-level I/Os from the blktrace into the file-level I/Os. Joo answered that, although it is possible, capturing all the read system calls generates an extremely huge trace size even though very few of the read() calls will actually trigger the block requests. Therefore, it is not an efficient approach in terms of the function call overhead and the prefetcher size. Ethan Miller from UCSC asked about how the cold start process was done and whether they used the application to do any work. If you use an application, it will involve a different set of blocks than if you simply open it. Joo answered that they flushed the page cache and only prefetched the blocks accessed during the measured cold start. This limits the approach in that it associates a fixed set of blocks for launch with each application.

Cost Effective Storage using Extent Based Dynamic Tiering

Jorge Guerra, Florida International University; Himabindu Pucha, Joseph Glider, and Wendy Belluomini, IBM Research Almaden; Raju Rangaswami, Florida International University

This work investigates the area of tiered storage systems using SSDs and shows that SATA and SAS drives still hold a useful place in the hierarchy. The two questions are how to choose the most cost-effective set of devices and how to best use those devices. Jorge Guerra discussed several initial approaches that motivated the design of the final data placement algorithm, called Extent-based Dynamic Tiering (EDT). EDT monitors the workload for each data extent and migrates extents in 30-minute epochs. The goal of EDT is to reduce both the initial system cost and the operational cost. SATA drives are best used for idle data, SAS drives for sequentially accessed data, and SSDs for randomly accessed data. The total utilization for a given extent on each tier is computed using the time utilization, a measure of both IOPS and access sequentiality, and the capacity utilization, a measure of the fraction of space used to store that extent. By choosing the lowest-cost tier for each extent, the tiered system response time is 43% better than an all-SAS system while using about half the power.

Another issue is how to initially configure a tiered system using this approach to minimize costs and maximize performance. Guerra discussed a configuration advisor based on four steps. The approach is to use a representative I/O trace, divide the trace into fixed-length epochs, estimate the minimum cost per epoch, and, finally, estimate the overall set of resources needed across all epochs. For each epoch, the minimum-cost hardware set is computed based on all extents accessed in that epoch. Then looking at all epochs, the maximum number of resources of each tier-type is needed for the overall configuration.

The system is implemented in Linux and evaluated with SSDs, SAS, and SATA drives and a power meter. The paper contains results for SPC1-like benchmarks as well as several Microsoft traces, but Guerra presented only the server workload results. EDT reduced the capital needed to build the system by 36% from an all-SAS solution. More importantly, the response time is 60% better than SAS and 50% better than their initial approach. Because EDT migrates data to SSD when it's most frequently accessed, power is reduced by 57%.

Fred Douglis from EMC clarified that EDT is not strictly better than the simpler IOPS-DT approach in all workloads, then asked about data migration and whether EDT prefers to not move data. Guerra answered that yes, at every epoch there are several optimizations to minimize data movement. Ajay Gulati from VMware asked if they evaluated configurations using only SSD and SATA. Guerra answered that some of the traces have highly sequential access periods and require SAS to match necessary throughput. Westerlund asked about factoring the operational cost into the cost model. Guerra showed that power is significantly reduced but hasn't yet incorporated that into the total system cost model.

Thursday Poster Session

First set of posters summarized by Shivaram Venkataraman (venkata4@illinois.edu)

A File System for Storage Class Memory

Xiaojian Wu and A.L. Narasimha Reddy, Texas A&M University

Xiaojian Wu presented a poster on designing a new file system for non-volatile memory devices. Based on the virtual memory subsystem, the poster proposed a new layout for files and directories to avoid overheads associated with repeatedly accessing page tables. When asked how this differs from existing in-memory file systems like tmpfs, Xiaojian said that existing file systems are not optimal for Storage Class Memory and that the newly proposed file system provides better performance and metadata consistency.

RAMCloud: Scalable Storage System in Memory

Nanda Kumar Jayakumar, Diego Ongaro, Stephen Rumble, Ryan Stutsman, John Ousterhout, and Mendel Rosenblum, Stanford University

Nanda Kumar Jayakumar presented RAMCloud, a cluster-wide in-memory storage system. The storage system is designed for applications which require low latency accesses and was based on a log-structured design. Updates to the system are appended to an in-memory log and these updates are then asynchronously flushed to disk to ensure reliability. Nanda explained that the design was also optimized for high throughput and acknowledged that while the cost of building such a system might be high, there were many real-world applications for which this would be affordable.

Rewriting the Storage Stack for Big Data

A. Byde, G. Milos, T. Moreton, A. Twigg, T. Wilkie, and J. Wilkes, Acunu

Andy Twigg presented a poster on changing the storage stack in operating systems to enable high performance storage systems. Andy explained that distributed storage systems like Cassandra and HBase were built on top of legacy designs like file system buffer caches. Instead, the group proposed using a new, fully persistent, versioned B-tree which would provide fast updates and also be optimized for SSDs. When asked if some of their changes would conflict with the design of Cassandra, Andy acknowledged that they had made some

modifications to Cassandra and would be releasing patches for that.

DiskReduce: RAIDing the Cloud

Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson, Carnegie Mellon University

DiskReduce, presented by Wittawat Tantisiriroj, explored the possibility of integrating RAID with distributed file systems like HDFS. With traces from Facebook, Yahoo, and a research cluster at Carnegie Mellon University, the authors found that in most clusters, around 80% of the files were smaller than 64 MB, the block size used in HDFS. Hence they implemented RAID encoding algorithms for all the files in a particular directory and found that there was little performance degradation and no significant change in the data loss rate.

OrangeFS: Advancing PVFS

Michael Moore, David Bonnie, Walt Ligon, Nicholas Mills, and Shuangyang Yang, Clemson University; Becky Ligon, Mike Marshall, Elaine Quarles, Sam Sampson, and Boyd Wilson, Omnibond Systems

Michael Moore presented the poster on OrangeFS, which is an effort to improve PVFS, a parallel file system deployed in clusters. When asked about the similarities between metadata layout in OrangeFS and the paper on GIGA+ presented at the conference, Michael clarified that the implementation in OrangeFS was a simplified version of what was discussed in the paper. He also said that they were implementing OrangeFS to be "failure accepting," which meant that the file system was robust against events such as disk failures.

New Cache Writeback Algorithms for Servers and Storage

Sorin Faibish, Peter Bixby, John Forecast, Philippe Armangau, and Sitaram Pawar, EMC

The cache writeback algorithms used in the Linux kernel were the subject of the poster presented by Sorin Faibish. Based on multiple workloads, the poster presented evidence on how the existing cache writeback algorithms were detrimental to the performance and reliability of the system. The poster also proposed many new cache writeback algorithms; when asked which of those performed better, Sorin explained that the effort was meant to raise awareness about the problems associated with swapping and that the final solution could be any one of them.

dBug: Systematic Testing of Distributed and Multi-Threaded Systems

Jiri Simsa, Garth Gibson, and Randy Bryant, Carnegie Mellon University

Jiri Simsa presented the poster on dBug, a testing framework for distributed and multi-threaded systems. The poster described how dBug worked by intercepting system calls made by the program and had been used to find bugs in Stasis, a transactional storage system. Jiri also pointed out that these were not limited to multi-threaded programs and that dBug could also be used to find any implementation errors in protocols used in RPC-based message-passing systems.

Second set of posters summarized by Yuehai Xu (yhxu@wayne.edu)

PreFail: Programmable and Efficient Failure Testing Framework

Pallavi Joshi, Haryadi S. Gunawi, and Koushik Sen, University of California, Berkeley

Pallavi Joshi introduced a testing framework (PreFail) to address the challenges on how to systematically explore failures of large-scale distributed systems. Through PreFail, testers can easily express failure exploration polices of various complexities. Evaluation showed that the framework can produce improvements up to a factor of 21, depending on workload and failure type.

Repairing Erasure Codes

Dimitris S. Papailiopoulos and Alexandros G. Dimakis, University of Southern California

Alex Dimakis surveyed recent developments in the information theory community on designing new erasure codes that have efficient rebuild properties. He gave us an example about how to set the extra repair traffic to theoretic minimum cut-set bound when the coding was using maximum distance separable (MDS) erasure codes such as Reed-Solomon codes.

T2M: Converting I/O Traces to Workload Models

Vasily Tarasov, Koundinya Santhosh Kumar, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College

Traces are cumbersome to replay and do not scale well as a system becomes more complicated. Vasily Tarasov said their objectives are to investigate the possibility of automatically creating workload models from existing traces and to produce scalable and easy-to-use models while maintaining realism of the traces. Currently, three main directions are considered: the set of parameters extracted from the traces, the level at which the traces need to be generated, and the language used for expressing the traces.

Skyline-enabled Storage System

H. Howie Huang and Nan Zhang, George Washington University

Nan Zhang presented the idea of enabling efficient and automated management over a large-scale file system, in which the key prerequisite is the ability to process top-k queries in an efficient manner. They borrow the idea of the skyline operator from the database community and extend it to automated management of large file and storage systems. Their approach is to maintain a list of skyline files in the system that can support efficient processing of all top-k queries, regardless of what the scoring function might be.

Load-Aware Replay of I/O Traces

Sankaran Sivathanu, Georgia Institute of Technology; Jinpyo Kim and Devaki Kulkarni, VMware Inc.; Ling Liu, Georgia Institute of Technology

Jinpyo Kim introduced an I/O load-aware replay mechanism, so that the load profile of the original application/system can be matched with that of the replay system, even when the environments of these two systems are different. Number of pending requests is the key metric used for making the load-aware replay keep the trace performance behaviors. Evaluation showed that the objective is achieved pretty well.

A Novel Nested Qos Model for Efficient Resource Usage in Storage Servers

Hui Wang and Peter Varman, Rice University

Hui Wang proposed a nested QoS service model to provide flexible QoS performance guarantees to clients. This model places requests into different classes according to a user's SLO (service level objective) in terms of response time. A class tag is then set for each request to guide its scheduling. Evaluation showed that resources required for implementing the nested QoS model is several times smaller than that for a single-level QoS, while the service quality experienced by the clients is minimally degraded.

Reading from a Write Optimized Parallel File System Under Highly Concurrent Shared File Workloads

Adam Manzanares, Los Alamos National Laboratory; Milo Polte, Carnegie Mellon University; Meghan WingazMellon University

Adam Manzanares improved the read bandwidth of the Parallel Log-Structured File System (PLFS) by introducing several index aggregation optimizations. PLFS is a write-optimized file system that was designed for shared file workloads. PLFS transforms a logical shared file into independent log-structured components. Planned future work includes a thorough investigation into the read performance of PLFS using representative scientific workloads and the development of a scalable shared indexing mechanism for PLFS.

Analysis of Workload Behavior in Scientific and Historical Long-Term Data Repositories

Ian F. Adams, University of California, Santa Cruz; Mark W. Storer, NetApp; Ethan L. Miller, University of California, Santa Cruz

In this work, Ian F. Adams presented results of their recent study on the archival system's characteristics through analyzing access behavior data. The results are useful to guide the design of future archival systems. They discovered that storage systems appear to be becoming more disk-centric and the assumption of write-once read-maybe doesn't hold universally. Additionally, they discovered that it is extremely challenging to acquire useful datasets. To this end, data-centric tools for long-term tracing are needed.