2nd Workshop on Hot Topics in Storage and File Systems (HotStorage '10)

June 22, 2010
Boston, MA

DON'T WORRY, YOUR DATA IS SAFE WITH FLASH

Summarized by Rik Farrow (rik@usenix.org)

- ***Removing the Costs of Indirection in Flash-based SSDs with Nameless Writes***
  Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison; Vijayan Prabhakaran, Microsoft Research

Remzi Arpaci-Dusseau began with a quote attributed to Butler Lampson: "All problems can be solved . . . by another level of indirection." He went on to list the many uses in operating systems of redirection, such as virtual memory, RAID, and VMMs. But indirection introduces performance issues as a side effect. Arpaci-Dusseau said that the target of this research is the Flash Translation Layer (FTL), an abuser of indirection.

FTLs use indirection because writing to flash requires writing only to erased pages, and erasing a page takes milliseconds, not microseconds. FTL hides this latency by writing to a log. Arpaci-Dusseau presented the authors' main idea: nameless writes. Instead of attempting to write to a particular block on a flash device, the data is written to the device. On completion of the write, the device returns the physical location of the block written. Someone asked about handling wear-leveling, and Arpaci-Dusseau responded that the device would upcall into the client, updating the physical location. Randal Burns vehemently disagreed, saying that causes all sorts of problems, and Arpaci-Dusseau agreed with Burns. But then he said he still thought this is a good idea. He then pointed out that every write cannot be nameless, as there must be some known beginning address. The device must also be willing to share some low-level information.

Chris Small pointed out that all performance problems can be solved by removing indirection as a corollary to the opening quote. Then Small worried that making flash too dumb might cause problems. Arpaci-Dusseau agreed that there must be some information stored within the flash, for example, for wear-leveling. Peter Desnoyers didn't consider wear-leveling the big issue, but instead thought that garbage collection was more of a problem. Arpaci-Dusseau said he worried about this too, but didn't have a solution for this yet. Someone suggested adding new interfaces that handle resource allocation.

- ***Depletable Storage Systems***
  Vijayan Prabhakaran, Mahesh Balakrishnan, John D. Davis, and Ted Wobber, Microsoft Research Silicon Valley

Vijayan Prabhakaran pointed out that, traditionally, space is the major constraint in storage, but in SSDs, the primary issue is the number of erasures. Ideally, the lifetime of a device would be the product of the size of the device times the number of erase cycles, but wear-leveling reduces this in practice. When using flash, write patterns also influence wear: for example, sequential block-sized writes compared to small random writes.

In response, Prabhakaran suggested that we need depletion-aware resource management. This would allow predictable replacement based on the lifetime of a device, a way to charge users for usage and to compare designs that reduce depletion, and to deal with new attacks against devices that reduce lifetime. Prabhakaran listed two challenges: many layers in file systems, such as caching, journaling, schedulers, and RAID; and media heterogeneity, such as SLC vs. MLC with different performance and erasure limits. Their solution is to introduce a VM that isolates applications from the device, minimizing the layers before issuing writes to an SSD.

Dan Peek from Facebook wondered if doing this would hide important details from the application writing that need to be exposed. Prabhakaran agreed that this was the right way to do things, but wondered what metrics should be exposed to applications. Peter Desnoyers continued on this theme, using Intel's high-end SSD, which has 80GB of flash but only exposes 64GB, as an example. Desnoyers wondered how much information needs to be exposed. Prabhakaran said that as a community, we need to provide a set of techniques to expose this data. Someone else asked if they had disabled caching, and Prabhakaran said that they had tried their experiments both ways, with caching enabled and disabled.

- ***How I Learned to Stop Worrying and Love Flash Endurance***
  Vidyabhushan Mohan, Taniya Siddiqua, Sudhanva Gurumurthi, and Mircea R. Stan, University of Virginia

Vidyabhushan Mohan explained how stress events affect both the retention and endurance of flash. Flash memory can only be written to after being erased, so a stress cycle consists of write (program)/erase cycles (P/E). Most research on flash use utilizes manufacturer datasheets to calculate endurance, but recent papers on NAND flash chip measurements hint at a much higher endurance. An important but overlooked factor in flash endurance is a recovery process which occurs during the time between stress events and allows partial healing of a memory cell.

The authors designed a simulation that takes the time for recovery into account, while modeling the device physics and applying write traces from four different server applications: EXCH, LM, RADIUS, MSNFS. Using these traces, they could calculate the amount of recovery time between stress events and use this to calculate the number of P/E cycles under each workload. What they found is that endurance could be increased by two orders of magnitude with recovery times on the order of a couple of hours, and this occurred with all their example workloads. Their conclu-

sion was that SSDs are durable enough to support enterprise workloads, although this should be examined using real enterprise workloads.

Remzi Arpaci-Dusseau said he had hundreds of questions, but asked just one: is there a measurable difference with bandwidth and performance? Mohan responded that you can see better performance with newer (less stressed) memory. Arpaci-Dusseau then asked, "Add capacity in the SSD to spread out the load more?" Mohan said yes, this improves both performance and endurance. Someone asked if they had talked to vendors about this, and Mohan said they had. He then asked if endurance also depends on implementation of the hardware, and Mohan said yes. The same person asked about the vendor endurance numbers, and Mohan said they are worst-case estimates, created by testing SSDs in ovens. Peter Desnoyers said that this is exciting work, and he wondered if adding error detection could extend SSD life even further. Mohan said that bit error rate does increase after a few million cycles.

OUT WITH OLD (RAID)

Summarized by Aleatha Parker-Wood (aleatha@soe.ucsc.edu)

- ***Block-level RAID Is Dead***
  Raja Appuswamy, David C. van Moolenbroek, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Raja Appuswamy presented a modular file system stack called LORIS, which inverts the conventional file system/RAID stack, moving RAID-like file multiplexing into the logical layer, instead of the block layer. This allows the physical layer to implement parental checksumming on *all* blocks, rather than RAID being allowed to propagate corruption into the parity blocks.

LORIS divides the software system into a fully modular stack. At the physical layer, metadata caching, checksums, and on-disk layout are handled. Above that is a logical layer, which handles RAID and logical policy storage, and has a mapping file which maintains all of the policy information, such as the RAID level, the stripe size, and the file identifier. The caching layer is responsible for caching data, as usual. Finally, the naming layer handles POSIX call processing and manages directories.

LORIS uses a unique ID and a set of attributes for each file, which are fully shared between all layers. Any layer can get or set an attribute, and files are referred to by their common ID. Because of this shared infrastructure, any layer can set policy information. Because RAID is in the logical layer, it can be file aware, rather than block aware, and therefore can implement these policies in an intelligent fashion. LORIS offers a clean stack abstraction which allows more intelligent error handling for RAID, and allows other components of the stack to be swapped out at will, opening up new possibilities for filesystem designers.

Chris Small from NetApp asked how this was different from what NetApp currently does. Appuswamy replied that LORIS has the ability to isolate changes in the file system to a single layer, rather than being tightly integrated throughout the stack. Remzi Arpaci-Dusseau asked why they were stopping before the hardware layer, noting that there's a lot of abstraction that goes on in the hardware level and they're moving away from the common interface. He asked what kind of interfaces they would like to see at the lower layer. Appuswamy replied that because LORIS is a pure stack, any number of the layers could be moved into the hardware. For instance, hardware could move to an object-based interface without disrupting the stack.

- ***Mean Time to Meaningless: MTTDL, Markov Models, and Storage System Reliability***
  Kevin M. Greenan, ParaScale, Inc.; James S. Plank, University of Tennessee; Jay J. Wylie, HP Labs

Kevin Greenan presented a new reliability metric, called NOrmalized Magnitude of Data Loss (NoMDL). Greenan argued that mean time to data loss (MTTDL) is a metric which is meaningless and misleading. In the authors' opinion, a good reliability metric should be calculable, meaningful, understandable, and comparable. In other words, it should be generable using a known and understood method (such as closed form equations, or simulation), relate to real world systems, and be possible for system owners to understand and use to compare systems.

MTTDL is easy to calculate, since it relies on Markov models, which can be calculated in closed form. However, Greenan noted some flaws in the meaningfulness of the model. For instance, since Markov models are memoryless, the model completely ignores hardware aging. Every time a hard drive is replaced, the model assumes that all remaining hardware is in perfect condition. This does not accurately reflect reality. Likewise, MTTDL often is applied in a sector-failure-agnostic way. Even if sector failures are accounted for, Markov models do not describe the "critical mode" of a system, where additional sector failures during rebuild may cause data loss, depending on their location. Since the probability of data loss declines continually over the rebuild period, it is challenging for a Markov model to describe. Finally, MTTDL is a metric which only answers the question, "When will I lose data?" and not "How much data will I lose?" The authors argue that the latter is a more useful and important question to answer.

Greenan proposed NOrmalized Magnitude of Data Loss (NoMDL) as a replacement for MTTDL. NoMDL avoids some of the flaws of MTTDL and aims to answer the question "How much data will I lose?" by relying on Monte Carlo simulation, a popular statistical technique. The authors have built a framework for modeling drive failure and made it available for other researchers to use. It uses a "mission time" (such as the 15-year expected lifespan of a

system) and a number of simulation iterations to return an expected amount of data lost at the end of the time span. Comparing it to other metrics, Greenan noted that their system is the only one which is system-agnostic and provides a magnitude of failure.

Randal Burns from Johns Hopkins noted that an answer such as 14 bits is still not meaningful, because systems lose data in large chunks or not at all. Greenan replied that the simulator can actually return a histogram of data loss. Randal retorted that the final output was still a metric. Jim from EMC said that a lot of people associate MTTDL with lifetime, which is clearly inaccurate. Michael Condict from NetApp noted that Greenan was arguing against MTTDL for a single device, and he asked why the regenerative model was bad for a whole system. He suggested that they just change the model to say that the device is halfway through the lifespan. Greenan replied that that still wouldn't be accurate, because the system as a whole is aging, not just an individual device. Empirical testing suggested that just artificially aging the device in the model yielded unrealistic results.

- ***Discussion Panel***

The session chair, Arkady Kanevsky, kicked off the discussion panel by asking whether RAID was even relevant any more, given that failures are now known to be highly correlated. Greenan replied that block-level RAID is dead because of rebuild time. The window of vulnerability is getting bigger and bigger. Distributed RAID will make more sense, because the system can spread the load out. Appuswamy replied that by imparting semantic knowledge to the RAID layer, many things become possible. Search-friendly name schemes require a full rethinking of RAID, which requires an abstraction layer.

Ric Wheeler, addressing Greenan, noted that soft errors in hard drives are found via pro-active scanning, which offsets second drive failure problems. Greenan asked whether a higher-level process which was checking the errors would have the information to fix the errors. Wheeler replied that the information was available at the block level, since things like trim commands have a notion of which blocks are alive.

Jiri Schindler asked Greenan to give an argument that his model was a more general one than the one used by Elerath. Greenan replied that Elerath's model was specific to Weibull distributions, where their package allows them to plug in distributions. He also noted that Elerath was focused very specifically on a RAID 4 array, where theirs can take an arbitrary erasure code.

Michael Condict from NetApp noted that one of the benefits of MTTDL was the ease of calculation and asked Greenan what the inputs to his model were and whether his model was easy to calculate. Greenan said that the model was available already and that MTTDL didn't allow anything except a Markov model. Condict then asked whether he could input more intelligent data into the model if it was available. Greenan noted that it was possible to apply a Markov model in that way, but that it was very difficult and somewhat inaccurate. They chose simulation because it was more accurate and required less work for the systems engineer versus creating a very complex Markov model.

SCALING UP, VIRTUALLY

Summarized by Aleatha Parker-Wood (aleatha@soe.ucsc.edu)

- ***KVZone and the Search for a Write-Optimized Key-Value Store***

Salil Gokhale, Nitin Agrawal, Sean Noonan, and Cristian Ungureanu, NEC Laboratories America

Nitin Agrawal presented Alphard, a write-optimized local key-value store, and KVZone, a benchmarking tool for testing key-value stores. There are a variety of existing key-value stores, but benchmarking tools for key-value stores significantly lag behind development, and there has been no head-to-head comparison. The authors needed a low latency local key-value store to back their content-addressable file store, HydraStore, and therefore set out to benchmark existing key-value stores to find a suitable candidate.

KVZone is a benchmark specifically optimized for testing local key-value stores. It generates key-value pairs based on a specified set of properties. Key lifetimes and a mix of operation probabilities, such as a workload which is 60% reads, 20% writes, and 20% deletes, can be specified. The rate of requests can be specified in terms of either throughput or latency. Finally, KVZone can take an already existing key-value state to warm up the key-value store, in order to evaluate a particular real-world situation.

The results of their testing suggested that even the most performant of key-value stores operated at less than 40% of their raw device throughput. This was inadequate performance for HydraStore, which led them to create their own key-value store, Alphard. Alphard was specifically created with local, write-intensive workloads in mind and was optimized for SSDs. Some of the optimizations include direct I/O, block-aligned I/O, and request coalescing, as well as including metadata with key-value pairs to maximize the effectiveness of single writes. Alphard uses a logically centralized queue, with multiple physical queues and worker threads, in order to bring operations as close as possible to one synchronous I/O per key-value operation. Alphard achieves very close to device performance under their write-intensive workload.

Chris Small from NetApp noted that comparing a multithreaded KVS versus single-threaded KVS was not an apples-to-apples comparison. Agrawal replied that they were specifically focused on the properties of each key-value store under the required workload, rather than redesigning existing key-value stores. Ric Wheeler asked about the trade-offs

for durability and whether Alphard was durable. Agrawal replied that they did care about durability. Writes are persistent to the media, and mirrored. Mirroring requires a 3–5% overhead. Were there any insights about key-value stores in general that could be distilled from the authors' work, and why had they chosen an asynchronous interface? The asynchronous interface gave them the ability to coalesce operations into a single I/O. Also, since they were dealing with an asynchronous interface to the device, it preserved the semantics of the FS.

- ***Rethinking Deduplication Scalability***
  Petros Efstathopoulos and Fanglu Guo, Symantec Research Labs

Petros Efstathopoulos presented a highly scalable system for deduplication, designed to scale to one hundred billion objects, with high throughput. The authors were willing to sacrifice deduplication performance in order to achieve near-raw-disk performance.

The conventional approach to scaling deduplication performance has focused on using larger and larger segment sizes. However, this reduces the quality of deduplication and creates problems for reference management. The larger the segment size, the more catastrophic a deletion error or a lost reference is. In addition, the system still needs to be fast. The authors propose to use a sub-sampling technique, which they call progressive sampling.

The system creates a sample index, which is maintained in memory. The sampling rate is a function of the memory size, the size of each segment entry, and the total number of segments. When memory is plentiful, everything is indexed. However, as the system runs low on space, the sampling rate is progressively reduced. A purely random sampling strategy will result in decreased performance, so the system uses a fingerprint cache to take advantage of locality. In addition to the sample index, a full deduplication index is created and checkpointed to disk. Finally, the authors propose using SSDs for a fingerprint index, allowing memory to be used purely for caching and bloom filters which summarize the SSD index.

The final challenge in deduplication is reclaiming resources. Reference counting is simple, but challenging to make resilient in the face of failure. A reference list makes it possible to identify which files use which segments, but doesn't handle lost updates and is prohibitively expensive to scale up. Mark-and-sweep is another popular garbage collection technique, but this has a workload proportional to the capacity of the system, which is too slow at the petabyte scale. The authors propose a group mark-and-sweep, which improves the performance. The system tracks changes to a group and re-marks changed groups. If nothing has changed since the last iteration, mark results are saved and reused. This results in a workload which is a function of the work done since the last mark-and-sweep, rather than the size of the system.

Michael Condict from NetApp asked how the system decided which fingerprints to keep and which to discard to disk. Efstathopoulos replied that they just picked every *n*th segment. Condict suggested that they consider Extreme Binning as a complement to their work. Efstathopoulos noted that Bhagwat et al. were using Extreme Binning as a method for identifying super-segments. Condict noted that this method might improve the chances of the system having a hit. Dutch Meyer from the University of British Columbia asked how the system determined what constituted a group for their mark-and-sweep approach. Efstathopoulos replied that groups were composed of one or more backups of a system. Finally, Meyer asked if they had tried reference counting as a heuristic on their cache. Efstathopoulos replied that they had tried a variety of heuristics, and concluded that the overhead wasn't worth it.

- ***TrapperKeeper: The Case for Using Virtualization to Add Type Awareness to File Systems***
  Daniel Peek, Facebook; Jason Flinn, University of Michigan

Daniel Peek from Facebook presented TrapperKeeper, a method for extracting rich metadata from files without requiring file type creators to write plugins for every file system and search system. Rich metadata is the holy grail for designers of search systems. Unfortunately, extensions follow a long-tailed distribution, and it is uneconomical for either search systems or applications to support every file type in existence. Popular file types are well supported, but less popular file types are unlikely to be.

TrapperKeeper utilizes the already implemented behavior of applications to parse files, in order to capture metadata. It runs applications in a virtual machine environment. By opening a dummy file and then taking a snapshot at the moment of the open() call, the system can guarantee that the application is about to parse a file. When parsing behavior is needed, the VM can be restarted, and a real file can be substituted for the dummy file that was about to be invoked. From the application's point of view, this is seamless.

The next challenge is using the application to extract key-value pairs. However, most applications implement the accessibility APIs bundled with operating systems. By leveraging these and applying a variety of heuristics, the system can automatically detect tables, labels, and so on. Alternatively, the user can do manually guided extraction, which the system will cache for later use on other files of that type.

Someone raised a number of open questions about the system, which Peek noted were valid future work areas. For instance, what if the application has no accessibility support or does not expose metadata? What if the application needs external information, such as configuration files, in order to parse the input? One audience member suggested that a hybrid approach might be best, where plugins are used for the most common file types, but TrapperKeeper is used for the long tail.

- ***Discussion Panel***

The session chair, Ric Wheeler, started the discussion by asking each of the panelists how scalable they would like their systems. Peek replied that he worried both about system performance and human scalability. He wanted to avoid duplication of effort, such as multiple users creating parsing behavior for the same file types. Agrawal replied that he wanted to push the limits of Alphard and make it effective as a scalable store, as well. Efstathopoulos noted that scalability doesn't always rely on a new idea. The design principles are well known, but not always applied. Systems often aren't built with those in mind; sometimes the system designer has to go back and build it right later on.

Someone asked Efstathopoulos whether his system was processing directed acyclic graphs for garbage collection or was just a single level deep. Efstathopoulos replied that they had a flat space for garbage collection, where the storage group container has an ID and containers have chunks.

Someone asked Agrawal what the guarantees were that Alphard provided, from the time the client uses the system until the data is safely on disk. The questioner noted that coalescing writes just made matters worse and that there was an opportunity for something to go wrong while a request was in the queue. Agrawal replied that the actual interface didn't return until the data was safely committed, so while the system was asynchronous in implementation, the interface was, in fact, synchronous.

Another audience member asked whether the move to key-value stores would inhibit or help accessibility of rich metadata. Peek replied that right now there was no specialized file system handling for indexes, so a key-value store would have little impact. Efstathopoulos replied that there was a constant tug-of-war between specialized and general file systems. Agrawal added that system designers should think about what they actually want in a file system and design around it, rather than vacillating between extremes, as system designers realize they're missing key pieces each time they jump on a new technology.

Finally, someone asked about the differences between usage for key-value stores versus databases, noting that databases offer a many-to-many relationship, where key-value stores are strictly one-to-one, or one-to-many. Agrawal replied that he normally only used one or two keys, and that in practice data is often sharded across multiple databases, such that complex join operations, while possible in theory, are rarely used in practice.

ALL ABOARD HMS BEAGLE

No reports are available for this session.

- ***Fast and Cautious Evolution of Cloud Storage***
  Dutch T. Meyer and Mohammad Shamma, University of British Columbia; Jake Wires, Citrix, Inc.; Quan Zhang, Norman C. Hutchinson, and Andrew Warfield, University of British Columbia
- ***Adaptive Memory System over Ethernet***
  Jun Suzuki, Teruyuki Baba, Yoichi Hidaka, Junichi Higuchi, Nobuharu Kami, Satoshi Uchida, Masahiko Takahashi, Tomoyoshi Sugawara, and Takashi Yoshikawa, NEC Corporation
- ***Discussion Panel***