usenix ;login:
OCTOBER 2012 VOL. 37, NO. 5
ELECTRONIC SUPPLEMENT

Conference Reports

4th USENIX Workshop on Hot Topics in Storage and Files Systems (HotStorage '12)

Boston, MA
June 13-14, 2012

Dealing with Dynamics

Summarized by Timothy Zhu (timothyz@cs.cmu.edu)

Multi-Structured Redundancy

Eno Thereska, Phil Gosset, and Richard Harper, Microsoft Research, Cambridge, UK

Eno Thereska began by relating some common abstractions used in today's datastores. For example, key-value stores, file stores, and graph stores are all used to store data, but the different abstractions are often paired with certain assumptions and designs. When workloads change, we end up needing to redesign systems for a new set of assumptions. In his talk, Eno proposes the radical idea of using multiple datastores simultaneously to efficiently represent data. He relates this to the idea from the database community of representing tables both as row stores and column stores.

In particular for datacenters, Eno proposes using the redundancy in the system to store the multiple representations. This opens up many research questions such as: Can the different datastores be kept in sync? What about performance interference? Can this be supplanted with an in-memory structure? Eno also presented this idea from the perspective of a laptop environment. Here there is no redundancy to take advantage of, but the system can be partitioned to expose multiple data representations. This approach also brings along some interesting HCI questions about how people want to interact with their data.

The first question from the audience was whether this work has covered all the fundamental data structures. Eno responded that the key-value store, file store, and graph store are likely the most common, but additional structures may be needed. One point of caution when using more data structures simultaneously is the increased complexity in synchronization. Next, Timothy Zhu (CMU) asked about the impact of erasure coding replication in the datacenter. Eno responded that erasure codes are mostly used for cold data, and this idea might still apply to the hot data. He also remarked that perhaps the multiple data structures can be applied only to the indexes. In this case, it doesn't matter how the data is replicated. Timothy also asked if there was a tradeoff between ingesting data quickly and accessing the data quickly in an efficient format. Eno responded that their system always accesses the latest version, but it will eventually need to handle the backlog from converting the quickly ingested data into the appropriate datastore. Lastly, Nauman Rafique (Google) asked about how this research direction compares with trying to find one general datastore that works in all cases. Eno responded that in-memory stores work well in most cases, but some workloads need different data formats. He gave an example from the database community of how using both row stores and column stores make a big performance difference, both on-disk and in-memory.

MixApart: Decoupled Analytics for Shared Storage Systems

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

In this talk, Madalin Mihailescu looked at the problem of combining enterprise and analytics data in one datastore. Since these two workloads exhibit very different behaviors, different storage systems have been built for each type. For example, enterprise systems have a rich set of data management features and are designed for strong consistency. On the other hand, analytics systems are less concerned about consistency and data protection, but are designed for high throughput. The common practice of using two separate datastores unfortunately leads to excessive hardware and maintenance cost. Furthermore, sharing data often requires transferring data between the systems. What the authors propose is a method of integrating analytics data into enterprise storage systems while maintaining scalability and performance.

The key idea behind their technique is to use the local disk of the compute server as a cache for performance. Since there is high data reuse in many workloads, it is possible to reduce load at the shared storage system by caching data on the compute server's local disk. They also build a scheduler to better utilize the cache and prefetch input data for the next job. Madalin concluded with some evaluation results comparing their system to Hadoop.

Moises Goldszmidt (Microsoft Research) and Ning Wu (EMC) asked how this technique could be used to improve machine learning algorithms and Hadoop. Madalin responded by clarifying that this work is not attempting to improve performance, but rather to integrate analytics data with enterprise storage. Their techniques help to alleviate the burden from the analytics workload so that they can achieve a similar performance as when running on a dedicated datastore. Ajay Gulati (VMware) asked if large datacenters should be switching to use enterprise storage systems. Madalin replied by suggesting that companies need not use one large storage system instead of a cluster of storage devices, but could use something in the middle such as a small set of medium-sized storage servers.

LoadIQ: Learning to Identify Workload Phases from a Live Storage Trace

Pankaj Pipada, Achintya Kundu, K. Gopinath, and Chiranjib Bhattacharyya, Indian Institute of Science; Sai Susarla and P.C. Nagesh, NetApp

Pankaj Pipada presented the authors' investigation into the problem of identifying different I/O phases in a workload from a live trace. This information can be used to dynamically tune storage systems to better match the requirements of a workload phase. Doing so would require a tool that is automated and non-intrusive. They propose a generic technique, not based on heuristics, to automatically classify I/O phases based on storage traces and build a tool based on these ideas.

The key mechanism behind their technique is the Support Vector Machine (SVM) machine learning algorithm. They define a similarity metric between two traces by using histograms of shifts in the storage trace's offset field. This defines a similarity matrix. Using a technique from prior work, they modify this matrix to make it positive semidefinite. This is then used in a SVM classifier to classify a trace as one of two phases. To support m phases, they use m(m-1)/2 SVM classifiers, one for each pair of phases. A trace is tagged a particular phase if all m-1 of its classifiers vote for this phase. If a trace cannot be identified as one of the phases, then the trace is considered "unknown." After some time, the unknown category can be turned into its own phase.

Pankaj concluded with some results showing their classification accuracy. He also raised some questions for the audience. First, he asked about how concurrent I/O can be separated in a combined trace. Second, he asked about how to quantify the confidence of the result. Lastly, he asked about ways in which systems can exploit this phase knowledge.

Moises Goldszmidt (Microsoft Research) first commented that SVM already has a method of statistically quantifying the confidence. Second, he asked if this problem is more suited to a clustering algorithm rather than SVM, a binary classifier. He suggested perhaps using latent Dirichlet allocation, which handles unknown groups more effectively. This would also eliminate the need to modify the similarity matrix. Pankaj responded offline that defining the number of clusters is a problem when using a clustering-based approach. Also, he noted that using supervised learning makes it easier to interpret each cluster, which is important for the mentioned use cases. Eno Thereska (Microsoft Research) commented that using additional information about other parts of the system such as memory or network usage could provide better phase identification than from using storage traces alone. He asked if this technique could be applied to these other resources as well. Pankaj responded that their framework should work for other resources, but a similarity metric needs to be defined for that data. Lastly, Ajay Gulati (VMware) commented that phase identification can be useful in systems for performance diagnosis as well as throttling applications for QoS.

Cloudy with a Chance of QoS

Summarized by Pankaj Pipada (ppipada@gmail.com)

Gecko: A Contention-Oblivious Design for Cloud Storage

Ji Yong Shin, Cornell University; Mahesh Balakrishnan, Microsoft Research; Lakshmi Ganesh, UT Austin; Tudor Marian, Google; Hakim Weatherspoon, Cornell University

Log-structured storage designs can be used to solve the disk contention problem in modern datacenters, but they suffer from performance degradation due to garbage collection (GC) overheads. Ji Yong Shin presented Gecko, a system where a single log structure is distributed across a chain of drives, physically separating the tail of the log from its body.

Gecko cuts the log tail from the body so that GC reads do not interrupt the sequential write. This results in a single uncontended drive rather than N contended drives. Gecko avoids write-write contention and GC-write contention. Gecko design offers a tradeoff between potentially utilizing the maximum write throughput of all disks in an array and eliminating performance degradation due to gc-write contention. Gecko uses preferential caching of data, from the tail drive of the chain in a flash cache, to avoid read-write contention. Results comparing aggregate throughput for RAID-0 + LFS v/s Gecko using a move-to-tail GC were presented. Also it was shown that when compact-in-body GC is used the application throughput is not affected.

Umesh Maheshwari (Nimble Storage) first commented that Gecko would work with RAID-1 and then raised a question on how to extend this to parity-based RAID systems. Ji Yong Shin clarified that parity-based RAID can be implemented on top of Gecko by sending each parity stripe to each Gecko chain. Specifically, a disk chain in a Gecko-based RAID system will correspond to a disk in a regular RAID. Someone asked if a specific LFS was used to compare performance. Ji Yong Shin responded that they used the LFS used by Gecko, with chain-size 1, for comparisons. When asked about implementation details, Ji Yong responded that the current implementation is using block devices, but use of compression in such devices was not considered.

A Parallel Page Cache: IOPS and Caching for Multicore Systems

Da Zheng, Randal Burns, and Alexander S. Szalay, Johns Hopkins University

The current OS page cache is designed for magnetic disks (with thousands of IOPS) and for high cache hits. Whereas

cloud I/O is characterized by randomness and lower cache hits. Da Zheng presented a set-associative page cache for scalable parallelism of IOPS in multicore systems. The focus was on cloud workloads where most accesses are reads; few pages are accessed many times, and most pages are accessed few times.

The design eliminates lock contention and hardware cache misses by partitioning the global cache into many independent page sets, each requiring a small amount of metadata that fits in few processor cache lines. The design is extended by using message passing among processors in a non-uniform memory architecture (NUMA). It avoids the problem of long latencies due to remote memory access by partitioning the cache by NUMA nodes. All cores in a NUMA node share a cache partition. Each NUMA node is treated as a node in the distributed system. The evaluation studies scalability of the proposed cache with a high page turnover rate under random workload without cache hits. The cache hit rate and the overall runtime performance of the cache is done using a Zipfian workload. Da Zheng provided a performance evaluation on an SSD array, providing an in-kernel implementation and dynamic cache sizing as future work.

When asked about the overhead incurred if the implementation was done in the kernel rather than in user-space, Da Zheng responded that if performance is used as a metric to measure overhead he expects it to be the same. Someone else asked for the amount of contention in absolute time rather than in percent. Da Zheng responded that they measured the lock overhead with perf, which only shows the percentage of each component. To get the absolute time, they have to take the total time times the percentage of spin locks. The total runtime with 12 threads is about 28 seconds, so it's 28 * 52% = 14.56 seconds.

Efficient QoS for Multi-Tiered Storage Systems

Ahmed Elnably and Hui Wang, Rice University; Ajay Gulati, VMware Inc.; Peter Varman, Rice University

Providing performance isolation and QoS guarantees among various clients is challenging in multi-tiered storage systems. The notion of I/O cost used by existing solutions in such environments gets very hard to estimate and use. Ahmed showed through example that for existing fair schedulers, time-slice-based schedulers do not work well for multi-tiered storage. He also provided an example of a queue-partitioning-based scheme where static partitioning doesn't work well. Then Ahmed described a model called "reward scheduling" along with a corresponding algorithm, which favors the clients whose I/Os are less costly on the back-end storage array. This can be due to better locality, caching, hitting in the SSD tier, etc.

In their approach, each client is weighted to denote its relative priority. A running average of the response times of the last N requests of each client is used for I/O cost accounting. A queue is maintained per client with a tag and the scheduler dispatches the request with the smallest tag to the storage array when invoked. The clients who complete requests faster are given priority over those with slower requests, as are clients with higher static weights. The evaluation was done using a simulator for a multi-tiered array as well as on a Linux-based system with locally attached SSD and a disk. The evaluation showed that reward scheduling was able to maintain high utilization by favoring I/Os that are cheaper on the back-end array.

Ahmed raised a couple of interesting open question on whether interaction between cache management and scheduling algorithms could provide better results. He also asked whether it is possible to better isolate queueing delays from response time. When asked if the algorithm sacrifices system throughput, Ahmed responded that for most of the experiment the disk utilization was 100%, so the system throughput is not affected.

The key takeaway from the talk was that the problem of providing QoS in multi-tiered arrays is not solved using existing techniques, and new research is needed in this area. One also needs to figure out how to improve the interaction between an outside scheduler that is sending requests to the array and the internal array scheduling to better optimize the utilization of underlying storage devices.

Keynote Address: Designing Storage Systems with Flash

Umesh Maheshwari, Nimble Storage

Summarized by Rik Farrow (rik@usenix.org)

Umesh Maheshwari started out with a word completion exercise, asking the audience to finish sentences by providing these buzzwords: cloud, virtualized, dedup, big data, and flash. Five years ago no storage vendor had flash in their products, but today the big storage vendors and many start-ups are in the game. Is flash a done deal? No, everybody is doing it differently. Umesh then built a tree of decision points, such as if you use flash in storage, do you use it alone or with disk. His tree extends to five levels of decisions. Then he said, "In each of these cases, Nimble has made the right choice, or the right-hand choice," to laughter. "There is no right versus wrong here, and many of these choices are not mutually exclusive," he continued.

Umesh provided some background by explaining that he founded Nimble Storage with Varun Mehta (an early NetApp employee) in 2008, when there were no flash storage products. Their focus has been producing both good performance

and capacity for mainstream applications. Umesh then worked his way down the decision tree he presented earlier. While flash seems like a good idea in a server host, it actually will work just as well in a SAN once the queue depth gets deeper. Flash on storage supports multiple hosts, provides high availability, and can accelerate the storage block map or other metadata. At the next decision point, Umesh explained why hybrid makes the most sense: flash is expensive, and most data is cold. Enterprise flash costs 30x as much as enterprise disk. Then there is the uncertain long-term reliability of flash. These points all argue strongly for a hybrid solution.

The next decision point in his tree is simpler: flash as endpoint or accelerator. Flash used as an endpoint for storing data requires copying data into or out of flash, depending on whether it is hot or not. When using flash for acceleration, data copied to flash does not have to be copied back to disk. At the next level in Umesh's decision tree, he compares using flash as a write-back flash versus using it as a read cache. Umesh pointed out that NVRAM works better as a write-buffer, although flash can provide a much larger buffer. Flash also has the issue of write wear. By using flash as a read cache, you can control how often flash gets written. As for the unreliability of flash, the data in cache is a subset of clean data on disk, so there's no need for parity or mirroring. A checksum failure just means finding the data on disk.

The final decision point has to do with disk layout, read-optimized or write-optimized. Read-optimized layouts help with sequential reads as well as being simpler for calculating block offsets. In write-optimized layouts, blocks can go anywhere, requiring a large map and a more complex algorithm for locating blocks. But I/O has become increasingly write-dominated, as much as 80% currently. And systems include a large amount of memory, which acts as a cache for recently written data. Nimble focused on write-optimization, where they use the block map stored in flash to speed up garbage collection for coalescing blocks, both to improve read speed but also so they can write in large stripes using their log structured file system.

Umesh concluded with his wish for improvements in SSDs, mainly features that would fit best with their use of flash. Fred Douglas of EMC commented about cache: even though memory has gotten bigger, data has gotten much bigger as well. Umesh said that this depends on the application. In the case of big data, you will typically be reading in large chunks of data sequentially. Dilma Da Silva of IBM, said that if you don't trust the flash for write data, how can you use it to optimize it for reading the block map? How can you trust the block map? Umesh pointed out that the block map is just a cache of what's on disk, and that if a checksum fails when reading a block map location, you can just go back to the disk. Geoff Kuenning of Harvey Mudd College said that people have been pretending that SSDs have spinning heads and behave like disk drives. But Umesh's wish list shows he'd like things to be different. Umesh replied that flash manufacturers are optimizing their products for non-enterprise level products, so most of the smarts have to be in the device. He would like to leave some things, like wear-leveling, in the devices, but have more control over things like garbage collection and the ability to write to erasure blocks instead of pages.

Dealing with Devices

Summarized by Pankaj Pipada (ppipada@gmail.com)

Exploiting Peak Device Throughput from Random Access Workload

Young Jin Yu, Seoul National University; Dong In Shin, Taejin Infotec, Korea; Woong Shin, Nae Young Song, Hyeonsang Eom, and Heon Young Yeom, Seoul National University

Using polling instead of interrupts and removing delayed-execution due to the I/O scheduler and SoftIRQ handlers are common optimizations when using solid state devices to enhance random I/O performance. But this mechanism for handling I/O suffers a performance wall at 75,000 IOPS (approx. 13 usec/4KB). Young Jin Yu proposed a new batching scheme called "temporal merge," which dispatches non-contiguous block requests using a single I/O operation. This overcomes the disadvantages of the narrow block interface, and enables an OS to exploit the peak throughput of a storage device for small random requests as well as a single large request.

Temporal merge combines multiple (even non-sequential) requests within a short time window, and dispatches them by using a new I/O interface. In synchronous temporal merge, each thread submits a block request. A winner is chosen who combines concurrent block requests into one and dispatches it by using a new interface. The losing threads yield CPU and sleep until the completion of their requests. This method balances the synchronous I/O paths and batching to give low latency and high throughput, but the merge count is limited by the maximum number of threads entering into the I/O subsystem. Young Jin Yu proposed asynchronous temporal merge, which maximizes the accumulation of block requests in a queue when the concurrency is low. Young Jin Yu also proposed an extended block I/O interface of DRAM-based SSD and described how temporal merge can be implemented in the I/O subsystem in Linux. The experimental results show that under multithreaded random access workload, the proposed solution can achieve 87–100% of peak throughput of the SSD.

When asked about whether the CFQ I/O scheduler was used for implementation, Young Jin Yu responded that a custom-

ized I/O scheduler was used for evaluation. Regarding fairness of temporal merging, it is at least as good as a sync-poll mechanism since it guarantees FIFO dispatching order. However, it does not take process priority into consideration; the process priority of a winner may be lower than those of losing threads. Also it was suggested that atomic updates were done using standard test and set mechanisms which incurred an acceptable overhead.

Finding Soon-to-Fail Disks in a Haystack

Moises Goldszmidt, Microsoft Research

Moises Goldszmidt presented a statistical machine-learning-based detector of soon-to-fail disks. The model uses one performance signal for its prediction. The signal used is average-max-latency (AML) that goes through a model comparison filter (using Hidden Markov Models, HMM), and a peak counter. A logistic regression model then fuses the output of these two filters. The parameters of these models are automatically trained using signals from healthy and failed disks.

Evaluation was done by dividing the data into two sets of 1190 disks with 17 failed disks each. The first set is used for training, while the second is used for testing. The HMM models are trained by taking 24 hours of AML data at a time with a sliding window of four hours for 12 failed and 48 healthy disks. Logistic regression is trained using 200 disks, and 900 disks are used to set the threshold for minimizing false positives. During testing, simulating the production environment, the detector was able to predict 15 out of the 17 failed disks (88.2% detection) with 30 false alarms (2.56% false positive rate). The detector predicted the problem within two weeks to eight hours before failure. The workload was stable for all disks (healthy or soon-to-fail), and there is no solid correlation to SMART data.

When asked if there is any intuition to why there is no correlation to SMART signals, Moises Goldszmidt responded that more data is needed and speculated that from informal observations disks appear to fail for different reasons, each involving a different SMART signal. Moises also suggested that moving applications from the detected failed disk or modifying the amount of replication for the data can be the possible usages after an alarm signal by the detector. He also mentioned that the work started through an observation made by an operator in the performance of failing disks.

An Evaluation of Different Page Allocation Strategies on High-Speed SSDs

Myoungsoo Jung and Mahmut Kandemir, The Pennsylvania State University

Exploiting internal parallelism is becoming a key design issue in high-speed solid state disks (SSDs). Myoungsoo Jung presented a simulation of a cycle-accurate SSD platform with 24-page allocation strategies, geared toward exploiting both system-level parallelism and flash-level parallelism with a variety of design parameters.

The key findings of the experiments were: flash-level resource first-page allocation strategies can give better performance overall; channel-first page allocation schemes render high flash-level parallelism difficult under disk-friendly workloads; with most of the modern parallel data access methods, internal resources are significantly underutilized. Also, several optimization points were presented to achieve maximum internal parallelism. Incorporating a high-speed flash interface (400 MHz) in the design is an area of future work. Also, further evaluations using varying parameters such as different queue/buffer management, flash firmware, internal resource parameters, and more diverse workloads in evaluation was seen as future work.

When asked about the effect of larger pages on the performance, Myoungsoo Jung suggested that the channel contention would increase in the case of reads because channel bus activities of reads accounts for about 50% of the total execution time, and the amount of such bus activities mainly depends on their page sizes. He also suggested that reducing the eight-way channels to two-way or one-way would increase the resource conflict significantly.