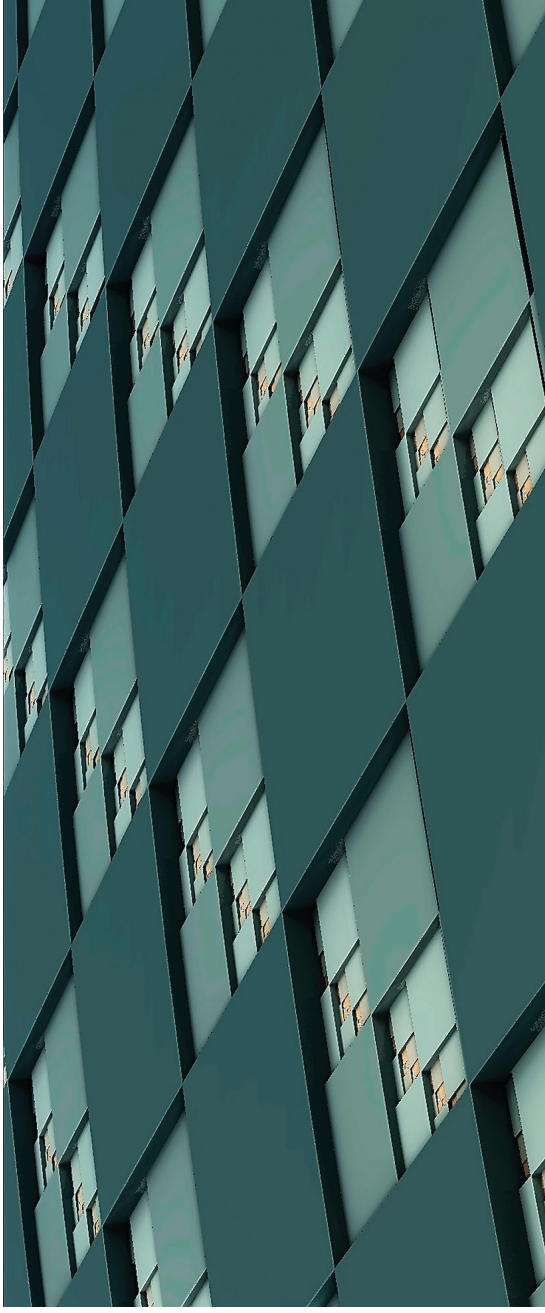
DOI:10.1145/2814342

Article development led by acmqueue
queue.acm.org

Implications of the datacenter's shifting center.

BY MIHIR NANAVATI, MALTE SCHWARZKOPF, JAKE WIRES, AND ANDREW WARFIELD

Non-Volatile Storage



FOR THE ENTIRE careers of most practicing computer scientists, a fundamental observation has consistently held true: CPUs are significantly more performant and more expensive than I/O devices. The fact that CPUs can process data at extremely high rates, while simultaneously servicing multiple I/O devices, has had a sweeping impact on the design of both hardware and software for systems of all sizes, for pretty much as long as we have been building them.

This assumption, however, is in the process of being completely invalidated.

The arrival of high-speed, non-volatile storage devices, typically referred to as storage class memories (SCM), is likely the most significant architectural change datacenter and software designers will face in the foreseeable future. SCMs are increasingly part of server systems, and they constitute a massive change: the cost of an SCM, at $3,000–$5,000, easily exceeds that of a many-core CPU ($1,000–$2,000), and the performance of an SCM (hundreds of thousands of I/O operations per second) is such that one or more entire many-core CPUs are required to saturate it.

This change has profound effects:

1. The age-old assumption that I/O is slow and computation is fast is no longer true: This invalidates decades of design decisions that are deeply embedded in today's systems.

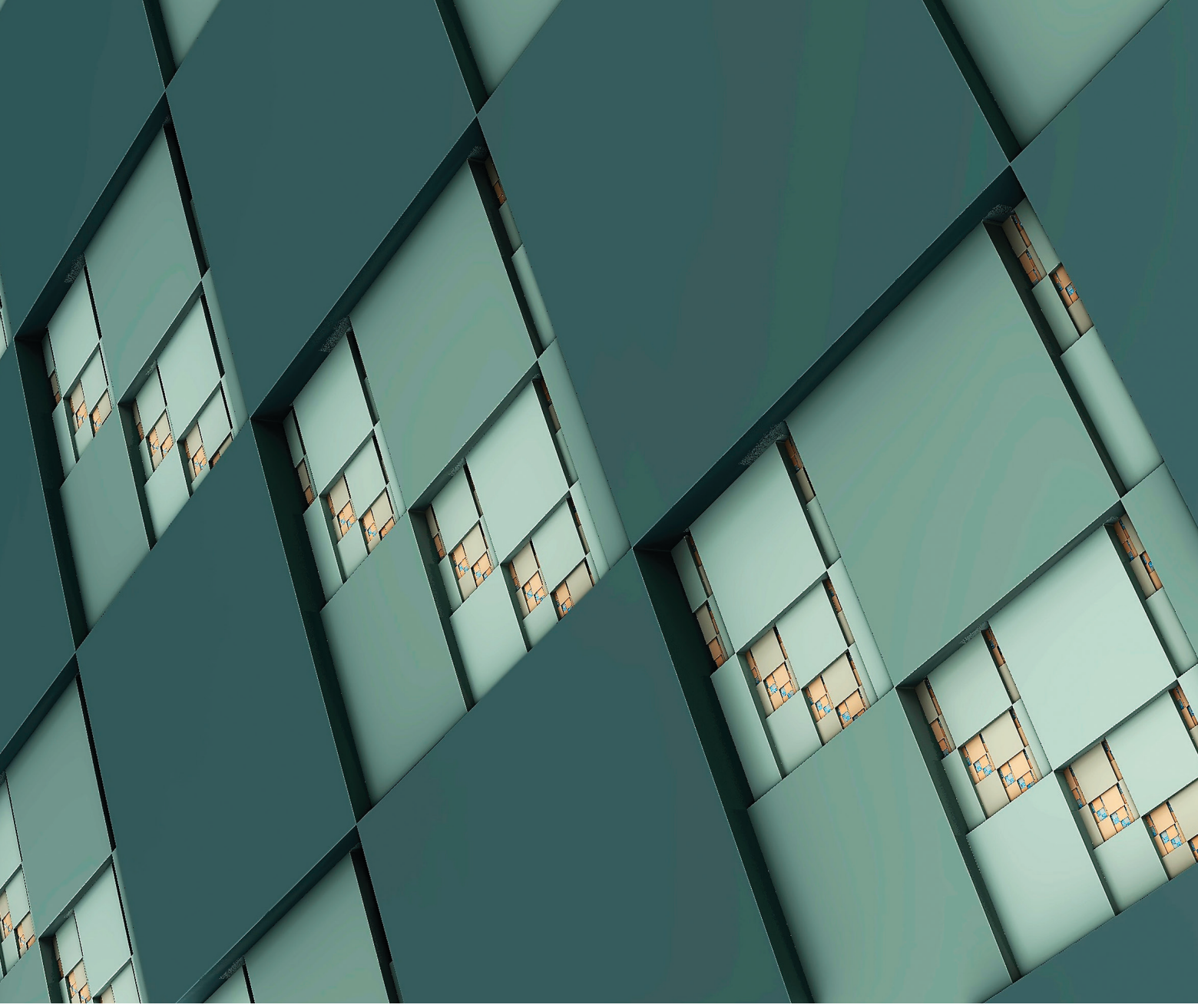


IMAGE BY MARALOVA

2. The relative performance of layers in systems has changed by a factor of a thousand over a very short time: This requires rapid adaptation throughout the systems software stack.

3. Piles of existing enterprise datacenter infrastructure—hardware and software—are about to become useless (or, at least, very inefficient): SCMs require rethinking the compute/storage balance and architecture from the ground up.

This article reflects on four years of experience building a scalable enterprise storage system using SCMs; in particular, we discuss why traditional storage architectures fail to exploit the performance granted by SCMs, what is required to maximize utilization, and what lessons we have learned.

Ye Olde World

"Processing power is in fact so far ahead of disk latencies that prefetching has to work multiple blocks ahead to keep the processor supplied with data. ... Fortunately, modern machines have sufficient spare cycles to support more computationally demanding predictors than anyone has yet proposed."

—Papathanasiou and Scott,[10] 2005

That disks are cheap and slow, while CPUs are expensive and fast, has been drilled into developers for years. Indeed, undergraduate textbooks, such as Bryant and O'Hallaron's *Computer Systems: A Programmer's Perspective*,[3] emphasize the consequences of hierarchical memory and the importance for novice developers to understand its impact on their programs. Perhaps less pedantically, Jeff Dean's "Numbers that everyone should know"[7] emphasizes the painful latencies involved with all forms of I/O. For years, the consistent message to developers has been that good performance is guaranteed by keeping the working set of an application small enough to fit into RAM, and ideally into processor caches. If it is not that small, we are in trouble.

Indeed, while durable storage has always been slow relative to the CPU, this "I/O gap" actually widened yearly throughout the 1990s and early 2000s.[10] Processors improved at a steady pace, but the performance of mechanical drives remained unchanged, held hostage by the physics of rotational velocity and seek times. For decades, the I/O gap has been the mother of invention for a plethora of creative schemes to

avoid the wasteful, processor-idling agony of blocking I/O.

Caching has always been—and still is—the most common antidote to the abysmal performance of higher-capacity, persistent storage. In current systems, caching extends across all layers: processors transparently cache the contents of RAM; operating systems cache entire disk sectors in internal buffer caches; and application-level architectures front slow, persistent back-end databases with in-memory stores such as memcached and Redis. Indeed, there is ongoing friction about where in the stack data should be cached: databases and distributed data processing systems want finer control and sometimes cache data within the user-space application. As an extreme point in the design space, RAMCloud[9] explored the possibility of keeping all of a cluster's data in DRAM and making it durable via fast recovery mechanisms.

Caching is hardly the only strategy to deal with the I/O gap. Many techniques literally trade CPU time for disk performance: compression and deduplication, for example, lead to data reduction, and pay a computational price for making faster memories seem larger. Larger memories allow applications to have larger working sets without having to reach out to spinning disks. Compression of main memory was a popular strategy for the "RAM doubling" system extensions on 1990s-era desktops.[12] It remains a common technique in both enterprise storage systems and big data environments, where tools such as Apache Parquet are used to reorganize and compress on-disk data in order to reduce the time spent waiting for I/O.

The Brave New World

"[M]ultiple sockets issuing IOs reduces the throughput of the Linux block layer to just about 125,000 IOPS even though there have been high end solid state devices on the market for several years able to achieve higher IOPS than this. The scalability of the Linux block layer is not an issue that we might encounter in the future, it is a significant problem being faced by HPC in practice today."

—Bjørling et al.,[2] 2013

The arrival of high-speed, non-volatile storage devices, typically referred to as storage class memories (SCM), is likely the most significant architectural change that datacenter and software designers will face in the foreseeable future.

Flash-based storage devices are not new: SAS and SATA SSDs have been available for at least the past decade, and have brought flash memory into computers in the same form factor as spinning disks. SCMs reflect a maturing of these flash devices into a new, first-class I/O device: SCMs move flash off the slow SAS and SATA buses historically used by disks, and onto the significantly faster PCIe bus used by more performance-sensitive devices such as network interfaces and GPUs. Further, emerging SCMs, such as non-volatile DIMMs (NVDIMMs), interface with the CPU as if they were DRAM and offer even higher levels of performance for non-volatile storage.

Current PCIe-based SCMs represent an astounding three-order-of-magnitude performance change relative to spinning disks (~100K I/O operations per second versus ~100). For computer scientists, it is rare the performance assumptions that we make about an underlying hardware component change by 1,000x or more. This change is punctuated by the fact the performance and capacity of non-volatile memories continue to outstrip CPUs in year-on-year performance improvements, closing and potentially even inverting the I/O gap.

The performance of SCMs means systems must no longer "hide" them via caching and data reduction in order to achieve high throughput. Unfortunately, however, this increased performance comes at a high price: SCMs cost 25x as much as traditional spinning disks ($1.50/GB versus $0.06/GB), with enterprise-class PCIe flash devices costing between $3,000–$5,000 each. This means the cost of the non-volatile storage can easily outweigh that of the CPUs, DRAM, and the rest of the server system they are installed in. The implication of this shift is significant: non-volatile memory is in the process of replacing the CPU as the economic center of the datacenter.

To maximize the value derived from high-cost SCMs, storage systems must consistently be able to saturate these devices. This is far from trivial: for example, moving MySQL from SATA RAID to SSDs improves performance only by a factor of 5–7[14]—significantly lower than the raw device differential. In a big data context, recent analyses of SSDs

by Cloudera were similarly mixed: "we learned that SSDs offer considerable performance benefit for some workloads, and at worst do no harm."[4] Our own experience has been that efforts to saturate PCIe flash devices often require optimizations to existing storage subsystems, and then consume large amounts of CPU cycles. In addition to these cycles, full application stacks spend some (hopefully significant) amount of time actually working with the data that is being read and written. In order to keep expensive SCMs busy, significantly larger numbers of CPUs will therefore frequently be required to generate a sufficient I/O load.

All in all, despite the attractive performance of these devices, it is very challenging to effectively slot them into existing systems; instead, hardware and software need to be designed together with an aim of maximizing efficiency.

Here, we discuss some of the techniques and considerations in designing for extremely high performance and utilization in enterprise storage systems:

Balanced systems address capacity shortfalls and bottlenecks in other components that are uncovered in the presence of SCMs. For example, sufficient CPU cores must be available and the network must provide enough connectivity for data to be served out of storage at full capacity. Failing to build balanced systems wastes capital investment in expensive SCMs.

Contention-free I/O-centric scheduling is required for multiple CPUs to efficiently dispatch I/O to the same storage device, that is, to share a single SCM without serializing accesses across all the CPUs. Failing to schedule I/O correctly results in sub-par performance and low utilization of the expensive SCMs.

Horizontal scaling and placement awareness addresses resource constraints by eschewing the traditional filer-style consolidation, and instead distributes data across the cluster and proactively moves it for better load balancing. Failing to implement horizontal scaling and placement awareness results in storage systems that cannot grow.

Workload-aware storage tiering exploits the locality of accesses in most workloads to balance performance, capacity, and cost requirements. High-speed, low-capacity storage is used to cache hot data from the lower-speed tiers, with the system actively promoting and demoting data as workloads change. Failing to implement workload-aware tiering results in high-value SCM capacity being wasted on cold data.

Finally, we conclude by noting some of the challenges in datacenter and application design to be expected from the even faster non-volatile devices that will become available over the next few years.

Balancing Systems

Can we just drop SCMs into our systems instead of magnetic disks, and declare the case closed? Not really. By replacing slow disks with SCMs, we merely shift the performance bottleneck and uncover resource shortfalls elsewhere—both in hardware and in software. As a simple but illustrative example, consider an application that processes data on disk by asynchronously issuing a large number of outstanding requests (to keep the disk busy) and then uses a pool of one or more worker threads to process reads

Figure 1. Per-packet processing time with faster network adapters.

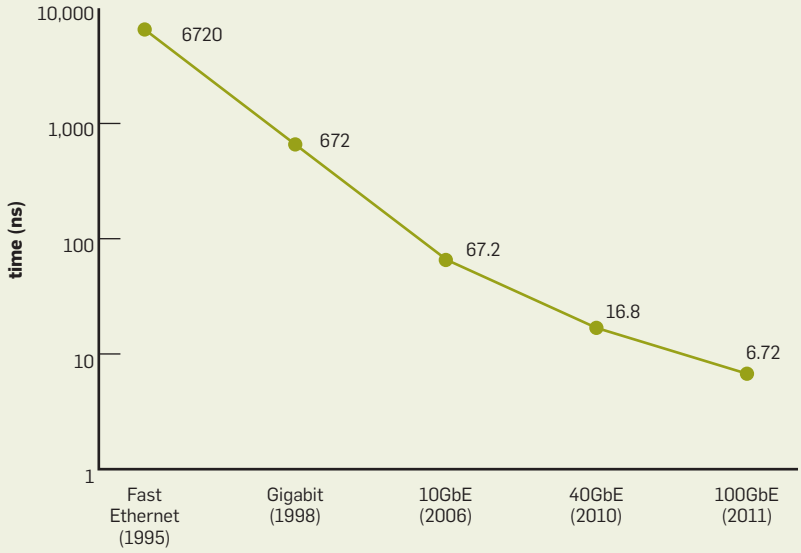


Figure 2. Progression in speed of SCMs compared to network adapters.

from disk as they complete. In a traditional system where disks are the bottleneck, requests are processed almost immediately upon completion and the (sensible) logic to keep disk request queues full may be written to keep a specified number of requests in flight at all times. With SCMs, the bottleneck can easily shift from disk to CPU: instead of waiting in queues ahead of disks, requests complete almost immediately and then wait for workers to pick them up, consuming memory until they are processed. As a result, we have seen real-world network server implementations and data analytics jobs where the concrete result of faster storage media is that significantly more RAM is required to stage data that has been read but not processed. Moving the performance bottleneck results in changes to memory demands in the system, which may, in the worst case, even lead to the host swapping data back out to disk!

Beyond the behavioral changes to existing software that result from the performance of SCMs, realizing their value requires they be kept busy. Underutilized and idle SCMs constitute waste of an expensive resource, and suggest an opportunity for consolidation of workloads. Interestingly, this is the same reasoning that was used, over a decade ago, to motivate CPU virtualization as a means of improving utilization of compute resources. Having been involved in significant system-building efforts for both CPU and now SCM virtualization, we have found achieving sustained utilization for SCMs to be an even more challenging goal than it was for CPUs. It is not simply a matter of virtualizing the SCM hardware on a server and adding more VMs or applications: We may encounter CPU or memory bottlenecks long before the SCM is saturated. Instead, saturating SCMs often requires using a dedicated machine for the SCM and spreading applications across other physical machines.

As a result, the cost and performance of storage devices dominates datacenter design. Ensuring their utilization becomes a key focus, and we found this can only be achieved by building balanced systems: systems with an appropriate number of cores and the right amount of memory to saturate exactly as many flash devices as needed for a given workload.

That balanced design with attention to storage can pay off is not a new insight: efforts like TritonSort, which won the sort benchmark in 2011 by carefully optimizing a cluster's magnetic disk throughput,[11] have done it before. However, such optimization efforts were rare in the age of magnetic disks, and are—certainly in TritonSort's case—hardware- and workload-specific. The traditional slowness of disks meant that balancing storage speed with other system components was out of the question for most workloads, so efforts focused on *masking* access to storage in order to stop it from disrupting the balance of other resources.

Good SCM utilization requires this balance far more urgently: buying too many flash devices and too few cores or too little RAM ends up wasting capital, but buying too few, sparsely spread out flash devices risks bottlenecks in accessing them—though the bottlenecks will most likely be in system resources other than the SCM itself! The right balance is, of course, still a property of the workload, which in combination with our consolidation goal makes it an incredibly challenging target to shoot for: heterogeneous workloads already make it difficult to achieve full system utilization, even before considering the storage layer. An example of this from our own work has been in the seemingly simple problem of dynamically scaling a traditional NFS server implementation to expose more physical bandwidth as additional SCMs (and accompanying CPUs and NICs) are added to the system.[5]

Contention-Free I/O-Centric Scheduling

Even if the hardware resources and the workload are perfectly balanced, the temporal dimension of resource sharing matters just as much. For a long time, interrupt-driven I/O has been the model of choice for CPU-disk interaction. This was a direct consequence of the mismatch in their speeds: for a core running at a few gigahertz, servicing an interrupt every few milliseconds is fairly easy. A single core can service tens or hundreds of disks without getting overwhelmed and missing deadlines.

This model must change drastically for low-latency ("microsecond era") devices. However, storage devices are not the only peripheral to have seen changes in speed—network devices have seen similar rapid improvements in performance from 10G to 40G and, recently, 100G. Maybe storage systems can use the same techniques to saturate devices?

Unfortunately, the answer is not a simple yes or no. The gains made by networking devices pale in comparison to the dramatic rise in speed of storage devices; for instance, Figures 1 and 2 show that in the same period of time when networks have sped up a thousandfold, storage devices have become a million times faster. Furthermore, storage stacks often have to support complex features such as compression, encryption, snapshots, and deduplication directly on the datapath, making it difficult to apply optimizations that assume independent packets without data dependencies.

One technique for reducing latency commonly adopted by network devices is to eliminate interrupt processing overhead by transitioning to polling when the system is under high load. Linux NAPI and Intel Busy Poll Sockets implement a polling mode for network adapters, which eliminates both the context switch and the cache and TLB pollution associated with interrupts. Also, busy polling cores never switch to power-saving mode, thereby saving on the cost of processor state transitions. Switching network adapters to polling mode reduces latency by around 30%, and non-volatile storage has demonstrated similar improvements.[16]

Polling comes with its own set of challenges, however. A CPU has responsibilities beyond simply servicing a device—at the very least, it must process a request and act as either a source or a sink for the data linked to it. In the case of data parallel frameworks such as Hadoop and Spark,[7,17] the CPU may also be required to perform more complicated transformations on the data. Thus, polling frequencies must be carefully chosen to ensure neither devices nor compute suffer from starvation, and scheduling strategies de-

signed to exploit traditional I/O-heavy workloads need to be reevaluated, since these workloads are now necessarily compute-heavy as well.

At 100K IOPS for a uniform random workload, a CPU has approximately 10 microseconds to process an I/O request. Because current SCMs are often considerably faster at processing sequential or read-only workloads, this can drop to closer to 2.5 microseconds on commodity hardware. Even worse, since these requests usually originate from a remote source, network devices have to be serviced at the same rate, further reducing the available per-request processing time. To put these numbers in context, acquiring a single uncontested lock on today's systems takes approximately 20ns, while a non-blocking cache invalidation can cost up to 100ns, only 25x less than an I/O operation.

Current SCMs can easily overwhelm a single core; they need multiple cores simultaneously submitting requests to achieve saturation. While hardware multi-queue support allows parallel submission, the kernel block layer serializes access to the queues and requires significant redesign to avoid contended locks.[2] However, even with a contention-free block layer, requests to overlapping regions must be serialized to avoid data corruption.

Another key technique used by high-performance network stacks to significantly reduce latency is bypassing the kernel and directly manipulating packets within the application.[13] Furthermore, they partition the network flows across CPU cores,[1,8] allowing the core that owns a flow to perform uncontended, lock-free updates to flow TCP state.

While bypassing the kernel block layer for storage access has similar latency benefits, there is a significant difference between network and storage devices: network flows are largely independent and can be processed in parallel on multiple cores and queues, but storage requests share a common substrate and require a degree of coordination. Partitioning both the physical storage device and the storage metadata in order to give individual CPU cores exclusive access to certain data is possible, but it requires careful in-data-structure design that has not been required of storage and file system designers in the past. Our experience has been that networking code often involves data structures that must be designed for performance and concurrency, while file system code involves complex data dependencies that require careful reasoning for correctness. With SCMs, systems designers are suddenly faced with the need to deal with both of these problems at once.

Beyond the behavioral changes to existing software that result from performance of SCMs, realizing their value requires they be kept busy. Underutilized and idle SCMs constitute waste of an expensive resource, and suggest an opportunity for consolidation of workloads.

The notion of I/O-centric scheduling recognizes that in a storage system, a primary task of the CPU is to drive I/O devices. Scheduling quotas are determined on the basis of IOPS performed, rather than CPU cycles consumed, so typical scheduling methods do not apply directly. For example, a common legacy scheduling policy is to encourage yielding when lightly loaded, in exchange for higher priority when busy and in danger of missing deadlines—a strategy that penalizes device-polling threads that are needed to drive the system at capacity. The goal of I/O-centric scheduling must be to prioritize operations that drive device saturation while maintaining fairness and limiting interference across clients.

Horizontal Scaling and Placement Awareness

Enterprise datacenter storage is frequently consolidated into a single server with many disks, colloquially called a JBOD (Just a Bunch Of Disks). JBODs typically contain 70–80 spinning disks and are controlled by a single controller or "head," and provide a high-capacity, low-performance storage server to the rest of the datacenter.

JBODs conveniently abstract storage behind this controller; a client need only send requests to the head, without requiring any knowledge of the internal architecture and placement of data. A single SCM can outperform an entire JBOD, but it provides significantly lower capacity. Could a JBOD of SCMs provide high-speed and high-capacity storage to the rest of the datacenter? How would this affect connectivity, power, and CPU utilization?

An entire disk-based JBOD requires less than 10G of network bandwidth, even when running at full capacity. In

contrast, a JBOD of SCMs would require 350G–400G of network bandwidth, or approximately 10 40G network adapters. At 25W per SCM, the JBOD would draw approximately 3,000W.

Obviously, this is impractical, but even worse, it would be terribly inefficient. A single controller is simply not capable of mediating access to large numbers of SCMs simultaneously. Doing so would require processing an entire request in around 100ns—the latency of a single memory access. A centralized controller would thus leave storage hardware severely underutilized, providing a poor return on the investment in these expensive devices. A different approach is required.

Distributing accesses across cores, that is, having multiple heads, requires coordination while accessing file system metadata. Multiple network adapters within the JBOD expose multiple remote access points, requiring placement-aware clients that can direct requests to the correct network endpoint and head. At this point the JBOD resembles a distributed system, and there is little benefit to such consolidation. Instead, horizontal scaling out across machines in the cluster is preferable, as it provides additional benefits related to provisioning and load balancing.

Rather than finalizing a specification for a JBOD when first building the datacenter, scaling out allows storage servers to be added gradually in response to demand. This can lead to substantial financial savings as the incrementally added devices reap the benefits of Moore's Law. Further, since these servers are provisioned across racks, intelligent placement of data can help alleviate hotspots and their corresponding network bottlenecks, allowing for uniformly high utilization.

However, maintaining high performance across clustered machines requires much more than just reducing interrupt overheads and increasing parallelism. Access to shared state, such as file system metadata, must be carefully synchronized, and additional communication may be required to serve large files spread across multiple servers. Updates to files and their metadata must be coordinated across multiple machines to prevent corruption, and the backing data structures themselves must scale across cores with minimal contention. Shifting workload patterns often lead to poor load balancing, which can require shuffling files from one machine to another. Distributed storage systems have faced these issues for years, but the problems are much more acute under the extremely high load that an SCM-based enterprise storage system experiences.

The takeaway here is that unless the majority of data in the system is hot, it is extremely inefficient to store it all in high-speed flash devices.

Workload-Aware Storage Tiering

The capacity and performance of SCMs are orthogonal: a 4TB flash drive has about the same performance characteristics as a 1TB or 2TB drive in the same series. Workload requirements for capacity and performance are not matched to hardware capabilities, leading to underutilized disks; for example, a 10TB dataset with an expected load of 500K IOPS is half idle when all the data is stored in 1TB SCMs capable of 100K IOPS.

Besides the obvious cost inefficiency of having underutilized expensive SCMs, there are processor socket connectivity constraints for PCIe-based SCMs. A single such device requires four to eight PCIe lanes, which are shared across all the high-speed I/O devices, limiting the number of drives a single socket can support. In contrast, SATA drives, whether spinning disk or flash, do not count against the same quota.

The takeaway here is that unless the majority of data in the system is hot, it is extremely inefficient to store it all in high-speed flash devices. Many workloads, however, are not uniformly hot, but instead follow something closer to a Pareto distribution: 80% of data accesses are concentrated in 20% of the dataset.

A hybrid system with different tiers of storage media, each with different performance characteristics, is a better option for a mixture of hot and cold data. SCMs act as a cache for slower disks and are filled with hot data only. Access patterns vary across time and need to be monitored so the system can actively promote and demote data to match their current hotness level. In practice, tracking miss ratio curves for the system al-

lows estimation of the performance impact of changing the cache size for different workloads with fairly low overheads[15] and enables fine-grained decisions about exactly where data should reside.

Tiering is an extension to caching mechanisms that already exist. System designers must account for tiers independently, rather like ccNUMA machines where local and remote memories have significantly different performance. Tiering allows systems to scale capacity and performance independently—a necessity for enterprise storage.

Despite the obvious benefits of tiering, it is fraught with complications. The difference in granularity of access at different storage tiers causes an impedance mismatch. For example, SCMs excel at random accesses, while spinning disks fare better with sequential access patterns. Maintaining a degree of contiguity in the disk tier may result in hot and cold data being "pinned" together in a particular tier.

This granularity mismatch is not unique to storage devices: MMUs and caches also operate at page and cache line granularities, so a single hot byte could pin an entire page in memory or a line in the cache. While there are no perfect solutions to this problem, the spatial locality of access patterns offers some assistance: predictable, repeated accesses allow for some degree of modeling to help identify and fix pathological workloads.

In adequately provisioned systems, simple tiering heuristics are often effective for making good use of hardware without degrading performance. However, different workloads may have differing priorities. In such cases, priority inversion and fairness become important criteria for determining layout. Tiering mechanisms must support flexible policies that prevent active but low-priority workloads from interfering with business-critical workloads. There is often a tension between such policies and the desire to maximize efficiency; balancing these concerns makes tiering a challenging problem.

The Future

PCIe SSDs are the most visible type of SCMs, and have already had a significant impact on both hardware and software design for datacenters—but they are far from the only member of that class of devices.

NVDIMMs have the performance characteristics of DRAM, while simultaneously offering persistence. A common recent approach to designing NVDIMMs has been to match the amount of DRAM on a DIMM with an equivalent amount of flash. The DRAM is then used as if it were normal memory, and the flash is left entirely alone until the system experiences a power loss. When the power is cut, a supercapacitor is used to provide enough power to flush the (volatile) contents of RAM out to flash, allowing it to be reloaded into RAM when the system is restarted. Flash-backed DRAMs are available today, and newer memory technologies such as resistive and phase-change memories have the potential to allow for larger and higher-performance nonvolatile RAM.

This emerging set of non-volatile memories has exposed software inefficiencies that were previously masked by the performance characteristics of the devices that they ran on—not just spinning disks, but even first-generation SSDs. We believe as these nefficiencies become more pronounced (and they likely will, given the continued fast pace of improvement in SCM performance), they will invite innovation in software systems design. This will have to occur at many layers of the infrastructure stack in order to be able to take advantage of fast non-volatile storage; what we see today is just the beginning! C

Related articles on queue.acm.org

Don't Settle for Eventual Consistency
Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, David G. Andersen
http://queue.acm.org/detail.cfm?id=2610533

Enterprise Grid Computing
Paul Strong
http://queue.acm.org/detail.cfm?id=1080877

Why Cloud Computing Will Never Be Free
Dave Durkee
http://queue.acm.org/detail.cfm?id=1772130

References

1. Belay, A., et al. IX: A protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*.
2. Bjørling, M., Axboe, J., Nellans, D. and Bonnet, P. Linux block IO: introducing multi-queue SSD access on multi-core systems. In *Proceedings of the 6th International Systems and Storage Conference*, 2013.
3. Bryant R. E. and O'Hallaron, D.R. *Computer Systems: A Programmer's Perspective, Vol. 2*. Prentice Hall, Englewood Cliffs, NJ, 2003.
4. Chen. Y. The truth about MapReduce performance on SSDs; http://radar.oreilly.com/2015/07/the-truth-about-mapreduce-performance-on-ssds.html
5. Cully, B. et al. 2014. Strata: Scalable high-performance storage on virtualized non-volatile memory. In *Proceedings of the 12th USENIX conference on File and Storage Technologies*, 2014.
6. Dean, J. Software engineering advice from building large-scale distributed systems. CS295 Lecture at Stanford University (July 2007); http://research.google.com/people/jeff/stanford-295-talk.pdf
7. Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.
8. Jeong, E.Y. et al. mTCP: A highly scalable user-level TCP stack for multicore systems. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, 2014.
9. Ongaro, D., Rumble, S. M., Stutsman, R., Ousterhout, J. and Rosenblum, M. Fast crash recovery in RAMCloud. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011.
10. Papathanasiou, A.E. and Scott. M.L. Aggressive prefetching: An idea whose time has come. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating Systems*, 2005.
11. Rasmussen, A. et al. 2011. TritonSort: A balanced large-scale sorting system. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 2011.
12. Rizzo, L. A very fast algorithm for RAM compression. *ACM SIGOPS Operating Systems Review 31*, 2 (1997), 36–45.
13. Rizzo, L. Netmap: A novel framework for fast packet I/O. In *Proceedings of the USENIX Annual Technical Conference*, 2012.
14. Tkachenko, V. Intel SSD 910 vs. HDD RAID in TPC-C-MySQL benchmark; https://www.percona.com/blog/2012/09/11/intel-ssd-910-vs-hdd-raid-in-tpcc-mysql-benchmark/
15. Wires, J., Ingram, S., Drudi, Z., Harvey, N. J. A., Warfield, A. Characterizing storage workloads with counter stacks. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.
16. Yang, J., Minturn, D.B., Hady, F. When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, 2012.
17. Zaharia, M. et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

Mihir Nanavati is a developer at Coho Data, where he works on extracting maximum performance from high-speed I/O devices. He is also a Ph.D. student at the University of British Columbia. His research interests include performance and scalability of multicore systems, and security and isolation in virtualized and containerized environments.

Malte Schwarzkopf is a Ph.D. candidate at the University of Cambridge Computer Laboratory and moonlights part-time at Coho Data. His research is primarily on operating systems and scheduling for datacenters, but he has worked at many levels of the stack as part of the CamSaS initiative (http://camsas.org/).

Jake Wires is a principal software engineer at Coho Data and a doctoral candidate at the University of British Columbia. He is broadly interested in the design of storage systems and scalable data processing.

Andrew Warfield is co-founder and CTO of Coho Data and an associate professor at the University of British Columbia. He is broadly interested in software systems.