RETHINKING FLASH IN THE DATA CENTER

DEPLOYMENT OF FLASH MEMORY DEPENDS ON MAKING THE MOST OF ITS UNIQUE PROPERTIES INSTEAD OF TREATING IT AS A DROP-IN REPLACEMENT FOR EXISTING TECHNOLOGIES.

David G. Andersen
Carnegie Mellon University

Steven Swanson
University of California, San Diego

Over the past few years, computer systems of all types have started integrating flash memory. Initially, flash's small size, low power consumption, and physical durability made it a natural fit for media players and embedded devices. Lately, flash's rising density has won it a place in laptops and some desktop machines.

Flash is now poised to make deep inroads into the data center. There, flash memory's high density, low power, and low-cost I/Os per second will drive its adoption and enable its application far beyond simple hard drive replacements. To date, however, many uses of flash have been hamstrung by a fundamental challenge of the technology: Flash is neither magnetic disk nor DRAM. It has its own performance advantages and quirks that system designers must address at several levels to best exploit it.

Disk and DRAM replacement

So far, most proposed applications for flash in the data center have fallen into two categories.

The first category is *disk replacement*. Quick access time and low power requirements make flash a compelling replacement for conventional disks, albeit at much lower density and higher cost. Accounting for servers and supporting infrastructure, solid-state disks (SSDs) consume roughly 10 times less energy when idle than disks. They deliver 2.6 times more bandwidth per watt and 3.2 times more bandwidth per dollar, 25 times more I/O operations per second (IOPS) per dollar, and 2,000 times more IOPS per watt (see Tables 1 and 2).

Flash sometimes also serves as a *DRAM replacement*. Density and (again) energy efficiency let flash compete with DRAM in applications where latency and bandwidth are less important. Flash consumes one-fourth the power of DRAM per byte at one-fifth the price.

Flash memory will remain a contender for both roles for the foreseeable future, but additional opportunities and challenges are on the horizon. Technology scaling will continue to increase bit density for another 1 to 2 process generations, which will drive down costs. However, smaller flash cells are less reliable and less durable. In the past two years, lifetime program/erase cycle ratings for high-density flash devices have dropped from 10 thousand to five thousand cycles. Raw bit error rates have increased as well. Understanding how to apply this shifting technology in the changing landscape of datacenter computing requires careful design.

Examples of how blithely applying SSDs to some datacenter applications can result in disappointing performance are easy to find. Our experience with a key-value store designed to hold huge numbers of small key-value pairs elicited nearly worst-case performance from SSDs. The FAWN-KV key-value storage system, developed as a part

Published by the IEEE Computer Society

Table 1. Basic disk and solid-state disk (SSD) performance and cost.

Drive	Cost	Capacity	Power draw while reading data	Read IOPS	Throughput
2.5'' disk	$40	320 Gbytes	2.1 watts/0.75 watt	240	80 Mbytes per second (spec sheet)
Solid-state disk	$220	80 Gbytes	0.1 watt	35,000	250 MBps

Table 2. Per-dollar and per-watt performance.

Drive	Gbytes per dollar	IOPS per dollar	IOPS per watt	Throughput per dollar	Throughput per watt
2.5'' disk	4	3	104	1 Mbytes/second	40 MBps
Solid-state disk	0.36	159	220,000	1.13 MBps	2,500 MBps

of the Fast Array of Wimpy Nodes (FAWN) project, can handle more than 200 times more inserts per second than the traditional, non-flash-optimized Berkeley DB, on both older flash devices and modern SSDs.[1]

Flash translation layer

The flash translation layer (FTL) hides many of flash's remaining warts. It provides reliability and the abstraction of a uniform block address space. This is necessary since flash cannot do in-place updates, flash cells wear out after between five thousand and one million program/erase cycles, and even read operations can potentially corrupt data. Through heroic engineering and daunting complexity, the FTL masks these problems, but its performance impact can be significant. Intel's Extreme SSDs have a read latency of 85 μs, but the flash chips the drive uses internally have a read latency of just 25 to 35 μs.

Flash clearly needs some kind of management layer to manage media errors and provide wear-leveling to maximize device lifetime. What interface should that layer present to the rest of the system, and where should it be implemented? Should the SSD give the application control over flash management, or should applications express their requirements to the SSD?

Exposing a richer interface to the system might include providing multiple ''personalities'' to the system. These could include a disk personality (the current standard) as well as others that might expose flash's block and page structure while providing block-level wear-leveling. Another personality would expose the flash directly and require the system to manage wear-leveling and remapping as needed by that application. The interface could let the system partition the SSD and expose a different personality for each partition. For such a storage device, customization could occur in the kernel driver or even at the application level, letting applications map their storage needs directly and efficiently onto the flash. Aggregating management across multiple flash devices would let a large server (or even a server farm) make global optimizations based on error rate and performance variation, allowing it to extract the maximum performance from the flash array while meeting application-specific reliability targets.

In the second approach, the FTL might provide an object store abstraction that lets applications indicate use patterns on a per-object basis. An application might designate one object for streaming, sequential writes and another for fast, random reads.

Example applications

We close with three examples of applications that could exploit a lower-level interface to flash.

Bloom filters store set membership information, with a small chance of false positives,

in a large bit array (for example, a flash page). To store an item, the filter computes k different hash values, and marks the bits addressed by those values. To test for membership, the filter applies the same hash functions and checks the k resulting locations. If all locations are marked, the item might be in the set.

Flash device data sheets recommend programming each page of flash only once between erase operations, to manage the *program disturb* phenomenon—that is, a program to one page could corrupt data on another page. In practice, flash chips allow multiple programs to a page with the caveat that each program can only turn 1s (the *erased* state) into 0s (the *programmed* state). Data we collected on real devices demonstrates that the single-level cell flash devices found in high-end SSDs can tolerate several hundred repeated programs before significant corruption occurs.[2] This combination of characteristics makes these devices a perfect fit for implementing Bloom filters. To insert an element into a filter, the application programs the bit vector for the element onto the page.

The program operation performs an effective logical OR with bits already programmed. To maintain data integrity, the application copies the Bloom filter to the next page every 1,000 insertions or so.

Other tricks are also possible. Write-once data encodings can allow applications to write multiple sets of arbitrary logical bits to a single page by encoding each pair of logical bits as three physical bits and ensuring that the second write only changes 1s to 0s. This technique can significantly improve flash lifetime and energy efficiency.[2] The trade-off is that both of these uses violate the manufacturers' guidelines. Whether that risk is worth additional lifetime or performance might best be decided on a per-application basis.

A richer interface to flash would also enable less risky ventures by eliminating the one-size-fits-all FTL. Several important applications for flash do not need wear-leveling because they are already log structured. Similar to our log-structured key-value store, Google has explored a design for large data caches. Using a first-in, first-out replacement policy leads to an elegant, log-structured management scheme that is a perfect match for flash memory.[3] An FTL would only get in the way.

Conventional flash interfaces and FTLs are innocuous for consumer devices, where performance is less critical. In the data center, however, the story is different. The value of replacing a one-size-fits-all FTL with a refined, mission-specific flash management system is potentially enormous. The challenge is to define interfaces and abstractions that make flash easy to manage while exposing flash's capabilities and performance potential. MICRO

References

1. D.G. Andersen et al., ''Fawn: A Fast Array of Wimpy Nodes,'' *Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles* (SOSP 09), ACM Press, pp. 1-14.
2. L.M. Grupp et al., ''Characterizing Flash Memory: Anomalies, Observations, and Applications,'' *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 09), ACM Press, pp. 24-33.
3. A. Borchers, ''Google's Experiences with Flash Memory,'' talk presented at the 2010 Nonvolatile Memory Workshop, 2010.

David G. Andersen is an assistant professor at Carnegie Mellon University. His research interests include distributed systems, energy efficient computing, and networks. Andersen has a PhD in computer science and electrical Engineering from the Massachusetts Institute of Technology. He is a member of IEEE, ACM, and Usenix.

Steven Swanson is an assistant professor in the Department of Computer Science and Engineering at the University of California, San Diego. His research focuses on high-performance, solid-state storage systems, low-power architectures, and memory system optimizations for multiprocessors. Swanson has a PhD in computer science from the University of Washington. He is a member of ACM.

Direct questions or comments about this article to David Andersen, dga@cs.cmu.edu, or Steven Swanson, swanson@cs.ucsd.edu.