Solid-state Cache

Mohit Saxena and Michael M. Swift
Department of Computer Sciences
University of Wisconsin-Madison
{*msaxena,swift*}*@cs.wisc.edu*

1 Introduction

Solid-state drives (SSDs) composed of multiple flash memory chips are often deployed as a cache in front of a cheap and slow disks [5, 3, 6]. This provides the performance of flash with the cost of disk for large data sets, and is actively used by Facebook and others to provide low-latency access to petabytes of data [4].

An SSD-backed cache, though, is limited by its narrow block interface and internal block management, both of which are designed to serve as a disk replacement [1, 7, 8]. Caches have at least three different behaviors than general-purpose storage. First, data in a cache may be present elsewhere in the system, and hence need not be durable. Thus, caches have more flexibility in how they manage data than a device dedicated to storing data persistently. Second, a cache stores data from a separate address space, the disks', rather than at native addresses. Thus, using a standard SSD as a cache requires an additional step to map block addresses from the disk into SSD addresses for the cache. If the cache is to survive crashes, this map must be persistent. Third, the consistency requirements for caches differ from storage devices. A cache must ensure it never returns stale data, but can also return nothing if the data is not present. In contrast, a storage device provides ordering guarantees on when writes become durable.

FlashTier is a caching system designed for a new type of device, a *solid-state cache (SSC)*. A *cache manager* in the operating system storage stack automatically migrates data between the flash caching tier and disk storage. This design provides a clean separation between the caching device and its internal structures, the system software managing the cache, and the disks storing data.

FlashTier exploits the three features of caching workloads to improve over SSD-based caches. First, FlashTier provides a *unified address space* that allows data to be written to the SSC at its disk address. This removes the need for a separate table mapping disk addresses to SSD addresses. In addition, an SSC uses internal data structures tuned for large, sparse address spaces to maintain the mapping of block number to physical location in flash.

Second, FlashTier provides *cache consistency guarantees* to ensure correctness following a power failure or system crash. It provides separate guarantees for clean and dirty data to support both write-through and write-back caching. In both cases, it guarantees that stale data will never be returned. Furthermore, FlashTier introduces new operations in the SSC interface, *evict* to invalidate data, and *exists* to test whether a block is present, and *clean* to indicate that data is clean and may be safely evicted. As a result, cache software can always use data from the SSC without verifying its freshness. FlashTier ensures that internal SSC metadata is always persistent and recoverable after a crash, allowing cache contents to be used after a failure.

Finally, FlashTier leverages its status as a cache to reduce the cost of garbage collection. Unlike a storage device, which promises to never lose data, a cache can evict blocks when it is beneficial. For example, flash must be erased before being written, requiring a garbage collection step to create free blocks. An SSD must copy live data from blocks before erasing them, requiring additional space for live data and time to write the data. In contrast, an SSC may instead evict the data, freeing more space faster.

This design allows an SSC to be adaptive to its workload: it may shift its internal use of flash resources between capacity (storing more live data), endurance (spreading less data over more cells), and write performance (providing more pre-erased blocks to accept new data). Thus, in a workload with a low churn, it can use the full capacity of the device. For a workload with frequent changes to the working set, it may shift resource to provide less cache capacity but greater performance for adding data to the cache.

While our first study of flash interfaces will be caches, we plan to then turn our attention to supporting other data structures in flash. Our focus is on the uses of flash where the existing block interface does not match the application need, and a lower- or higher-level interface would be superior. Specifically, we plan to investigate uses of flash for high-performance key-value stores and as intermediate storage for map-reduce computations. Key-value systems, such as FlashStore [2], already treat flash as a log and perform compaction in the application to avoid garbage collection and random writes in the device. We believe that a better interface, such as large write-once segments, which match the capabilities of flash, would be a better for for these applications. However, we still want to preserve the benefits of flash translation layer, such as internal management of wear and bad blocks, as well as the ability to use internal bandwidth to transfer data between flash pages. Thus, we seek a design that virtualizes flash at coarse granularity, to allow wear leveling, and also provides an efficient DMA-like mechanism to for application-directed compaction.

References

[1] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J., MANASSE, M., AND PANIGRAHY, R. Design tradeoffs for ssd performance. In *USENIX* (2008).

[2] DEBNATH, B., SENGUPTA, S., AND LI, J. Flashstore: High throughput persistent key-value store. In *VLDB* (2010).

[3] EMC. Fully Automated Storage Tiering (FAST) Cache. `http://www.emc.com/about/glossary/fast-cache.htm`.

[4] FACEBOOK INC. Facebook FlashCache. `https://github.com/facebook/flashcache`.

[5] KGIL, T., AND MUDGE, T. N. Flashcache: A nand flash memory file cache for low power web servers. In *CASES* (2006).

[6] OCZ. OCZ Synapse Cache SSD. `http://www.ocztechnology.com/ocz-synapse-cache-sata-iii-2-5-ssd.html`.

[7] PRABHAKARAN, V., RODEHEFFER, T., AND ZHOU, L. Transactional flash. In *OSDI* (2008).

[8] WU, M., AND ZWAENEPOEL, W. envy: A non-volatile, main memory storage system. In *ASPLOS-VI* (1994).