Solid-State Cache Management

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

We describe FlashTier*, a system architecture built upon* solid-state cache *(SSC), a novel flash device with an interface designed for caching.*

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, 2, 7]. 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 [3].

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, 8, 9]. 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.

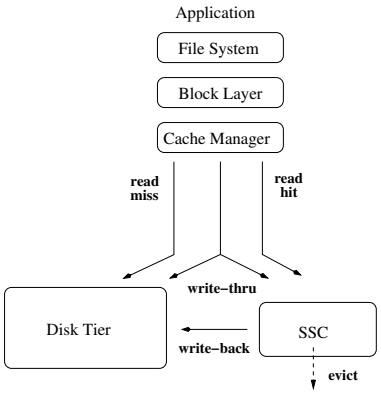

Figure 1: **FlashTier Request Path:** A cache manager forwards block read/write requests to disk and solid-state cache.

FlashTier is a caching system designed for a new type of device, a *solid-state cache (SSC)* (see Figure 1). 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

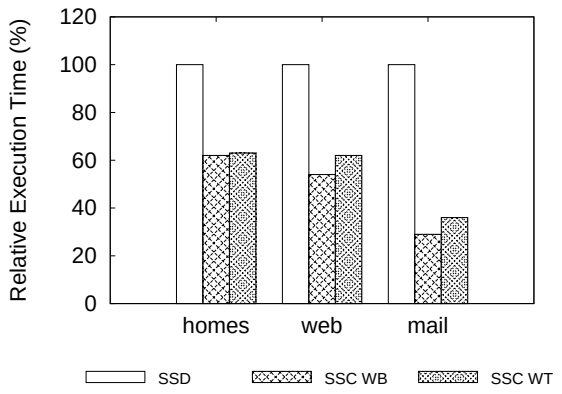

Figure 2: **System Comparison**: SSD cache compared to SSC in both write-back and write-through modes.

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.

In addition, 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. This requires the SSC to adapt its capacity at runtime provisioned for internal free space and wear management.

2 Results

We implemented an SSC simulator and a cache manager for Linux, and evaluate FlashTier on three real-world traces collected from file, mail and web servers [6]. In Figure 2, we compare the SSD cache against FlashTier in write-back and write-through modes.

FlashTier outperforms the SSD cache by 39-71% in write-back mode and by 37-64% in write-through mode. FlashTier performs silent eviction that greatly reduces the cost of garbage collection for writing new data. Write-back caching performs better than write-through mode because of a lower cost for writing data. A write-back cache sends the writes to the SSD or SSC and lazily copies it to disk, which reduces the cost of writing data. The write-through configuration must write all data to disk and SSD. This increases the cost of writing data and the number of cache misses.

To isolate the impact of SSC consistency and silent eviction mechanism on FlashTier performance, we separately disabled persistent metadata for the write-back SSD cache. This configuration, which would lose data in a crash, is 8% faster than the baseline system. Thus, a portion of FlashTier's improved performance comes from the unified address space managing clean/dirty metadata within the cache. The remaining performance gain comes from FlashTier's silent eviction, which greatly reduces the cost of garbage collection.

Finally, we compare the memory usage of three different data structures for address translation within the SSC used to store forward and reverse mappings for a hybrid flash translation layer – sparse hash map (SHM), dense hash map (DHM, also from Google [4]) and a two-level page table (MPT). Similar to virtual memory management, FlashTier exploits sparseness in SSC address space incurred because of caching hot blocks from a much larger disk logical block address space. We find that SHM requires up to 70% less memory for storing the mappings as compared to MPT. We also measure the added value of sparseness by comparing against DHM. DHM is comparable to MPT for homes and mail, and up to 50% worse for web, because data is not dense enough to leverage DHM's memory representation.

References

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

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

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

[4] GOOGLE INC. Google Sparse Hash. `http://goog-sparsehash.sourceforge.net`.

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

[6] KOLLER, R., AND RANGASWAMI, R. I/o deduplication: Utilizing content similarity to improve i/o performance. In *FAST* (2010).

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

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

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