Multi-level Hybrid Cache: Impact and Feasibility

Zhe Zhang[*]*, Youngjae Kim*[*]*, Xiaosong Ma*[†*]*, Galen Shipman*[*]*, Yuanyuan Zhou*[‡]
[*]*Oak Ridge National Laboratory,* [†]*North Carolina State University,* [‡]*University of California San Diego*

Abstract

Storage class memories, including flash, has been attracting much attention as promising candidates fitting into in today's enterprise storage systems. In particular, since the cost and performance characteristics of flash are in-between those of DRAM and hard disks, it has been considered by many studies as an secondary caching layer underneath main memory cache. However, there has been a lack of studies of correlation and interdependency between DRAM and flash caching. This paper views this problem as a special form of multi-level caching, and tries to understand the benefits of this multi-level hybrid cache hierarchy. We reveal that significant costs could be saved by using Flash to reduce the size of DRAM cache, while maintaing the same performance. We also discuss design challenges of using flash in the caching hierarchy and present potential solutions.

1 Introduction

The past decade has witnessed significant advances in semi-conductor technologies and the emergence of various forms of storage class memory (SCM) such as NAND flash, magnetic RAM (MRAM), phase-change memory (PRAM), and Ferroelectric RAM (FeRAM). In particular, NAND flash memory has low unit price (in $/byte) due to its massive production. Therefore it has not only been a major storage media for mobile devices in recent years but also is becoming on the road to replace a certain portion of hard disks (HDD) currently used in desktop and server environments [12, 18].

Researchers have also proposed using SCM to extend and complement traditional DRAM-based main memory in different ways: to compose hybrid main memory [15], to form new virtual memory hierarchy [19], and as secondary buffer caches [8, 14]. Among them, SCM-based caching is particularly important in I/O intensive environments, where enterprise storage servers are provisioned with large DRAM buffer caches to improve I/O throughput and response time. A secondary SCM caching layer can reduce the need for DRAM cache, saving hardware costs and energy, or supplying additional cache capacity.

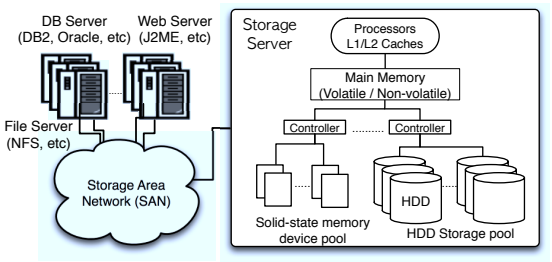


Figure 1: DRAM-based main memory and SCM form a multi-level hybrid cache hierarchy.

When both DRAM and SCM caches are used in a storage server, a multi-level hybrid cache hierarchy is formed, as illustrated in Figure 1. However, existing work has overlooked the coordinated management of DRAM and SCM caches in this hierarchy.

In fact, coordinated multi-level caching have been widely studied in distributed computing environments [3, 6, 22, 25, 24]. In this work we view the above problem as a special form of multi-level cache management, and revisit existing mechanisms based on the following unique characteristics of the DRAM-SCM-HDD hybrid storage hierarchy.

Firstly, in previously studied environments, multiple levels of caches are typically deployed in and managed by distributed computer systems connected with local or remote networks. SCM, on the other hand, often resides in the same node as, and has fast bus connection to, the main memory.

Moreover, the low unit price of SCM, especially flash, determines that its cache space can be an order of magnitude larger than DRAM capacity. This is different than in distributed multi-level caches, where a lower level often have comparable or even smaller cache space than an

upper level [3].

Finally, as a storage media, SCM possesses the following unique characteristics:

(i) SCM's write is noticeably worse than the read counterpart. (ii) SCM becomes less reliable after many repeated write-erase cycles. Depending on the media type, some SCM devices also suffer from degraded performance during garbage collections, which are performed when insufficient free blocks are available. (iii) The non-volatile property of SCM makes it an attractive option for buffering dirty data blocks longer.

Based on these observations, in this paper we make two major contributions.

- To understand the *impact* of multi-level hybrid caches, we analyze the amount of DRAM that each GB of SCM "saves" when used for demand paging, prefetching, and write buffering, respectively.
- To address the *challenges* of adopting this architecture, we propose several novel mechanisms as well as a number of design guidelines. Their purpose is to help the system approach the ideal saving mentioned above. For simplicity, the rest of our discussion will be based on flash. However, they can be extended to any type of SCM.

2 How much DRAM can be saved?

In this section, we analyze the multi-level caching problem in a hybrid storage system consisting of DRAM, flash memory, and hard disk drive. We consider *saving DRAM* as the primary impact of using flash in this I/O stack. Therefore, our basic approach is to solve the following equation:

$$Flash(F) = DRAM(D) \quad (1)$$

In Equation 1, F represents the size of flash memory being used, D denotes the amount of DRAM with equivalent impact to the performance. Intuitively, the equation indicates that we can save D amount of DRAM with F amount of flash. If we use $Cost_f$ and $Cost_d$ to represent the unit prices for flash and DRAM, then the $ saving of using F amount of flash is $Cost_d \cdot D - Cost_f \cdot F$.

2.1 Demand paging

Memory cache space can be used in three ways:

- Demand paging – keeping data block that are likely to be re-accessed in the future
- Prefetching – fetching data blocks before they are requested based on prediction of access patterns
- Write buffering – absorbing asynchronous write requests

In this subsection we present our analysis and preliminary results on demand paging. Section 2.2 extends the discussion to prefetching and write buffering.

In multi-level demand paging the role of the lower level cache is to catch misses from the upper level. For a given workload, let $h(x)$ denote the cache hit rate with cache size x. If we have a DRAM cache of size x_d and a flash cache of size x_f, then we have a first level hit rate of $h(x_d)$ and a second level hit rate of $h(x_f) - h(x_d)$. If we further assume the random read cost on DRAM, flash and HDD are r_d, r_f, r_h respectively, then the average I/O response time is:

$$h(x_d) \cdot r_d + (h(x_f) - h(x_d)) \cdot r_f + (1 - h(x_f)) \cdot r_h \quad (2)$$

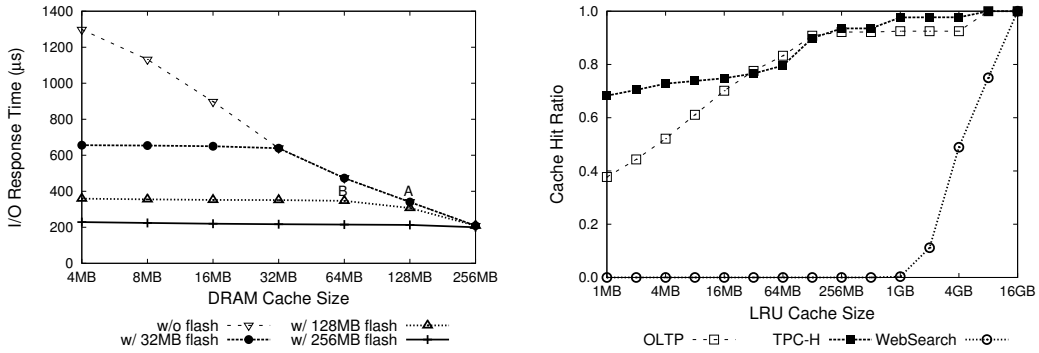
Having x_f amount of flash saves total access time by $save(x_f) = [h(x_f) - h(x_d)] \cdot (r_h - r_f)$. Therefore, when coupled with x_d amount of DRAM, then we can deduct the following equation:

$$Flash(F) = DRAM(h^{-1}(h(x_d) + \frac{save(F)}{r_h - r_d}) - x_d)$$
$$\text{, where } save(F) = [h(F) - h(x_d)] \cdot (r_h - r_f). \quad (3)$$

To empirically evaluate the saving we use a multi-level cache emulator extended from and cross-validated with a multi-level cache simulator used in [3, 25]. The emulator runs on the application level and bypasses the OS buffer cache by using direct and synchronous I/O. Real SATA-II SSD (Intel SSDSA2SH032G1GN) and HDD (WDC WD3200AAKS-75L9A0) are used in experiments. Our preliminary experiments use a simple $DRAM - Flash - HDD$ architecture, and adopt the LRU replacement policy on both caches. Three I/O traces are used: *OLTP* (collected from online transaction processing applications at a large financial institute) and *WebSearch* (collected from a popular search engine) are obtained from SPC [1], while *TPC-H* were collected at the Purdue university by running the corresponding benchmark. Prefetching is turned off and write requests in traces are ignored.

Figure 2.a shows the I/O response time of the OLTP trace with different DRAM and flash sizes. In this trace the average request size is 15KB, and the read access costs on DRAM, SSD and HDD are: $r_d = 1.56\mu s$, $r_f = 86\mu s$, and $r_h = 2.35ms$. From the figure, we see that without flash, the response time decreases almost linearly with larger DRAM cache. Adding a flash cache improves the performance significantly in most cases. Moreover, the response time is much less sensitive to DRAM cache size when flash is larger than DRAM. That is because even with a high aggregate cache hit ratio ($\sim 90\%$) the I/O response time is dominated by disk accesses. As long as flash is larger than DRAM, increasing the DRAM size will only transfer flash cache hits into DRAM cache hits, without reducing disk accesses.

[1] http://traces.cs.umass.edu/index.php/Storage/Storage



(a) (b)

Figure 2: (a) Average I/O response time of OLTP trace. (b) LRU cache hit ratio with different traces.

Due to the space limitation we do not show the response time results for the other two traces. However, in Figure 2.b we present their cache hit ratios on a single level LRU cache. This corresponds to the $h(x)$ function above, and enables the estimation of flash cache's impact using Equation 3. We see that in OLTP and TPC-H traces, almost 90% of requests can be hit in 256MB memory space. In contrast, WebSearch shows much less temporal locality than other workloads. 4GB memory space can barely satisfy 50% hits of requests.

$ Saving: We illustrate the actual $ saving by comparing two sample configurations: a 128MB DRAM cache and a hybrid cache of 64MB DRAM and 128MB flash (marked as "A" and "B" in Figure 2.a). Since they have almost the same response time, we have $Flash(128MB) = DRAM(64MB)$ (as in Equation 1). With a 5:1 price ratio of DRAM and flash ($/GB), the hybrid cache saves 59% over the single level DRAM cache. With a 10:1 ratio the saving becomes 79%.

2.2 Prefetching and Write Buffering

Prefetching: In modern systems the behavior of sequential prefetching is controlled by two parameters, *prefetch degree*, indicating how much data to prefetch for each prefetching request, and *trigger distance*, indicating how early to issue the next prefetch request. The memory consumption with a giving prefetching degree P and trigger distance T is roughly $T+\frac{P}{2}$ [24]. While prefetching mechanisms have different ways of setting T and P values, the main factors affecting these decisions are applications' data access rates and storage media's latency and throughput.

Since flash has smaller read latency and higher read throughput than HDD, with a given workload we have $P_f < P_h$ and $T_h < T_h$, where P_f, T_f, P_h, and T_h are prefetch degrees and trigger distances for flash and HDD, respectively. Therefore, with n_{sr} sequential reading streams, the following equation can be deducted:

$$Flash(n_{sr} \cdot (T_h + \frac{P_h}{2})) = DRAM(n_{sr} \cdot (T_h + \frac{P_h}{2} - T_f - \frac{P_f}{2}))$$

Write buffering: Write buffering is less studied than demand paging and prefetching because of its asynchronous nature. However, researchers have pointed out that the lack of careful management could lead to considerably degraded performance [2].

Current DRAM write buffering mechanisms are designed with several main considerations in mind. Firstly, multiple updates to the same page may be aggregated and performed once. Secondly, spatially related writes may be merged into large sequential requests which are favorable for hard disk performance.

In contrast to HDD, flash devices do not have rotating parts, and the main benefit of large sequential writes is to reduce the number of erases. Therefore, with a flash memory cache layer underneath, DRAM write buffering only needs to accumulate write requests up to sizes of a flash erasing unit f_pg (typically 128KB or 256KB).

However, it is still beneficial for DRAM to leverage temporal locality and catch re-writes to the same page. Otherwise the same page will be erased multiple times on the flash layer. Meanwhile, the flash memory layer could serve as a buffer region to re-organize dirty blocks into spatially sequential batches to optimize disk I/O.

If there are n_{sw} sequential writing streams, with an average rate of wr, and the system uses al_d as the age limit in flushing dirty pages in DRAM, al_f for flash, then we have the following equation:

$$Flash(n_{sw} \cdot wr \cdot al_f) = DRAM(n_{sw} \cdot dr \cdot al_d - f\_pg)$$

3 Approaching the ideal saving

It is not trivial to achieve the ideal savings as analyzed in Section 2. In this section we discuss major technical

challenges of multi-level hybrid caching and also propose possible mechanisms to overcome those challenges.

3.1 Challenges

In NAND flash memory devices, read and write operations are executed at the granularity of pages. Unlike in-place updates in HDD, an update on a flash page needs to mark this page as "invalid" and write the new content in a free page elsewhere. The device will eventually reach a state where too few free pages are remaining, at which time a process is triggered to collect and erase invalid pages marked by page updates and create free pages/blocks for future write operations. This process is called Garbage Collection (GC). Normal read and write operations will be slowed down during GC, to different extents depending on device types [11].

In addition, even without being disturbed by GC, write operations on flash take considerably longer time than reads (up to 10× latency). If occurring on the critical path such as synchronous reads, this could cause application slowdowns.

Moreover, the reliability and lifetime of flash devices could be severely degraded with frequent page updates. [17] has analyzed the case where flash is used as write buffers for storage volumes. It is pointed out that since all writes to a volume are absorbed by a relatively small flash device, the wear-out time is much shorter (less than 5 years with more than half of the workloads) compared to the scenario where flash is used as persistent storage (over 100 years for the majority of workloads).

We further argue that when used as a cache (for demand paging, prefetching and write buffering), flash faces page updates caused by *both write and read* requests from applications. As an extreme example, a read-only workload still introduces write operations to flash cache when new cache blocks are inserted and old ones are evicted. Putting it precisely, *every read/write request observed on the disk level corresponds to one cache content update*. Therefore, the total update/erase frequency of flash cache is the summation of disk read and write I/O rates.

Trace	Read (MB/s)	Write (MB/s)	Size
MSR Cambridge [16]	12.77	5.98	5.7TB
Exchange Server [17]	22.84	36.79	750GB

Table 1: Workload characteristics of block I/O traces.

In Table 1 we characterize representative disk block I/O traces collected from real workloads. For instance, with the MSR Cambridge trace, the write rate to flash cache is 213% compared to write rate perceived by the disk volume.

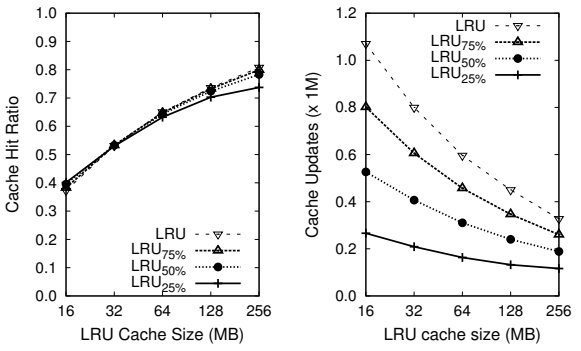
3.2 Possible solutions

To address the above challenges, existing single- and multi-level cache management mechanisms need to be revisited. In this section we discuss several new designs and present preliminary results.

Direct path between DRAM and HDD: Toward the goal of eliminating flash write operations from critical paths, we propose keeping a direct path between DRAM and HDD, instead of using a traditional tiered architecture assumed by most existing multi-level caching studies [3]. Doing so enables data blocks that are missing from both caches to be fetched directly from HDD to DRAM, and then to be written to flash in background. In addition, the system can select to bypass the flash cache layer for some requests.

Frequency-aware policies: We argue that new cache replacement policies should be designed and adopted that are aware of the cache content update frequency. As a first step we propose a simple algorithm LRU_f. It is a variation of LRU with a cache content update frequency that is only a fraction f of the original algorithm. This can be achieved in different ways, the simplest among which is to randomly discard some updates.

To evaluate this trade-off we have implemented LRU_f on the flash layer in the emulator introduced in Section 2.1, and performed experiments with the OLTP trace. The DRAM cache size is fixed at 4MB and flash cache size changes from 8MB to 256MB.



(a) (b)

Figure 3: (a) and (b) show flash cache hit ratio and number of content updates of LRU and different LRU_f schemes with the OLTP trace.

Figure 3 shows both the flash cache hit ratio and number of cache updates of of LRU_f with different f values. It can be observed that $LRU_{75\%}$ has very similar hit ratios as LRU, while incurring 25% fewer cache updates. $LRU_{25\%}$ reduces the number of updates much more aggressively, with the penalty of degraded hit ratios with relatively large cache sizes. It decreases the hit ratio by 4% for a 128MB cache and 9% for a 256MB one, translating 13% and 37% increases in the number

of disk reads, which dominates the response time as we discussed earlier.

4 Related Work

Multi-level Caching: There have been many studies on improving secondary cache hit ratio in multi-level demand paging. Existing work has been focused on providing exclusiveness [22, 4, 6] or capturing long-term access patterns [26]. A comprehensive empirical evaluation is provided in [3]. With these optimizations the savings with hybrid cache will be further improved. However, some techniques increases data transfers between upper and lower caches which need to be considered when applied to flash. Multi-level prefetching has also been studied recently [23, 24, 25]. However, it has not been empirically studied how to set appropriate prefetching aggressiveness on a very fast secondary cache such as locally attached flash, which we consider as our future work.

SSD storage and caching: Many studies have been conducted on the performance optimization of systems employing flash/SSD [8, 20, 13]. In [21] a hybrid storage device is proposed that can use a HDD as a write cache for a SSD, reducing over-writes. There are also ongoing efforts to design and develop adaptive file systems for heterogeneous media [10, 5].

There have been researches on SSD-aware caching [18] as well as using flash as a partial replacement of main memory/DRAM [8]. CFLRU [18] is an optimization of main memory caching mechanism with underlaying SSD-based persistent storage. In [9] a hybrid memory system is proposed where some portion of main memory is replaced with larger flash. However, they focused an architectural design issue rather than cache managements.

5 Conclusion

In this paper we discovered that a multi-level hybrid cache can have significant $ cost saving ($59\% \sim 79\%$) over a traditional DRAM cache by adding an additional flash layer.

We also found that flash memory has its limitations when used as a secondary cache, due to its garbage collection overheads, life time constraints and slow write operations, and proposed potential solutions.

References

[1] L. N. Bairavasundaram, M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. X-ray: A Non-Invasive Exclusive Caching Mechanism for RAIDs. *SIGARCH Comput. Archit. News*, 32(2):176, 2004.

[2] A. Batsakis, R. Burns, A. Kanevsky, J. Lentini, and T. Talpey. Awol: An Adaptive Write Optimizations Layer. *FAST*, 2008.

[3] Z. Chen, Y. Zhang, Y. Zhou, H. Scott, and B. Schiefer. Empirical Evaluation of Multi-level Buffer Cache Collaboration for Storage Systems. *SIGMETRICS*, 2005.

[4] Z. Chen, Y. Zhou, and K. Li. Eviction-based Cache Placement for Storage Caches. *USENIX ATC*, 2003.

[5] J. A. Garrison and A. L. N. Reddy. Umbrella File System: Storage Management across Heterogeneous Devices. *Trans. Storage*, 5(1):1–24, 2009.

[6] B. S. Gill. On Multi-level Exclusive Caching: Offline Optimality and Why Promotions Are Better than Demotions. *FAST*, 2008.

[7] S. Jiang and X. Zhang. ULC: A File Block Placement and Replacement Protocol to Effectively Exploit Hierarchical Locality in Multi-level Buffer Caches. *ICDCS*, 2004.

[8] T. Kgil and T. Mudge. Flashcache: A Nand Flash Memory File Cache for Low Power Web Servers. *CASES*, 2006.

[9] T. Kgil, D. Roberts, and T. Mudge. Improving Nand Flash based Disk Caches. *ISCA*, 2008.

[10] E. Kim, H. Shin, B. Jeon, S. Han, J. Jung, and Y. Won. Frash: Hierarchical File System for Fram and Flash. *LNCS*. 2007.

[11] Y. Kim, S. Oral, D. Dillow, F. Wang, D. Fuller, S. Poole, and G. Shipman. An Empirical Study of Redundant Array of Independent Solid-state Drives (RAIS). *ORNL/TM-2010/61, National Center for Computational Sciences*, March 2010.

[12] I. Koltsidas and S. D. Viglas. Flashing Up The Storage Layer. *Proc. VLDB Endow.*, 1(1):514–525, 2008.

[13] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas. Using Transparent Compression to Improve SSD-based I/O Caches. *EuroSys (To Appear)*, 2010.

[14] E. L. Miller, S. A. Brandt, and D. D. E. Long. Hermes: High-Performance Reliable MRAM-Enabled Storage. *HOTOS*, 2001.

[15] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi. Operating System Support for NVM+DRAM Hybrid Main Memory. *HotOS*, 2009.

[16] D. Narayanan, A. Donnelly, and A. Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. *Trans. Storage*, 4(3):1–23, 2008.

[17] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Enterprise Storage to SSDs: Analysis of Tradeoffs. *EuroSys*, 2009.

[18] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee. CFLRU: A Replacement Algorithm for Flash Memory. *CASES*, 2006.

[19] M. Saxena and M. M. Swift. Flashvm: Revisiting the Virtual Memory Hierarchy. *HotOS*, 2009.

[20] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending SSD Lifetimes with Disk-based Write Caches. *FAST*, 2010.

[21] D. L. Willick, D. L. Eager, and R. B. Bunt. Disk Cache Replacement Policies for Network Fileservers. *ICDCS*, 1993.

[22] T. M. Wong and J. Wilkes. My Cache or Yours? Making Storage More Exclusive. *USENIX ATC*, 2002.

[23] G. Yadgar, M. Factor, K. Li, and A. Schuster. MC2: Multiple Clients on A Multilevel Cache. *ICDCS*, 2008.

[24] Z. Zhang, A. Kulkarni, X. Ma, and Y. Zhou. Memory Resource Allocation for File System Prefetching: From A Supply Chain Management Perspective. *EuroSys*, 2009.

[25] Z. Zhang, K. Lee, X. Ma, and Y. Zhou. PFC: Transparent Optimization of Existing Prefetching Strategies for Multi-level Storage Systems. *ICDCS*, 2008.

[26] Y. Zhou, J. F. Philbin, and K. Li. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. *USENIX ATC* , 2001.