Cluster Comput (2018) 21:1275–1287
https://doi.org/10.1007/s10586-017-1076-0

HALO: a fast and durable disk write cache using phase change memory

Zhuo Liu[1] · **Bin Wang**[1] · **Weikuan Yu**[2]

Received: 3 September 2014 / Revised: 22 January 2015 / Accepted: 25 July 2017 / Published online: 4 August 2017

Abstract To close the increasing performance gap between disk storage and processors, flash based solid-state devices (FSSDs) have been adopted within the memory hierarchy to improve the performance of hard disk drive (HDD) based storage system. However, FSSDs still suffer from erase-before-write restriction, coarse access granularity and limited write endurance. Recently, the cutting-edge non-volatile memory technologies are merging, e.g., phase-change memory and resistive memory, which offer us new storage alternatives with faster access, byte-addressability and better endurance. In order to address the imperative data intensive computing issues, we propose to leverage PCM as disk write cache for constructing a hybrid PCM+HDD storage architecture in this paper. First, we develop a novel hash-based write caching scheme called *HALO* to improve both spatial and temporal locality on hard disks, thus addressing the limitations of traditional LRU caching algorithms and rendering better I/O performance. To deal with the limited durability of PCM devices and reclaim the degraded spatial locality in previous wear-leveling techniques, we further propose novel PCM wear leveling algorithms that provide effectively uniform writes while maximizing access parallelism. We have evaluated this PCM-based hybrid storage architecture using applications with a diverse set of I/O access patterns. Our experimental results demonstrate that the HALO caching scheme leads to an average reduction of 36.8% in execution time compared to the LRU caching scheme, and that the space filling curve based wear leveling extends the lifetime of PCM by a factor of 21.6.

Keywords PCM · Cache · Wear leveling · Hybrid storage · Cuckoo hashing

1 Introduction

In current *Big Data* era, advances in computing technologies have triggered a great explosion of digital information. Such gigantic amounts of data have further widen the speed gap between calculation and storage and brought about both performance and power consumption challenges.

One intuitive solution to such challenges is to introduce hybrid storage architectures which can combine the advantages of multiple types of storage devices. For example, flash-based hybrid storage drives (HSDs) have been proposed to integrate standard hard disk drives (HDDs) and flash-based solid-state drives (FSSDs) into a single storage enclosure [1]. In doing so, Flash-based HSDs deliver fast random access by combining low-power, fast-access FSSDs and low-cost, high-capacity HDDs together to end users. Although flash-based HSDs are gaining popularity in personal computers, they suffer from several striking shortcomings of FSSDs, namely high write latency and low endurance, which seriously hinder the successful integration of FSSDs into HSDs, especially in data intensive server scenarios. A series of techniques have been proposed to address the issues [34,36]. However, most of them only target specific usage scenarios and cannot act as a general solution to eliminate FSSDs' drawbacks, which continue to threaten the future success of

✉ Weikuan Yu
yuw@cs.fsu.edu

Zhuo Liu
zhuoliu@auburn.edu

Bin Wang
bwang@auburn.edu

[1] Auburn University, Auburn, AL, USA

[2] Florida State University, Tallahassee, FL, USA

FSSD-based HSDs. There exists a continuing need of better technologies in the storage market.

Fortunately, new cutting-edge non-volatile random-access memory (NVRAM) devices are emerging, such as phase-change memory (PCM), spin-torque transfer memory (STTRAM), and resistive RAM (RRAM). These memory devices provision a rich set of advanced features, for example, they support similar non-volatility as conventional HDDs and FSSDs while providing speeds approaching those of DRAMs. Among these technologies, PCM is particularly promising, with several companies and universities already providing prototype chips and devices [2,22]. Compared to FSSD, PCM is equipped with a number of performance and energy advantages [6]. First, PCM has much lower read response time than FSSD. It offers a read response time of around $50ns$, nearly 500 times faster than that of FSSD. Second, PCM can overwrite data directly on the memory cell, in contrast to FSSD's erase-before-write. The write response time of PCM is less than $1\mu s$, nearly three orders of magnitude faster than that of FSSD. Third, the program energy for PCM is 6 Joules/GB, 3 times smaller than that of FSSD [6]. Thus, PCM is a viable alternative to FSSDs for building hybrid storage systems.

In this paper, we design a novel hybrid storage system that leverages PCM as a write cache to merge random write requests and improve access locality for HDDs. It includes new caching and destaging algorithms for better I/O performance as well as wear leveling algorithms for enhanced PCM life time.

A rich set of techniques have used NVRAM as the data cache to improve disk I/O [3,9,11,17,38]. Most of them use LRU-like methods (e.g., Least Recently Written, LRW [9,11]) to manage small size non-volatile cache to improve performance and reliability of HDD-based storage and file system. However, the fine-grained LRW for large PCM cache can cause big mapping overheads while CSCAN introduced in [11] has a O(log(n)) insertion speed makes it not suitable for relatively large PCM cache. Specifically, for high density PCM cache with capacity of GBs, using LRU to manage them will cause big DRAM overheads in managing the LRU stack and mapping. In addition, LRU/LRW cannot ensure that destaging I/O traffic be presented as sequential writes to hard disks. Some methods (e.g., CSCAN [11]) as a supplement for LRW can ease this issue to some extent but it requires O(log(n)) time for insertion, making it not suitable for large size cache management.

Therefore, it is crucial to rethink the current cache management strategies for PCM. In this paper, we propose a novel cache management algorithm, named HALO which uses a chained hash table to manage PCM, merge random write requests, and improve data access locality to hard disks. It manages address mapping through cuckoo hash tables and implements a new two-way destaging policy. These techniques together save DRAM overheads significantly while maintaining constant speeds for both insertion and query. Also, HALO is very beneficial in terms of managing caching items, merging random write requests, and improving data access locality to hard disks. In addition, by removing the dirty-page write-back deadline limitations that commonly exist in DRAM-based caching systems, HALO enables better write caching and destaging, and thus achieves better I/O performance. And by storing cache mapping information on non-volatile PCM, the storage system is able to recover quickly and maintain integrity in case of system crashes.

To use PCM as a write cache, we must address PCM's limited durability. Several existing wear-leveling techniques have shown good endurance improvement for PCM-based memory systems [28,29,44]. However, these techniques are not specifically designed for PCM used in storage and file systems, and thus can negatively impact spatial locality of file system accesses, which in turn degrade read-ahead and sequential access performance of file systems. We propose two new wear leveling techniques: rank-bank round robin wear-leveling and space filling curve wear-leveling. In the space filling curve wear-leveling, we not only provide a good write balance among different regions of the device, but also keep data locality and enable good adaptation to the file system's I/O access characteristics.

For assessing the effectiveness of our hybrid storage system, we conduct extensive experiments for a diverse set of real-world I/O workloads. By thoroughly studying the system's I/O performance and endurance characteristics compared to traditional techniques, HALO is demonstrated to be a fast and durable PCM write cache in HDD-based storage system. To be specific, HALO leads to an average reduction of 36.8% compared to LRU caching scheme in terms of execution time; in addition, the SFC wear leveling extends the lifetime of PCM device by a factor 21.6. This paper is a substantial extension upon our preliminary study presented at [20]. It further includes the rank-bank round-robin wear leveling algorithm for comparisons and the two-way destaging for better front-end service performance. In addition, we have provided more comprehensive evaluation results such as the HDD throughput, response time, detailed write access counts, bank deviation and memory overheads of mappings.

The rest of the paper is organized as follows. We first describe the design of a hybrid PCM-HDD architecture in Sect. 2, followed by Sect. 3 where the details of our wear-leveling algorithms are provided. Section 4 then provides experimental methodology and results. Section 5 provides a brief overview of related work. Finally, we conclude the paper in Sect. 6.

2 PCM based hybrid storage

Hybrid storage devices have been constructed in many different ways. Most HSDs are built using flash-based solid state devices as either a non-volatile cache or a prefetch buffer inside the hard drives. The combination of FSSDs and HDDs offers an economic advantage with low-cost components and the mass production. This composition of hybrid storage devices, as shown in Fig. 1a, is currently the most popular.

Exploring emerging NVRAM devices such as PCM as components in hybrid storage devices has attracted significant research interests. Research in this direction proceeds along two distinct paths. Along the first path, PCM is used as a direct replacement for FSSDs, as shown in Fig. 1b. Along the second path, PCM is used in combination with FSSDs to compensate FSSDs' lack of in-place updating, and possibly push HDDs out of hybrid storage devices, as shown in Fig. 1c. For example, Sun et al. [36] use this type of hybrid storage devices to demonstrate its capability of high performance and increased endurance with low energy consumption. However, there are two major problems associated with this approach. First, since FSSDs provide primary data storage space, the erase-before-write problem still exists, although it happens at lower frequency. This causes significant performance loss for data intensive applications. Second, without HDDs in the memory hierarchy, large volumes of storage space cannot be leveraged at reasonable performance costs. In terms of cost per gigabyte, FSSDs are still about 10 to 20 times more expensive than HDDs.

For the above reasons, we investigate the benefits of leveraging PCM as a write cache for hybrid storage devices that are designed along the first path. As shown in Fig. 1b, we use PCMs to completely replace FSSDs while retaining HDDs for their advantages in storage capacity. With the fast development of PCM technologies, we expect that the PCM-based hybrid storage drive will become more popular. In this section, we describe our hybrid storage system - HALO that uses PCM as a write cache for HDDs to improve performance and reliability of HDD-based storage and file systems. Specifically, we first introduce the HALO framework and its basic supportive data structures, and then elaborate on the caching and destaging algorithms.

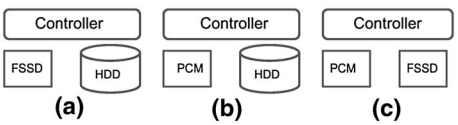


Fig. 1 Different architectures of hybrid storage devices

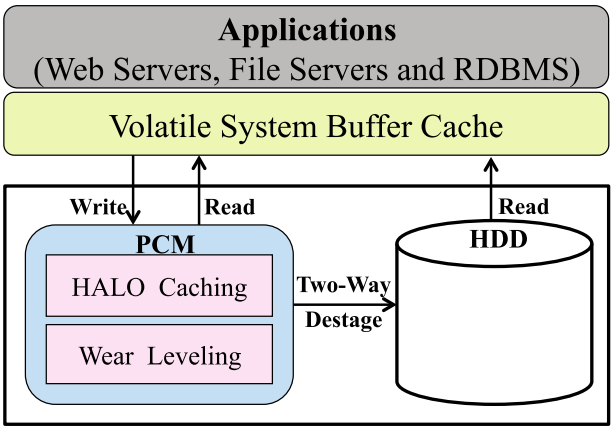


Fig. 2 Design of the HALO framework

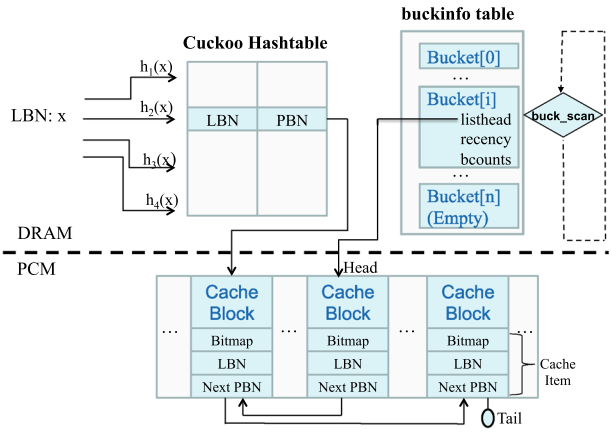


Fig. 3 Data structures of HALO caching

2.1 HALO framework and data structures

Using PCM as caches for HDDs demands efficient caching algorithms. We design a new caching algorithm, referred to as **HALO**, to manage data blocks that are cached in PCM for hard disk drives. HALO is a non-volatile caching algorithm which uses a **HA**sh table to manage PCM and merge random write requests, thereby improving access **LO**cality for HDDs. Figure 2 shows the HALO framework. The basic data structure of HALO is a chained hash table used to maintain the mapping of HDD's LBNs (Logical Block Number) to PCM's PBNs (PCM block addresses). Sequential regions on HDDs, in units of 1MB, are managed by one hash bucket. The information associated with sequential regions is used to make cache replacement decisions.

2.1.1 Mapping management

As shown in Fig. 3, the chained hashtable includes an in-DRAM array (i.e., the bucketinfo table) and on-PCM mapping structures. Another *CuckooHashtable* enables

space-efficient fast query. The bucketinfo table stores information for HDD data regions. Each bucket item in the table represents a 1 MB region on the disk partition or logical volume. Hence, the number of buckets in the bucketinfo table is determined by the size of the disk volume. Each bucketinfo item, if activated, contains three components: *listhead*, *bcounts*, and *recency*. *listhead* maintains the head block's PBN of a list of cache items that map to the same sequential 1 MB disk area, *bcounts* represents the number of caching blocks, and *recency* records the latest access time-stamps for all cache items in this bucket. We use a global request counter *globalReqClock* to represent the time-stamp; whenever a request arrives, the counter increases by one. The *total_counts* variable records how many HDD blocks have been cached inside PCM, while *activated_bucks* indicates the number of bucketinfo items activated in the bucketinfo table. *buck_scan* is used to search the bucketinfo table for a candidate destaging bucket.

Cache items that are associated with a bucket item do not need to be linked in ascending order of LBNs, because they are only accessed in groups during destaging. Each newly inserted item will be linked to the head of the list. This guarantees insertions to be finished in constant time. Each cache item maintains a 4 KB mapping from HDD block address (LBN) to PCM block number (PBN). It contains a LBN (the starting LBN of 8 sequential HDD blocks), the PBN of the next PCM block in the list and an 8-bit bitmap which represents the fragmentation inside a 4KB PCM block. If the 8-bit bitmap is nonzero, the nonzero bits represent cached 512B HDD blocks. Each cache item is stored on each PCM block's meta data section [2].

2.1.2 Cuckoo hash table

To achieve fast retrieval of HDD blocks, a DRAM-based cuckoo hash table is maintained using the LBN as the key and the PBN as the value. On a cache hit, the PBN of the cache item is returned, which enables fast access of data information in the PCM. Traditionally, hash tables resolve collisions through linear probing or chained hash and they can answer lookup queries within O(1) time when their load factors are very low, i.e., smaller than $log(n)/n$, where n is the table size. With an increasing load factor, its query time can degrade to O(log(n)) or even O(n). Cuckoo hashing solves the issue by using multiple hash functions [10,26]. It can achieve fast lookups within O(1) time (albeit a bigger constant than linear hashing), as well as good space efficiency i.e., high load factor. Next, we introduce how we achieve such space efficiency.

We set 100 as the maximum displacement threshold in the Cuckoo hashtable. When the Cuckoo hashtable cannot find an available slot for a new inserting record within 100 item displacements, it indicates that the hashtable is almost full and requires a larger size table and rehashing. And such critical load factor before rehashing is counted as maximum load factor. Figure 4 shows how the number of hash functions influences the average maximum load factor we can achieve for running seven traces. When the number of functions is two, only 50% load factor can be achieved; as the number of functions increases, the load factor of Cuckoo hash tables initially grows rapidly and then the slope of the curve becomes smaller thus the benefit achieved becomes marginal. Therefore we use four functions in our design since a larger number of functions can in turn bring higher query and computation overheads. In addition, we set a larger initial size of hashtable to keep the load factor lower than 80% when PCM cache is fully loaded. In this way we are able to maintain the average displacements per insert below 2 for better performance.

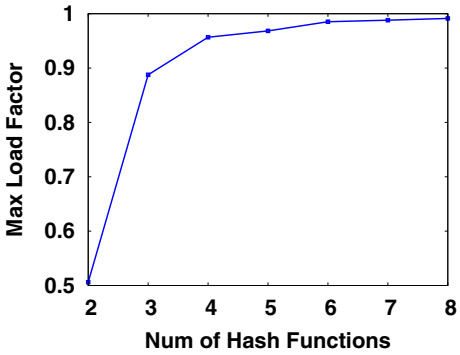


Fig. 4 Max load factors for different numbers of hash functions in Cuckoo hashing

Here, we give a sample calculation of DRAM overhead by the HALO data structures. In total, for a 2 GB PCM cache with 4 KB cache block size, 0.5 M items will be placed in the hashtable. As each item takes 8 Bytes and the load factor of the cuckoo hashtable is 0.8, the total memory overhead of the Cuckoo hashtable is about 5 MB. With the bucketinfo table normally consuming about 6–12 MB DRAM, we need less than 20 MB DRAM to implement HALO cache management for 2 GB PCM and 1 TB hard disk.

2.1.3 Recovery from system crashes

Mapping information of a PCM block that contains the LBN, the next PBN and the bitmap are stored on non-volatile PCM. Therefore, in case the system crashes, it can first reboot and then either destage the dirty items from PCM to HDD or rebuild the in-DRAM hashtables by scanning information on fixed positions of the PCM meta data sections (to get the cache items' information including LBN, bitmap and next PBN). As the PCM's read performance is similar to that of DRAM, the recovery procedure should only take seconds to rebuild the in-memory mapping data structures. In doing so,

we can avoid loss of cached data and guarantee the system integrity.

2.2 HALO caching scheme

Our caching algorithm is described in Algorithm 1. When a request arrives, the bucket index is computed using the request's LBN. The hash table is then searched for an entry corresponding to the LBN. In the event of a cache hit, the PBNs are returned from the hash table and the corresponding blocks are either written in-place to, or read from, the PCM. The corresponding bucket's recency in the bucketinfo table is also updated to the current time-stamp. In the event of a cache miss on a read request, data is read directly from the HDD without updating the cache. In the event of a cache miss on a write request, a cache item is allocated in the PCM, and data is written to that cache block. Then, if the bucket item of the bucketinfo table for the LBN is empty, it will be activated. After that, the bucket item's list of cache items is updated, the address mapping information is added to the hash table, the recency of this bucket is set to the current time-stamp, and the bucket's $bcounts$ is incremented. The updated access statistic information are used by the two-way destaging algorithm to conduct destaging procedures.

Algorithm 1 Cache management algorithm

```
1: Compute the bucket index i from the LBN
2: if this is a write request then
3:    Search the cuckoo hashtable using the LBN
4:    if this is a cache hit then
5:       Write to the PCM block with returned PBN
6:       Bucket[i].recency ← globalReqClock
7:    else
8:       //Cache miss
9:       Allocate and write a PCM block
10:      if Bucket[i] is empty then
11:         Activate Bucket[i]
12:         activate_bucks ← activate_bucks + 1.
13:      end if
14:      Link item to Bucket[i].listhead, add to cuckoo hashtable
15:      Bucket[i].recency ← globalReqClock
16:      Bucket[i].bcounts ← Bucket[i].bcounts + 1
17:      total_bcounts ← total_bcounts + 1
18:   end if
19: else
20:   //This is a read request
21:   Search the cuckoo hashtable.
22:   if cache hit then
23:      Read the PCM block with the returned PBN.
24:      Bucket[i].recency ← globalReqClock.
25:   else
26:      //Cache miss
27:      Read the block from HDD.
28:   end if
29: end if
```

2.3 Two-way destaging

Since the capacity of PCM cache is limited, we have to destage some dirty data from PCM to HDD in order to spare cache space for accommodating new requests. Therefore, we propose a Two-Way Destaging approach to achieve this target. The Two-Way Destaging approach contains two types of destaging: on-demand destaging and background destaging, which are activated to evict some data buckets out of PCM. Next, we will introduce when to trigger each destaging and how to select the victim buckets.

The on-demand destaging is activated when the utilization of PCM cache reaches a high percentage, e.g, 95% of the total size. Such on-demand method can sometimes incur additional wait delay to front-end I/O requests, especially when the I/O load intensity is high. To complement this approach and relax such contention, we introduce another destaging method which is triggered when both the PCM utilization is relatively high, e.g., 80%, and the front-end I/O intensity is low (specifically, disk performance utilization smaller than 10%). Through combination of these ways of destaging, PCM space can be appropriately reclaimed with minimal performance impacts to front-end workloads.

For either destaging method, a bucket is eligible to be destaged to HDDs if any of the following two conditions holds: First, the bucket's $bcounts$ needs to be greater than the average value of $bcounts$ plus a constant threshold $TH_{BCOUNTS}$ and the bucket's $recency$ needs to be older than $globalReqClock$ by a constant $TH_{RECENCY}$. For every unsuccessful round of scan, these two thresholds will dynamically decrease to make sure that victim buckets can be found within a reasonable number of steps. Second, the bucket's $recency$ needs to be older than $globalReqClock$ by a constant $OD_{RECENCY}$ ($OD_{RECENCY} \gg TH_{RECENCY}$). As soon as a bucket is identified as eligible for destaging, all cache blocks associated with the bucket are destaged to the HDD in a batching manner, the bucket is deactivated and the corresponding items in the cuckoo hash table are deleted. As these cache blocks are mapped to 1MB sequential region of HDD, this batch of write-backs are supposed to only incur one single seek operation to HDD, thus providing good write locality and causing minimal affects to read requests.

We select these two criteria for determining destaging candidates for the following reasons. First, we want to choose a bucket that has enough items to form a large enough sequential write to the HDD to increase spatial locality of write operations, and at the same time it needs to be one that is not recently used in order to preserve temporal locality. Second, for those very old and small buckets, we evict them from the PCM by setting the control variable $OD_{RECENCY}$.

Table 1 Parameters used for wear leveling

LSN	Global stripe number (0–64 K)
Blk	Offset of blocks in a bank
Seq	Sequence number in a SFC cube
$Cube$	Cube number (0–31)
S_{stripe}	Number of blocks in a stripe (64)
S_{cube}	Number of stripes in a cube (2048)
N_{cubes}	Number of cubes (32)
N_{ranks}	Number of ranks in a PCM (8)
N_{banks}	Number of banks in a rank (16)
$OS_{inStripe}$	Offset of blocks in a stripe
OS_{inCube}	Offset of stripes in a cube
OS_{inBank}	Offset of stripes in a bank
(R, B, S)	Rank, bank, stripe

3 Wear leveling

Although the write-endurance of PCM is 3–4 orders of magnitude better than that of FSSDs, it is still worse than that of traditional HDDs. When used as storage, excessively unbalanced wearing of PCM cells must be prevented to extend its lifetime. A popular PCM wear leveling technique [28] avoids frequent write requests to the same regions by shifting cache lines and spreads requests through randomization at the granularity of cache lines (256 B). This technique is feasible when PCM is used as a part of main memory; however, when PCM is used as a cache for back-end storage, this technique can negatively impact spatial locality of file system requests that are normally several KBytes or MBytes in size. In addition, the use of Feistel network or invertible binary matrix for address randomization requires extra hardware to achieve fast transformation. To address these issues, we propose two wear leveling algorithms for PCM in hybrid devices, namely *rank-bank round-robin* and *space filling curve (SFC)-based wear leveling*. Instead of using 256-Byte cache lines or single bits as wear leveling units, our algorithms use stripes (32 KB each). Such bigger units can significantly reduce the number of data movements in wear leveling. In addition, with the fast access time of PCM devices, the time to move a 32 KB stripe is quite small (less than 0.1 ms). Hence, the data movement overhead will not affect the response times of front-end requests. The important parameters for our algorithms are listed in Table 1.

3.1 Rank-bank round-robin wear leveling

The rank-bank round-robin wear leveling technique is inspired by the RAID architecture. It adopts a similar round-robin procedure to distribute address space among PCM memory ranks and banks for achieving uniformity in inter-region write traffic. We firstly apply block striping over PCM devices in order to ensure an even distribution of writes at the rank and bank granularity. This scheme iteratively distributes data first over ranks, and then over banks within the same rank. Similarly, consecutive writes to the same rank are distributed over the banks within that rank. This scheme is shown in Fig. 5. Aside from assuring a good write distribution between ranks and banks, the proposed scheme also takes full advantage of parallel access to all ranks in the PCM. This means that writing N_{rank} blocks of data at the same time is possible, where N_{rank} represents the number of ranks in a particular device. This parallel access translates into improved response times, which is important for data-intensive applications. After block striping, we apply a start-gap rotation scheme inside each bank similar to the method in [28], but different in terms of the size of data units (i.e., in stripes of 32 KB rather than cache lines of 256 B for better spatial locality and less frequent rotations). We illustrate the calculation of LSN, rank index, bank index and logical stripe offset for the address mapping of Rank-Bank round-robin wear leveling in Eq. (1).

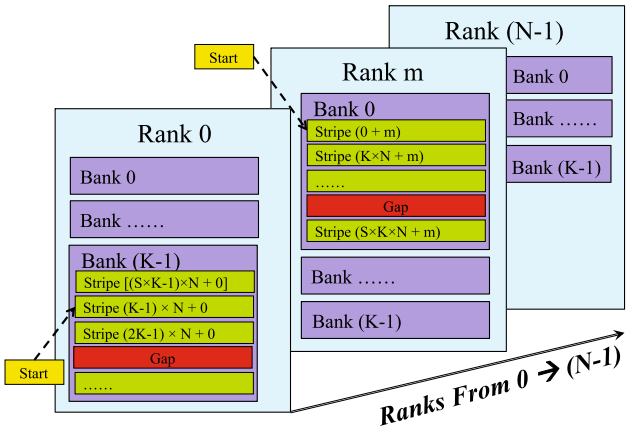


Fig. 5 Rank-bank round-robin wear leveling

$$\begin{aligned} LSN &= \left\lfloor \frac{PBN}{S_{stripe}} \right\rfloor \\ Rank &= LSN \bmod N_{ranks} \\ Bank &= LSN \bmod N_{banks} \\ OS_{inBank} &= \left\lfloor \frac{LSN}{N_{ranks}} \times N_{banks} \right\rfloor \end{aligned} \tag{1}$$

3.2 Space fill curve based wear leveling

The rank-bank round robin wear leveling algorithm can achieve even distribution among all banks and ranks for most cases as described later in Sect. 4. However, under certain workloads, a few intensively accessed banks still reduce lifetime of the PCM device. To solve this problem, we propose

using the Hilbert space filling curve (SFC) to further improve wear leveling. SFCs are mathematical curves whose domain spans across a multidimensional geometric space in a balanced manner [19].

In theory, there are an infinite number of possibilities to map one-dimensional points to multi-dimensional ones, but what makes SFCs suitable in our case is the fact that the mapping schemes of SFCs maintain the locality of data. In particular, points whose 1D indices are close together are mapped to indices of higher dimensional spaces that are still close. In our case, the LBN sequence is represented by the 1D order of points. The 3D space, into which the LBN sequence is mapped, is constructed with a tuple of three elements along the stripe dimension (the offset of stripes in a bank), the bank dimension (the offset of banks in a rank), and the rank dimension (the offset of ranks in a device).

$$\begin{aligned} \text{LSN} &= \left\lfloor \frac{\text{PBN}}{S_{stripe}} \right\rfloor \\ OS_{\text{inStripe}} &= PBN \bmod S_{stripe} \\ Cube_{\text{no}} &= Stripe \bmod N_{cubes} \\ OS_{inCube} &= \left\lfloor \frac{Stripe}{N_{cubes}} \right\rfloor \\ Seq &= StartGapMap(O\,S_{inCube}) \\ (R, B, S) &= SFCMapFunc(Seq) \end{aligned} \tag{2}$$

We have 512 stripes in a bank, 16 banks in a rank and 8 ranks in a device. We evenly split the 3D space into 32 cubes along the stripe dimension. In other words, the number of stripes in each cube is 16 × 16 × 8 (i.e., #stripe × #bank × #rank). After splitting, we apply the round-robin method to distribute accesses across these cubes. And inside every cube, a start-gap like stripe shifting is implemented, making the 3D SFC cube move like a snake. The consequence is that consecutive writes in the same cube can only happen for addresses that are 32 stripes away, which dramatically reduces the possibility of intensive writing in the same region. Within each cube, we apply SFC to further disperse accesses. We orchestrate SFC to disperse accesses across ranks as much as possible. This helps exploit parallelism from the hardware.

In summary, using SFC in combination with the round-robin method, we are able to map a 1D sequence of block numbers into a 3D triple of stripe number, rank number and bank number. The address mapping scheme is generally depicted in Fig. 6. The left figure shows the logical organization of the device with its 32 cubes (or parallelepiped's, to be more precise, because the size is 8 × 16 × 16). The right figure shows a 3-dimensional space filling curve that is used in our work. The mapping scheme starts with PBN provided by the system and ends up with a 3-tuple (R, B, S) calculated based on Eq. (2). The SFC based wear leveling is designed for a best trade-off among write uniformity, access parallelism and spatial locality.

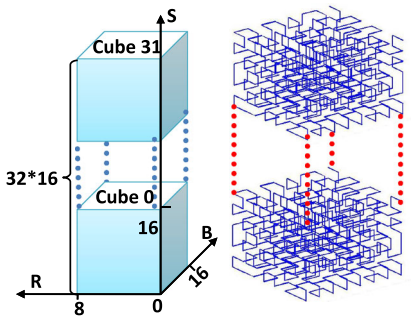


Fig. 6 Space filling curve based wear leveling

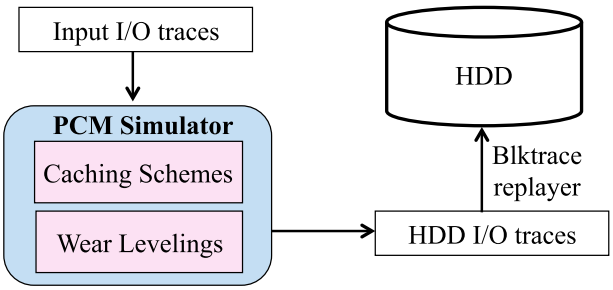


Fig. 7 PCM simulation and trace replay

4 Experimental evaluations

In this section, we first introduce the evaluation methodology and then demonstrate the experimental results in terms of I/O performance and PCM endurance.

4.1 Evaluation methodology

4.1.1 Experimental setup

To realize our proposed PCM-based write cache for hybrid storage devices we have designed a PCM simulation framework that simulates different caching schemes (HALO and LRU), wear leveling algorithms and PCM devices' characteristics including hardware structure, performance and wearing status. As we can see from Fig. 7, during evaluation, the block-level I/O traces are input to the simulators. The I/O requests are then processed by caching and wear leveling schemes, which generate two types of intermediate I/O requests: PCM requests and HDD requests. The PCM requests are processed by the PCM simulator to get response and wear leveling results. The HDD requests come from cache misses and destaging, which are stored as HDD trace files. HDD trace files are then replayed by the blktrace tool [4] on a 500 GB, 7200RPM Seagate disk in a CentOS 5 Linux 2.6.32 system with an Intel E4400 CPU and 2.0 GB memory. The DRAM-based system buffer cache is bypassed by the HDD trace replaying process. Traces are replayed in a close-loop way for measuring system service rate.

Table 2 Workload statistics

	Fin1	Fin2	Dap	Exchange	TPC-E	Mail	Randw
Write ratio	84.60%	21.50%	54.90%	74%	99.8%	90.10%	100%
Dataset (GB)	18.03	8.85	84.2	163.8	13.2	85	5.9
AvgReqSize (KB)	3.38	2.4	77	13.65	10.48	4	4

4.1.2 Metrics

Because PCM devices have much higher (more than 10 times) throughput rates and response performance than those of HDDs [2], we reasonably assume that the total execution time of a workload trace is dominated by the replay time of the HDD trace. For example, if the HDD traffic rate is about 10% and the write throughput is about 50 MB/sec, then the PCM cache must have a throughput rate of about 500 MB/sec, which is consistent with the reported performance of current PCM devices [2].

Based on the above discussion, the workload execution time can be calculated as follows: $(Total\_IO\_Size * Traffic\_Rate/Average\_Throughput)$. The traffic rate is calculated as the total number of accessed disk sectors (after the PCM cache's filtering) divided by the total number of requested sectors in the original workloads. This metric is similar to the cache miss rate. The lower the traffic rate we can achieve, the better the cache scheme performs. In order to achieve shorter execution times and better I/O performance, we must minimize the traffic rate and at the same time maximize the average HDD throughput. According to our tests, a standard hard disk can achieve as high as 100 MB/sec of throughput for sequential workloads but can only achieve 0.5 MB/sec for workloads with small random requests. We will evaluate whether the HALO caching scheme can reduce the HDD traffic rate while maximizing average throughput of a hard disk by reducing the inter-request seek distance among all disk writes.

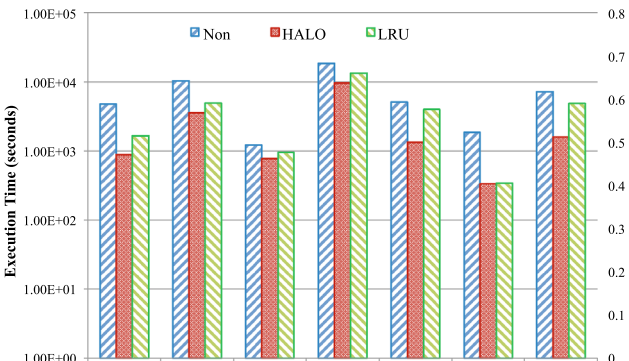
To evaluate wear leveling techniques, we define the PCM life ratio metric, which is calculated by dividing the achieved lifetime with the maximum lifetime. The life ratio is significantly affected by the uniformity of write requests. For example, if all write traffic goes to 1% of the PCM area, the life ratio can be reduced to 1% of the maximum life time. The life ratio is directly determined by the region with the maximum write count if there are no over-provisioning regions provided by the device.

4.1.3 Workloads

In our tests, we use seven representative real-world I/O traces. The workload statistics are described in Table 2. Specifically, the traces *Fin1* and *Fin2* were collected with the SPC-1 benchmark suite at a large financial organization [37]; the trace *Dap* was collected at a Display Advertisement Platform's payload server; the trace *Exchange* was collected at a Microsoft Exchange 2007 mail server for 5000 corporate users; the trace *TPC-E* was collected on a storage system of 12 28-disk RAID-0 arrays under an OLTP benchmark, TPC-E [35]; the trace *Mail* was collected on a mail server by Florida International University [35]. The seventh trace *Randw* was collected by us on the target disk while running the IOmeter benchmark [25] with the 4 KB-100% random-100% write workload for 120 min.

4.2 I/O performance

To evaluate the execution times of aforementioned seven traces, we choose 512 MB as the cache size for Fin1 and Fin2, and 2GB as the cache size for the other five traces. Figure 8 shows the results for Non-cache, HALO-cache and LRU-cache respectively. As we can see, the execution times are reduced greatly for all traces. The execution improvement of HALO caching over LRU caching is 47.11% for Fin1, 27.48% for Fin2, 18.97% for Dap, 28.27% for Exchange, 66.10% for TPC-E, 2.65% for Mail, and 67.11% for Randw. The improvement level is mainly determined by the randomness and write ratio of the traces. Note that for the Mail trace, the improvement is only 2.65%. The reason is that most of write requests in the Mail trace are sequential requests, for which there is not much room for HALO caching to improve on LRU caching. As introduced in Sect. 4.1, the improve-



	Fin1	Fin2	Dap	Exchange	TPCE	Mail	Randw
Non	4821	10273	1219	18501	5124	1855	7206
HALO	879	3579	769	9560	1345	330	1592
LRU	1662	4935	949	13328	3968	339	4840
Improve%	47.11%	27.48%	18.97%	28.27%	66.10%	2.65%	67.11%

Fig. 8 Execution time

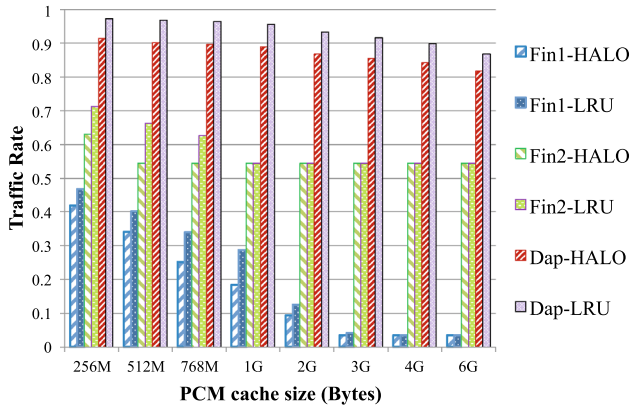


Fig. 9 Traffic rate for Fin1, Fin2 and Dap

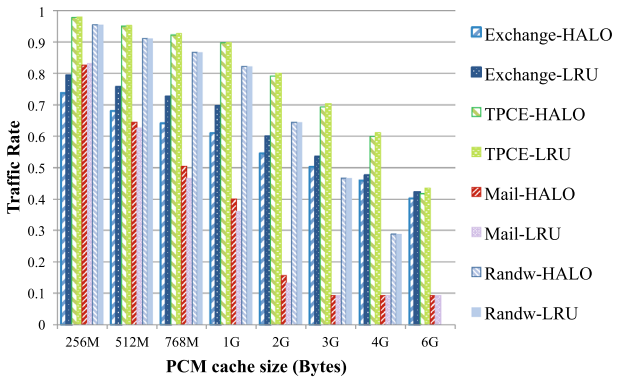


Fig. 10 Traffic rate for exchange, TPC-E, mail and Randw

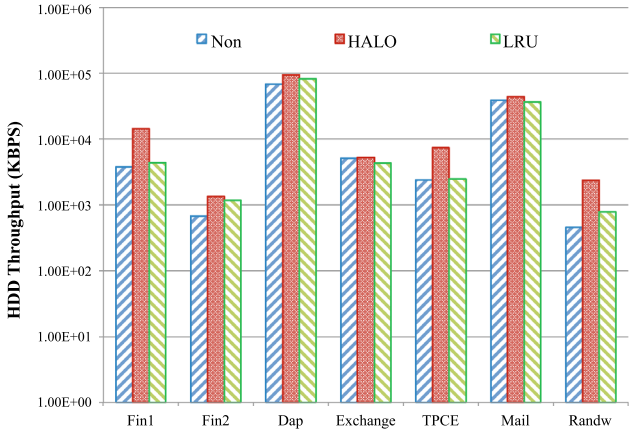


Fig. 11 HDD throughput rate and HALO's improvement over LRU

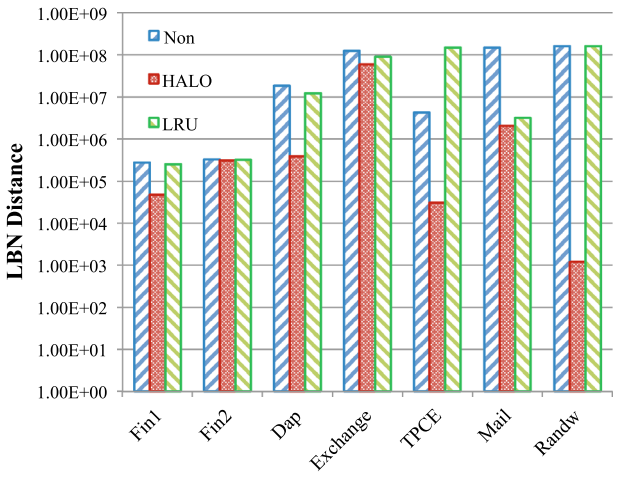


Fig. 12 Average inter-request LBN distance

ment of execution time comes from two aspects: first, the reduction of traffic rate because of better cache hit; second, the increasing on HDD average throughput due to improved write access sequentiality.

Figures 9 and 10 show the traffic rates for the seven traces with the HALO cache policy and the LRU cache policy, respectively. In most cases (with the cache size ranging from 256 MB to 6 GB), HALO consistently achieves 5–10% lower traffic rates than LRU. Take the trace Fin1 for example, HALO achieves 5, 6, 9, 10.3, 3, 1% lower traffic rates than LRU, where the cache size varies from 256 MB to 3 GB. We observe similar results for Exchange and TPC-E. For Dap, because the access repeatability and temporal locality is very poor, the traffic rate for HALO and LRU remains relatively high. However, HALO still gets a 6% lower traffic rate than LRU. For Randw, as it has a completely random write access pattern, all cache schemes can get almost no cache hits. That explains why the traffic rate of HALO and LRU are almost identical. Yet, HALO can still bring significant performance improvement in terms of execution time because of improved write access locality. For Fin2, as the cache size becomes larger, the PCM write cache consistently reduces more traffic until it reaches 768 MB. This is due to Fin2's relatively high read ratio (84.60%) and thus the opportunities for optimizing write operations are limited. For Mail, HALO's improvement is more insignificant for larger cache sizes. However, for small cache sizes (512 MB–2 GB), the traffic rate under LRU caching is about 2% less than that under HALO caching, because most write requests are already in a uniformly sequential pattern, so our cache scheme—which is targeted at random-write workloads—cannot show good improvement.

We use the average inter-request LBN distance as a metric to evaluate the IO access sequentiality to HDD, and the results are shown in Fig. 12. We notice that the average inter-request LBN distance is reduced greatly by HALO caching for almost all traces. This explains why the average disk throughput with HALO is much larger than with Non-cache and with LRU, as shown in Figs. 11 and 12. However, for Fin2, HALO does not reduce the LBN distance, because the majority of Fin2 requests are read requests, and there is little room for HALO to improve performance. As we can see in Fig. 13, normalized HDD's request response times are demonstrated for LRU and HALO. On average, HALO leads to 34.2%

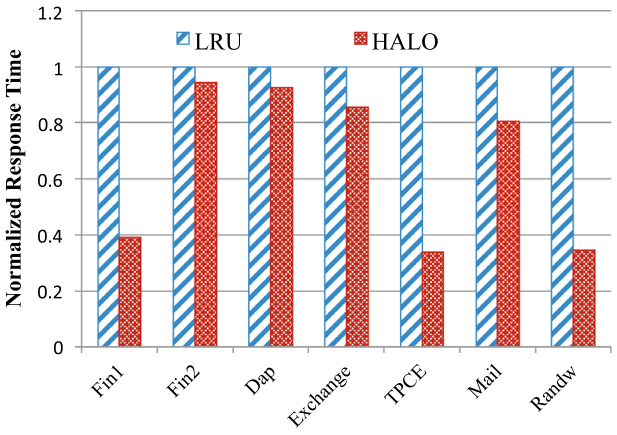


Fig. 13 Normalized disk response time

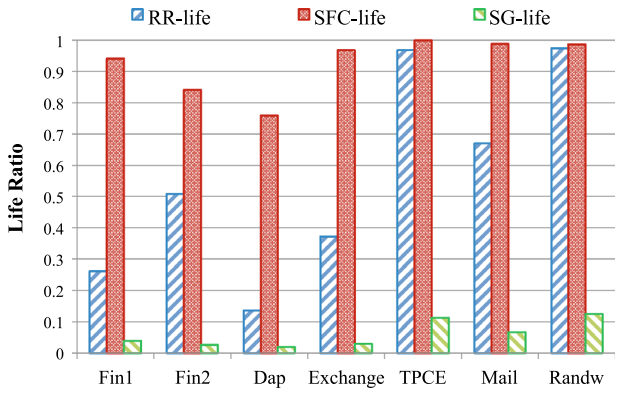


Fig. 14 Life ratio comparison between different wear leveling techniques

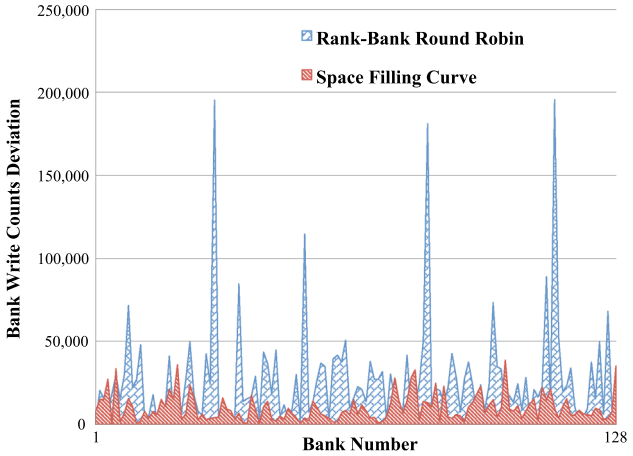


Fig. 15 Wear leveling results for Fin2

better request response time than LRU because of improved access locality and workload intensities achieved by HALO caching and Two-Way destaging algorithms.

4.3 Wear leveling results

Table 3 and Fig. 14 illustrate the wear leveling results for different wear leveling schemes. SG-max represents the maximum bank write count deviations for non-randomized region-based start-gap wear leveling (SG). Avg-counts represents the average bank write counts of all 128 banks. NAME-life represents the life ratio compared to perfect wear leveling, which can be calculated by $\frac{1}{(NAME\text{-}max/Avg\text{-}counts+1)}$. The "lifetime improvement" column gives an indication of the SFC wear leveling technique's lifetime improvement compared to that of SG. The average life ratio is 0.9255 for SFC-based wear leveling (SFC), 0.619 for rank-bank round-robin wear leveling (Rank-Bank RR or RR), and 0.0598 for SG. The average lifetime improvement with our schemes for all traces is 21.60.

Figure 15 shows the bank write count deviations for RR and SFC under Fin2 workload. The x-axis is the bank number for 8 ranks * 16 banks per rank. The y-axis is the deviation of wear counts among all banks. We observe that SFC can achieve very good wear uniformity and very low deviations for all banks. This is very helpful for extending the life ratio and lifetime of a PCM device. Bank deviations for the other traces are similar and not shown here for space limit.

5 Related work

In this section, we review recent works on nonvolatile memory and caching techniques.

Table 3 Wear leveling results

	SFC-max	RR-max	SG-max	Avg-counts	SFC-life	RR-life	SG-life	Improve#
Fin1	113,921	5,094,200	44,559,020	1,799,300	0.9405	0.261	0.0388	23.23
Fin2	38,507	195,648	7,510,848	202,822	0.8404	0.509	0.0263	30.96
Dap	90,802	1,820,127	13,999,887	285,233	0.7585	0.1355	0.0200	36.99
Exchange	1,93,633	9,699,420	1.87E+08	5,765,866	0.9675	0.3728	0.0299	31.36
TPC-E	3148	56,823	13,154,423	1,665,767	0.9981	0.967	0.1124	7.88
Mail	32,278	1,320,622	37,488,462	2,678,274	0.9881	0.6698	0.0667	13.82
Randw	12,762	23,859	6,135,289	871,481	0.9856	0.9733	0.1244	6.924

5.1 Hybrid memory systems

While NVRAM has non-volatility and energy benefits over DRAM and has great performance advantages over HDDs, it has limitations such as short device endurance and asymmetric access latency. To overcome these limitations, researchers have combined conventional memory systems with NVRAM to avoid these limitations and leverage the benefits of both HDDs and NVRAM. Ramos et al. [30], Wang et al. [40] and Zhang et al. [43] place PCM and DRAM side-by-side behind the bus to build a hybrid system [18]. Ramos et al. [30] have introduced hardware extensions to the memory controller (MC) to monitor popularity and write intensity of memory pages. They migrate pages between DRAM and PCM, and let operating systems (OS) in charge of updating memory mapping [39], such that performance-critical pages and frequently written pages are placed in DRAM, while non-critical pages and rarely written pages are in PCM. Zhang et al. [43] also rely on the MC to monitor access pattern. Their work differs from [30] by completely relying on OS to manage pages and treating DRAM as an OS-managed write partition. Over time, frequently written pages are placed into DRAM to reduce writes to PCM. Shi et al. present a flash+PCM hybrid nonvolatile disk cache system where flash is used mainly as read cache and PCM is used as write cache [33]. In [16], non-volatile memory is used as both buffer cache and journaling layers for ext4 file system. Our work aims at combining PCM and HDDs as a back-end storage device. Instead of relying on hardware and/or online detection mechanisms to partition data between PCM and HDDs, we store data blocks based on logical block numbers and record the mapping with hash tables, which avoids the involvement of OS in monitoring data access patterns and the need of additional hardware.

5.2 Caching and logging

Least recently used (LRU) and least frequently used (LFU) are common and effective cache replacement policies. LRU-k [24], LRFU [15], MQ [45] and LIRS [14] are important improvements to the basic LRU policy. They consider inter-access time, access history and access frequency to improve hit ratio. DULO [13] and DISKSEEN [7] complement LRU by leveraging spatial locality of data access. DULO gives priority to random blocks by evicting sequential blocks (those with similar block addresses and timestamps). However, the limited sequential bank size and volatility of DRAM prevents DULO from detecting sequences within a larger global address space and over a longer time scale. For this reason, DULO is not positioned to attain performance improvements over LRU for random access workloads in storage and file systems.

Logging is a method that aims to mitigate random writes to hard drives [31] and flash-based SSDs [42]. When data blocks are appended to early blocks rather than updated in place, the garbage collection is necessary and becomes a critical issue. DCD [23] uses a hard disk as a log disk for improving the random write performance of hard disks. However, it does not solve issues such as random reads from the logs disk and suffers from expensive destaging operations (still random writes) under heavy workloads. Several techniques employ non-volatile devices to boost the performance of storage and file systems. Some use NRVAM as the file system metadata storage [8,9,27], while others use NVRAM as LRU/LRW caches in file and storage systems [3,17,21,38]. However, these techniques have limitations. For example, they can only boost performance for certain types of file systems; they also cannot ensure the sequential write-back to HDDs due to the usage of LRU policy to manage cache replacement. The HALO scheme as proposed in our hybrid storage system addresses both of these limitations.

5.3 Wear leveling for PCM

Many research efforts have been invested in studying wear leveling in order to extend the lifetime of PCM. Qureshi et al. [29] make the writes uniform in the average case by organizing data as rotating lines in a page. For each newly allocated page, a random number is generated to determine the detailed rotation behavior. Seong et al. [32] use a dynamic randomized address mapping scheme that swaps data using random keys to prevent adversaries. Zhou et al. [44] propose a wear-leveling mechanism that integrates two techniques at different granularities: a fine-grained row shifting mechanism that rotates a physical row one byte at a time for a given shift interval, and a coarse-grained segment swapping mechanism that swaps the most frequently written segment with the less frequently written segments. Their work suffers from the overhead of hardware address mapping and the overhead of periodical sorting to pick up appropriate segments for swapping. Ipek et al. [12] propose a solution to improve the lifetime of PCM by replicating a single physical memory page over two faulty, otherwise unusable PCM pages. With modifications to the memory controller, TLBs and OS, their work greatly improves the lifetime of PCM. In [41], Wang et al. provide inter-set and intra-set wear-leveling techniques to uniformize write operations to ReRAM which is used as on-chip caches. Our wear leveling work is distinguished from these prior efforts. We regard the global address space as a multidimensional geometric space and employ a novel space filling curve-based algorithm to evenly distribute accesses across different dimensions. Comparing with existing work, our approach significantly extends the lifetime of PCM in hybrid storage devices.

6 Conclusions

In this paper, we propose a new hybrid PCM+HDD storage system that leverages PCM as a write cache to merge random write requests and improve access locality for the storage system. Along with this, we also design a cache scheme, named HALO, which utilizes the fast access and non-volatility features of PCM to improve system I/O performance with guaranteed reliability. Results from a diverse set of workloads show that HALO can achieve lower traffic rates to HDDs due to better caching and achieve higher system throughput led by smarter destaging techniques. Therefore, our approaches reduce execution times significantly, 36.8% on average. This hybrid storage organization is especially beneficial for workloads with intensive random writes. We also design two wear leveling schemes, rank-bank round-robin wear leveling and space filling curve based wear leveling, to extend the lifetime of PCM in the proposed hybrid devices. Our results show that SFC-based wear leveling improves the life time of PCM devices by as much as 21.6 times. Also, we have integrated an in-house PCM simulator into DiskSim 4.0 [5] as a new hardware model.

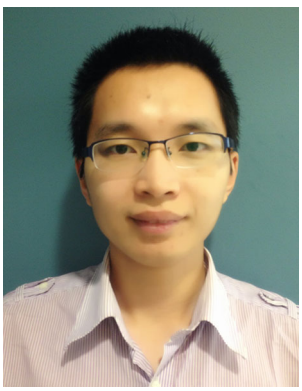
In the future, we plan to devise dynamically partitioned read/write cache schemes for improving a wider range of workload patterns. In addition, a scalable and hierarchical hash table can be introduced for further reducing memory space overhead.

Acknowledgements We are thankful to Mr. Michael Pritchard for his comments on the paper. This work is supported in part by the NSF awards ACI-1561041, CNS-1320016 and CNS-1059376 and by a UT-Battelle Grant to Auburn University.

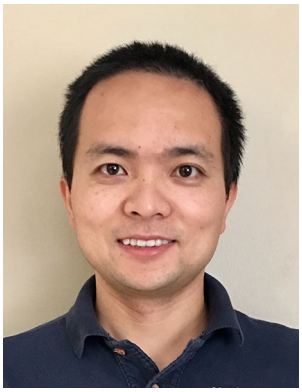
References

1. Akel, A., Caulfield, A., Mollov, T., Gupta, R., Swanson, S.: Onyx: A prototype phase change memory storage array. In: HotStorage'11
2. Are hybrid drives finally coming of age?—single-user. http://shop.objective-analysis.com/product.sc?productId=33
3. Baker, M., Asami, S., Deprit, E., Ouseterhout, J., Seltzer, M.: Non-volatile memory for fast, reliable file systems. ACM SIGPLAN Not. **27**(9), 10–22 (1992)
4. Brunelle, A.D.: Blktrace user guide (2007)
5. Bucy, J.S., Ganger, G.R.: Tech. rep. http://www.pdl.cmu.edu/DiskSim/
6. Chen, S., Gibbons, P., Nath, S.: Rethinking database algorithms for phase change memory. CIDR11 pp. 21–31 (2011)
7. Ding, X., Jiang, S., Chen, F., Davis, K., Zhang, X.: Diskseen: exploiting disk layout and access history to enhance i/o prefetch. In: USENIX ATC (2007)
8. Doh, I., Choi, J., Lee, D., Noh, S.: Exploiting non-volatile ram to enhance flash file system performance. In: ACM EMSOFT'07
9. Doh, I., Lee, H., Moon, Y., Kim, E., Choi, J., Lee, D., Noh, S.: Impact of nvram write cache for file system metadata on i/o performance in embedded systems. In: ACM SAC'09
10. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.: Space efficient hash tables with worst case constant access time. STACS **2003**, 271–282 (2003)
11. Gill, B., Modha, D.: Wow: wise ordering for writes-combining spatial and temporal locality in non-volatile caches. In: USENIX FAST'05
12. Ipek, E., Condit, J., Nightingale, E.B., Burger, D., Moscibroda, T.: Dynamically replicated memory: building reliable systems from nanoscale resistive memories. In: ASPLOS'10
13. Jiang, S., Ding, X., Chen, F., Tan, E., Zhang, X.: Dulo: an effective buffer cache management scheme to exploit both temporal and spatial locality. In: USENIX FAST'05
14. Jiang, S., Zhang, X.: LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. ACM SIGMETRICS Perform. Eval. Rev. **30**, 31–42 (2002)
15. Kim, C.: LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Comput. **50**(12), 1352–1361 (2001)
16. Lee, E., Bahn, H., Noh, S.H.: Unioning of the buffer cache and journaling layers with non-volatile memory. In: FAST, pp. 73–80. USENIX (2013)
17. Lee, K., Doh, I., Choi, J., Lee, D., Noh, S.: Write-aware buffer cache management scheme for nonvolatile ram. In: Proceedings of Advances in Computer Science and Technology. ACTA Press (2007)
18. Li, D., Vetter, J.S., Marin, G., McCurdy, C., Cira, C., Liu, Z., Yu, W.: Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications. In: Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pp. 945–956. IEEE (2012)
19. Liu, X., Schrack, G.: An algorithm for encoding and decoding the 3-d hilbert order. IEEE Trans. Image Process. **6**(9), 1333–1337 (1997)
20. Liu, Z., Wang, B., Carpenter, P., Li, D., Vetter, J.S., Yu, W.: PCM-based durable write cache for fast disk I/O. In: Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on, pp. 451–458. IEEE (2012)
21. Liu, Z., Wang, B., Wang, T., Tian, Y., Xu, C., Wang, Y., Yu, W., Cruz, C.A., Zhou, S., Clune, T., et al.: Profiling and improving i/o performance of a large-scale climate scientific application. In: Computer Communications and Networks (ICCCN), 2013 22nd International Conference on, pp. 1–7. IEEE (2013)
22. Micron phase change memory. http://www.micron.com/products/phase-change-memory
23. Nightingale, T., Hu, Y., Yang, Q.: The design and implementation of a dcd device driver for unix. In: USENIX ATC'99
24. O'neil, E., O'neil, P., Weikum, G.: The LRU-K page replacement algorithm for database disk buffering. In: ACM SIGMOD'93
25. OSDL: Iometer project. http://www.iometer.org/ (2004)
26. Pagh, R., Rodler, F.: Cuckoo hashing. AlgorithmsESA **2001**, 121–133 (2001)
27. Park, Y., Lim, S., Lee, C., Park, K.: Pffs: a scalable flash memory file system for the hybrid architecture of phase-change ram and nand flash. In: ACM SAC'08
28. Qureshi, M., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., Abali, B.: Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In: IEEE/ACM Micro'09
29. Qureshi, M.K., Srinivasan, V., Rivers, J.A.: Scalable high performance main memory system using phase-change memory technology. In: Proc. of the Internationl Symposium on Computer Architecture (2009)
30. Ramos, L., Gorbatov, E., Bianchini, R.: Page placement in hybrid memory systems. In: Proc. of the International Conference on Supercomputing (2011)

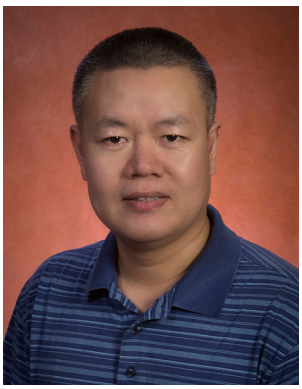
31. Rosenblum, M., Ousterhout, J.: The design and implementation of a log-structured file system. ACM SIGOPS Oper. Syst. Rev. **25**(5), 1–15 (1991)
32. Seong, N.H., Woo, D.H., Lee, H.H.S.: Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In: ISCA'10
33. Shi, L., Li, J., Jason Xue, C., Zhou, X.: Hybrid nonvolatile disk cache for energy-efficient and high-performance systems. ACM Trans. Des. Autom. Electron. Syst. **18**(1), 8 (2013)
34. SNIA BlockIO Traces. http://iotta.snia.org/tracetypes/3 (2006)
35. Soundararajan, G., Prabhakaran, V., Balakrishnan, M., Wobber, T.: Extending ssd lifetimes with disk-based write caches. In: USENIX FAST'10
36. Sun, G., Joo, Y., Chen, Y., Niu, D., Xie, Y., Chen, Y., Li, H.: A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In: IEEE HPCA'10
37. Umass trace repository. http://traces.cs.umass.edu/
38. Wang, A., Reiher, P., Popek, G., Kuenning, G.: Conquest: Better performance through a disk/persistent-ram hybrid file system. In: USENIX ATC'02
39. Wang, B., Liu, Z., Wang, X., Yu, W.: Eliminating intra-warp conflict misses in gpu. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE). IEEE (2015)
40. Wang, B., Wu, B., Li, D., Shen, X., Yu, W., Jiao, Y., Vetter, J.S.: Exploring hybrid memory for gpu energy efficiency through software-hardware co-design. In: Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, PACT '13, pp. 93–102. IEEE Press, Piscataway, NJ, USA (2013)
41. Wang, J., Dong, X., Xie, Y., Jouppi, N.P.: i2wap: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations. In: High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on, pp. 234–245. IEEE (2013)
42. Woodhouse, D.: Jffs: The journalling flash file system. In: Ottawa Linux Symposium, vol. 2001 (2001)
43. Zhang, W., Li, T.: Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architecture. In: International Conference on Parallel Architecture and Compilation Techniques (2009)
44. Zhou, P., Zhao, B., Yang, J., Zhang, Y.: A durable and energy efficient main memory using phase change memory technology. In: ISCA'09
45. Zhou, Y., Philbin, J., Li, K.: The multi-queue replacement algorithm for second level buffer caches. In: USENIX ATC'02



Zhuo Liu Zhuo Liu received his Ph.D. from the Department of Computer Science and Software Engineering at Auburn University in 2015 and received his bachelor's degree from Huazhong University of Science and Technology in 2007. His research interests include Cloud Computing, Big Data Analytics, Parallel I/O, and Storage Systems. He is working as a senior software engineer at Apple Inc. He is a member of The Institute of Electrical and Electronics Engineers and a member of The Association for Computing Machinery.



Bin Wang Bin Wang received his Ph.D. from the Department of Computer Science and Software Engineering at Auburn University in 2015 and received his Master's and Bachelor's degrees from Xi'an Jiaotong University in 2009 and 2006, respectively. His research interests include Computer Architecture, Computer Networks, High Performance Computing, and Distributed Systems. He is currently working for Arista Networks as a software engineer.



Weikuan Yu Weikuan Yu is a Full Professor in the Department of Computer Science at Florida State University (FSU). He served as a Research Staff Member in the Future Technologies group at Oak Ridge National Laboratory until 2009, and then an assistant and associate professor at Auburn University until 2015. Yu has founded the Computer Architecture and Systems Laboratory (CASTL) at FSU. His research interests include a multitude of technical areas including processor-memory architecture, big data analytics in social networks, high speed interconnects, cloud and distributed systems, storage and I/O systems. Yu is a senior member of IEEE and member of AAAS, ACM and USENIX.