PCM-Based Durable Write Cache for Fast Disk I/O

Zhuo Liu[†] *Bin Wang*[†] *Patrick Carpenter*[†] *Dong Li*[‡] *Jeffrey S. Vetter*[‡] *Weikuan Yu*[†]
Department of Computer Science[†] *Oak Ridge National Laboratory*[‡]
Auburn University, AL 36849 *Oak Ridge, TN 37831*
{zhuoliu,bwang,carpept,wkyu}@auburn.edu *{lid1,vetter}@ornl.gov*

Abstract

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, with the fast development of storage-class memories, new storage technologies with better performance and higher write endurance than FSSDs are emerging, e.g., phase-change memory (PCM). Understanding how to leverage these state-of-the-art storage technologies for modern computing systems is important to solve challenging data intensive computing problems. In this paper, we propose to leverage PCM for a hybrid PCM-HDD storage architecture. We identify the limitations of traditional LRU caching algorithms for PCM-based caches, and develop a novel hash-based write caching scheme called *HALO* to improve random write performance of hard disks. To address the limited durability of PCM devices and solve the degraded spatial locality in traditional wear-leveling techniques, we further propose novel PCM management algorithms that provide effective wear-leveling 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 SFC wear leveling extends the lifetime of PCM by a factor of 21.6.

I. Introduction

The explosive growth of data brings both performance and power consumption challenges. To solve these challenges, flash-based Hybrid Storage Drives (HSDs) have been proposed to combine standard hard disk drives (HDDs) and Flash-based Solid-State Drives (FSSDs) into a single storage enclosure [1]. Although flash-based HSDs are gaining popularity, 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. Lots of techniques have been proposed to address the issues [30, 29]. 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 FSSD-based HSDs. There remains a need of better technologies in the storage market.

Latest storage technologies are bringing in new non-volatile random-access memory (NVRAM) devices such as phase-change memory (PCM), spin-torque transfer memory (STTRAM), and resistive RAM (RRAM). These memory devices support the non-volatility of conventional HDDs 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 [5, 2]. Compared to FSSD, PCM is equipped with a number of performance and energy advantages [7]. First, PCM has much faster read response time than FSSD. It offers a read response time of around 50*ns*, nearly 500 times faster than that of FSSD. Second, PCM can overwrite data directly on the memory cell, unlike FSSD's write-after-erase. The write response time of PCM is less than 1μs, nearly three orders of magnitude faster than that of FSSD. Third, the program energy for PCM is 6 Joule/GB, 3 times smaller than that of FSSD [7]. Thus, PCM is a viable alternative to FSSDs for building hybrid storage systems.

A number of techniques have used NVRAM as data cache to improve disk I/O [6, 11, 18, 31, 13]. Most of them use LRU-like methods (e.g., Least Recently Written, LRW [11, 13]) to manage small size non-volatile cache to improve performance and reliability of HDD-based storage and file system. However, for GBs of high density PCM cache, 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. CSCAN method used in [13] 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.

1526-7539/12 $26.00 © 2012 IEEE
DOI 10.1109/MASCOTS.2012.57

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. To support this hybrid architecture, we propose a novel cache management algorithm, named HALO. It implements a new eviction policy and manages address mapping through cuckoo hash tables. These techniques together save DRAM overheads significantly while maintaining constant O(1) speeds for both insertion and query. Furthermore, HALO is very beneficial in terms of managing caching items, merging random write requests, and improving data access locality. In addition, by removing the dirty-page write-back 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 also address PCM's limited durability. Several existing wear-leveling techniques have shown good endurance improvement for PCM-based memory systems [26, 25, 33]. 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 will degrade read-ahead and sequential access performance of file systems. We propose a wear leveling technique called space filling curve wear-leveling, which not only provides a good write balance between different regions of the device, but also keeps data locality and enables good adaptation to the file system's I/O access characteristics.

Using two in-house simulators, we have evaluated the functionality of the proposed PCM-based hybrid storage devices. 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 SFC wear leveling extends the lifetime of PCM by a factor of 21.6. Our results demonstrate that PCM can serve as a write cache for fast and durable hybrid storage devices.

The rest of the paper is organized as follows. Section II provides a brief overview of related work, Section III the design of a hybrid PCM-HDD architecture. Then Section IV offers the details of wear-leveling. Section V provides experimental methodology and results, followed by our conclusions in Section VI.

II. Related Work

A. Caching and Logging

Least recently used (LRU) and least frequently used (LFU) are common cache replacement policies. LRU-k [21], LRFU [17], MQ [34] and LIRS [16] are important improvements to the basic LRU policy. They consider inter-access time, access history and access frequency to improve hit ratios. DULO [15] and DISKSEEN [9] 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 [27] and flash-based SSDs [32]. When data blocks are appended to early blocks rather than updated in place, the garbage collection is necessary and becomes a critical issue. DCD [20] 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 log 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 NVRAM as the file system metadata storage [10, 11, 24], while others use NVRAM as LRU/LRW caches in file and storage systems [6, 18, 31]. 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.

B. 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. [26] 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. [28] use a dynamic randomized address mapping scheme that swaps data using random keys to prevent adversaries. Zhou et al. [33] 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. [14] 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. 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. Compared with existing work, our approach significantly extends the lifetime of PCM in hybrid storage devices.

III. Leveraging PCM for 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 Figure 1(a), is currently popular.

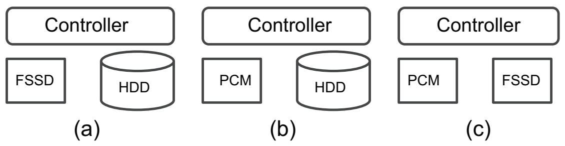


Fig. 1: Different architectures of hybrid storage devices

Exploring emerging NVRAM devices such as PCM as components in hybrid storage devices has attracted significant research interest. 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 Figure 1(b). 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 Figure 1(c). For example, Sun et al. [30] use this type of hybrid storage devices to demonstrate its 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 erasure-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 economical costs. In terms of cost per gigabyte, FSSDs are still about 10 to 20 times more expensive than HDDs.

For the aforementioned 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 Figure 1(b), we use PCMs to completely replace FSSDs while retaining HDDs for their capacity advantage. 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 drive that uses PCM in a write cache for HDDs to improve performance and reliability of HDD-based storage and file systems.

A. HALO-Based Caching and Mapping

It demands novel caching algorithms to use PCM as caches for HDDs. 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 HAsh table to manage PCM and merge random write requests, thereby improving access LOcality for HDDs. Figure 2 shows the HALO framework. The basic idea of HALO is to use a chained hash table 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.

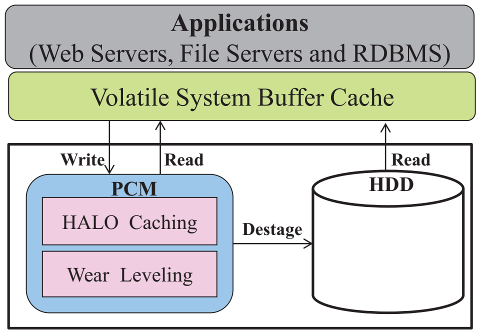

Applications
(Web Servers, File Servers and RDBMS)
Volatile System Buffer Cache
Write
Read
Read
PCM
HDD
HALO Caching
Destage
Wear Leveling

Fig. 2: The design of durable write cache

Data Structures – As shown in Figure 3, the chained hashtable includes an in-DRAM array (the *bucketinfo table*) and on-PCM mapping structures. Another cuckoo hashtable enables space-efficient fast query. The bucketinfo table stores information for HDD data regions. Each bucket item in the table represents a 1MB 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 for a list of cache items that map to the same sequential 1MB disk area; Bcounts represents the number of caching blocks; Recency records the latest access time-stamps for all cache items in this bucket. We use a global request counter 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 the 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 4KB 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, nonzero bits represent cached 512B HDD blocks. Each cache item is stored on each PCM block's meta data section [5].

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, PBN of the cache item is returned, which enables fast access of data information in the PCM. Traditional hash tables resolve collisions through linear probing or chained hash and they can answer lookup queries within O(1) time when its load factor is 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 [23, 12]. It can achieve fast lookups within O(1) time (albeit a bigger constant than linear hashing), as well as good space efficiency (high load factor). To address any 512B block in a 2TB disk volume, we need only four bytes (2^{32} blocks) to represent the LBN key. Thus we store full keys rather than partial keys into the hash table because using partial keys [8] leads to additional read accesses to PCM during false-positive read and full key read caused by key-value displacement.

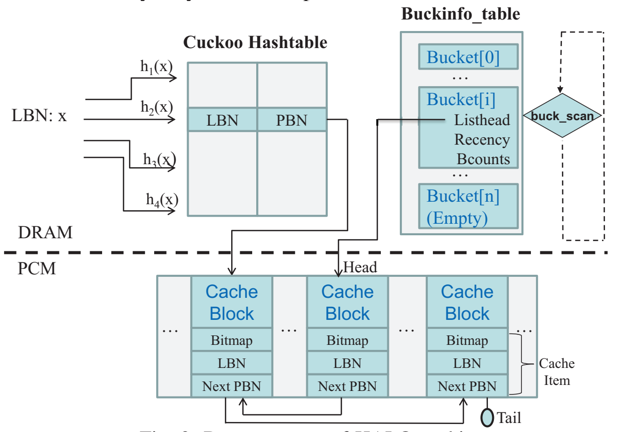


Fig. 3: Data structure of HALO caching

Caching Algorithm – 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 item. Then, if the bucket item of the bucketinfo table for the LBN is empty, it will be activated. Then, 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.

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:     $Bucket[i].recency \leftarrow globalReqClock$
7:   else {/*Cache miss*/}
8:     Allocate and write a PCM block
9:     if $Bucket[i]$ is empty then
10:      Activate $Bucket[i]$
11:      $activate\_bucks \leftarrow activate\_bucks + 1$.
12:    end if
13:    Link item to $Bucket[i].Listhead$, add to cuckoo hashtable
14:    $Bucket[i].recency \leftarrow globalReqClock$
15:    $Bucket[i].bcounts \leftarrow Bucket[i].bcounts + 1$
16:    $total\_bcounts \leftarrow total\_bcounts + 1$
17:  end if
18: else {/*This is a read request*/}
19:   Search the cuckoo hashtable.
20:   if cache hit then
21:     Read the PCM block with the returned PBN.
22:     $Bucket[i].recency \leftarrow globalReqClock$.
23:   else {/*Cache miss*/}
24:     Read the block from HDD.
25:   end if
26: end if
```

Destaging – When the utilization of PCM reaches a threshold, e.g., 95% of its total size, we will activate the destaging process to evict some data buckets out of PCM. A bucket is eligible to be destaged to HDDs if either of the following two conditions holds.

First, the bucket's bcounts needs to be greater than the average value of bcounts plus a constant $TH\_BCOUNTS$ and the bucket's recency needs to be older than the global request timestamp by a constant $TH\_RECENCY$. For every scan, these two thresholds will dynamically decrease with more scanned buckets 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 the global request timestamp by a constant $OUTDATE\_RECENCY$ ($OUTDATE\_RECENCY \gg TH\_RECENCY$). As soon as a bucket is identified as eligible for destaging, all cache blocks associated with the bucket are destaged to the HDD, 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 very old and small buckets, we evict them from the PCM by setting the control variable $OUTDATE\_RECENCY$.

Integrity Upon 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, system 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.

IV. Space Filling Curve Based 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 [25] avoids frequent write requests to the same regions by shifting cache lines and spreads requests through randomization at the granularity of cache lines (256B). 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 a wear leveling algorithm for PCM in hybrid devices, namely *Space Filling Curve (SFC)-based wear leveling*. Instead of using 256-Byte cache lines or single bits as wear leveling units, our algorithm uses stripes (32KB 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 32KB stripe is quite small (less than 0.1 ms). Hence, the data movement overhead will not affect the response times of front-end requests. Relevant parameters for SFC based wear leveling are listed in Table I.

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).

TABLE I: Parameters Used for Wear Leveling.

LSN	Global Stripe Number (0-64K)
$Stripe_{size}$	64 blocks (32KB)
N_{cubes}	Number of cubes (32)
N_{ranks}	Number of ranks in a PCM (8)
N_{banks}	Number of banks in a rank (16)
$Offset_{inStripe}$	Offset of blocks in a stripe
$Cube_{no}$	Cube Number $(0-31)$
$Cube_{stripes}$	Number of stripes in a cube (2048)
$Offset_{inCube}$	Offset of stripes within a cube
Seq_{no}	Sequence Number within a SFC cube
(R,B,S)	Rank, Bank, Stripe

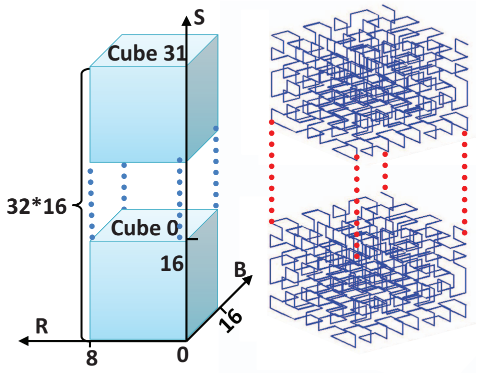


Fig. 4: Space filling curve based wear leveling

$$\begin{aligned}
\text{LSN} &= \left\lfloor \frac{\text{LBN}}{Stripe_{\text{size}}} \right\rfloor \\
Offset_{\text{inStripe}} &= \text{LBN}\ mod\ Stripe_{\text{size}} \\
Cube_{\text{no}} &= Stripe_{\text{no}}\ mod\ N_{\text{cubes}} \\
Offset_{\text{inCube}} &= \left\lfloor \frac{Stripe_{\text{no}}}{N_{\text{cubes}}} \right\rfloor \\
Seq_{\text{no}} &= StartGapMap(Offset_{inCube}) \\
(R,B,S) &= SFCMapFunc(Seq_{\text{no}})
\end{aligned} \tag{1}$$

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 $\times$ 16 $\times$ 8 (i.e., #stripe $\times$ #bank $\times$ #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 of 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 Figure 4. 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 $8 \times 16 \times 16$). The right figure shows a 3-dimensional space filling curve that is used for our work. The mapping scheme starts with LBN provided by the system and ends up with a 3-tuple (R, B, S) calculated based on Equation 1.

V. Experimental Evaluation

To evaluate the proposed PCM-based hybrid storage devices, we have designed a simulation framework that implements the HALO algorithm, traditional LRU caching schemes, and several wear leveling algorithms. In the simulator, we have a PCM simulator that simulates the performance and wearing characteristics of PCM. During evaluation, the block-level I/O traces are input to simulators. The I/O requests are then processed by caching and wear leveling schemes, which generate two intermediate trace files (i.e., a PCM trace file and an HDD trace file). The PCM trace file collects requests that hit in the PCM cache. The HDD trace file collects requests that result from cache misses and destaging. PCM trace file are replayed by PCM simulator to get response and wear leveling results. HDD trace files are replayed on a 500GB, 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. Because PCM devices have much higher (more than 10 times) throughput rates and response performance than those of HDDs [5], we reasonably assume that the total execution time of a workload trace is dominated by the replay time of the HDD trace.

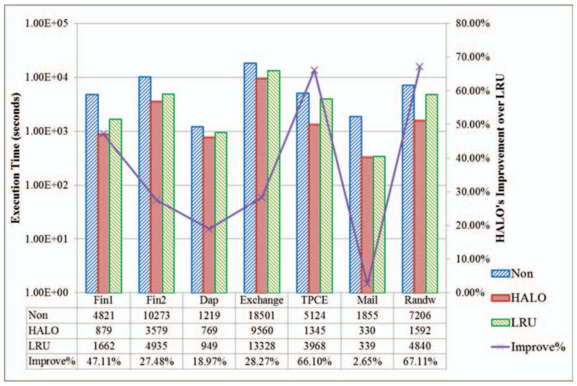
Based on the above discussion, the workload execution time can be calculated as follows: ($Total\_IO\_Size * Traffic\_Rate/Average\_Throughput$). The traffic rate is calculated as the total number of accessed disk sectors (after the PCM cache's filtering) divided by the total number of requested sectors 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 and 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.

1) Workloads: In our tests, we use seven trace files. Specifically, the traces *Fin1* and *Fin2* were collected with the SPC-1 benchmark suite at a large financial organization [3]; 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 5,000 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 [4]; the trace *Mail* was collected on a mail server by Florida International University [4]. The workload statistics are described in Table II. The seventh trace *Randw* was collected by us on the target disk while running the IOmeter benchmark with 4KB, 100% random, 100% write workloads for 2 hours with a dataset of 5.9GB [22].

TABLE II: Workload Statistics

	Fin1	Fin2	Dap	Exch	TPC-E	Mail
Write%	84.6%	21.5%	54.9%	74%	99.8%	90.1%
Dataset (GB)	18.0	8.85	84.2	163.8	13.2	85
AvgSize (KB)	3.38	2.4	77	13.65	10.48	4

A. Performance



	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. 5: Execution time

To evaluate execution times of seven traces, we choose 512MB as the cache size for Fin1 and Fin2, and 2GB as the cache size for the other five traces. Figure 5 shows the results. We notice that 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. The improvement of execution time comes from the reduction on traffic and the improvement on the average throughput of HDDs.

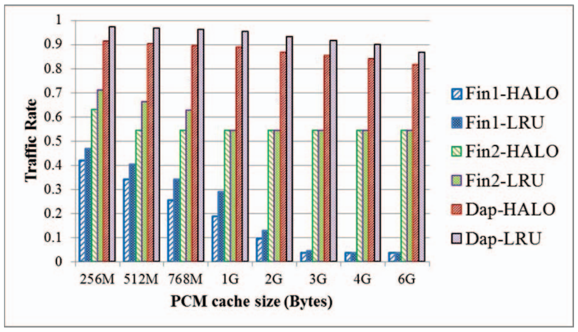


Fig. 6: Traffic rate for Fin1, Fin2 and Dap

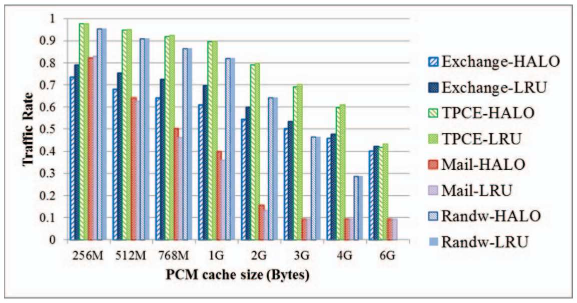


Fig. 7: Traffic rate for Exchange, TPC-E, Mail and Randw

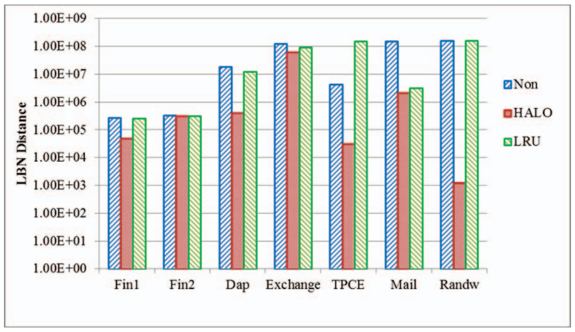


Fig. 8: Average inter-request LBN distance

Figures 6 and 7 show the traffic rate for the seven traces with no cache, the HALO cache policy and the LRU cache policy, respectively. In most cases (the cache size ranging from 256MB to 6GB), HALO consistently achieves 5% - 10% lower traffic rates than LRU. For the trace Fin1, HALO achieves as much as 10.3% lower traffic rates than LRU, where the cache size varies from 256MB to 3GB. 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% better traffic rate than LRU. For Randw, as it has a completely random write access pattern, no cache schemes can achieve good cache hits. That explains why the traffic rate of HALO and LRU are almost

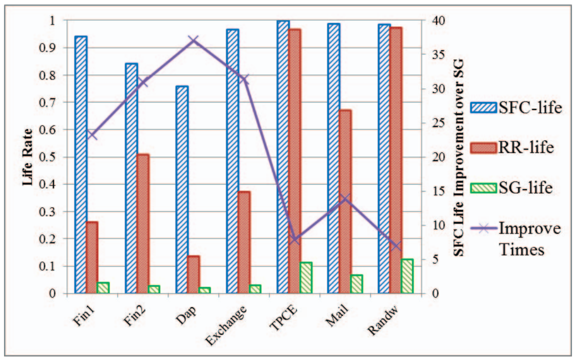


Fig. 9: Life rate comparison between different wear leveling techniques

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 768MB. 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 (512MB–2GB), the traffic rate under LRU caching is about 2% less than that under HALO caching, because most write requests are already in a uniform sequential pattern, so our cache scheme—which is targeted at random-write workload—cannot show good improvement.

We use the average inter-request LBN distance as a metric to evaluate the I/O access sequentiality to HDD, and the results are shown in Figure 8. We notice that the distance is reduced greatly by HALO caching for almost all traces. This explains why the execution time and average disk throughput with HALO are much better than with non-cache and with LRU, as shown in Figure 6 and 7. For Fin2, HALO does not reduce the distance much, because a majority of Fin2 requests are read requests, and there is little room for HALO to improve performance.

B. *Wear Leveling Results*

Figure 9 illustrates the wear leveling results for different wear leveling schemes. RR is a simple round-robin wear-leveling technique which iteratively distributes stripes first over ranks, and then over banks within the same rank. And for RR, a start-gap like stripe shifting scheme is implemented in each bank. 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 rate compared to perfect wear leveling, which can be calculated by $\frac{1}{(NAME\text{-}max/Avg\text{-}counts+1)}$. The "lifetime improvement" column indicates that SFC-based wear leveling improves the endurance much better compared to SG. The average life rate is 0.9255 for SFC-based wear leveling

(SFC), 0.619 for rank-bank round-robin wear leveling (RR), and 0.0598 for SG. The average lifetime improvement with our schemes for all traces is 21.60.

VI. 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 hybrid device. Along with this, we also design a cache scheme, named HALO, which utilizes the non-volatility of PCM to improve I/O performance. Results from a diverse set of workloads demonstrate that HALO can achieve lower traffic rates to HDDs and achieve better system throughput. Our approaches reduce execution times significantly. This hybrid storage organization is especially beneficial for workloads with intensive random writes. We also design a space filling curve based wear leveling scheme, 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.

In the future, we plan to design a scalable and hierarchical hash table to reduce memory space overhead. In addition, we plan to improve the on-demand destaging scheme with a load-aware batch strategy. This is likely to reduce the interference from read requests in sequential destaging operations, further reducing seek distances between read and write requests.

Acknowledgment

This work is funded in part by a UT-Battelle grant (UT-B-4000103043) to Auburn University, and enabled by National Science Foundation CRI award CNS-1059376.

References

[1] Are hybrid drives finally coming of age? http://shop.objective-analysis.com/product.sc?productId=33.

[2] Micron phase change memory. http://www.micron.com/products/phase-change-memory.

[3] Umass trace repository. http://traces.cs.umass.edu/.

[4] *SNIA BlockIO Traces*. http://iotta.snia.org/tracetypes/3, Online, 2006.

[5] A. Akel, A.M. Caulfield, T.I. Mollov, R.K. Gupta, and S. Swanson. Onyx: A prototype phase change memory storage array. In *HotStorage'11*.

[6] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. *ACM SIGPLAN Notices*, 27(9):10–22, 1992.

[7] S. Chen, P.B. Gibbons, and S. Nath. Rethinking database algorithms for phase change memory. *CIDR11*, pages 21–31, 2011.

[8] B. Debnath, S. Sengupta, and J. Li. Chunkstash: speeding up inline storage deduplication using flash memory. In *USENIX ATC'10*.

[9] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. Diskseen: exploiting disk layout and access history to enhance i/o prefetch. In *USENIX ATC*, 2007.

[10] I.H. Doh, J. Choi, D. Lee, and S.H. Noh. Exploiting non-volatile ram to enhance flash file system performance. In *ACM EMSOFT'07*.

[11] I.H. Doh, H.J. Lee, Y.J. Moon, E. Kim, J. Choi, D. Lee, and S.H. Noh. Impact of nvram write cache for file system metadata on i/o performance in embedded systems. In *ACM SAC'09*.

[12] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. *STACS 2003*, pages 271–282, 2003.

[13] B.S. Gill and D.S. Modha. Wow: wise ordering for writes-combining spatial and temporal locality in non-volatile caches. In *USENIX FAST'05*.

[14] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. *ASPLOS'10*.

[15] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang. Dulo: an effective buffer cache management scheme to exploit both temporal and spatial locality. In *USENIX FAST'05*.

[16] S. Jiang and X. Zhang. Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 31–42. ACM, 2002.

[17] C.S. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12), 2001.

[18] KH Lee, IH Doh, J. Choi, D. Lee, and SH Noh. Write-aware buffer cache management scheme for nonvolatile ram. In *Proceedings of Advances in Computer Science and Technology*. ACTA Press, 2007.

[19] Xian Liu and G.F. Schrack. An algorithm for encoding and decoding the 3-d hilbert order. *Image Processing, IEEE Transactions on*, 6(9):1333–1337, sep 1997.

[20] T. Nightingale, Y. Hu, and Q. Yang. The design and implementation of a dcd device driver for unix. In *USENIX ATC'99*.

[21] E.J. O'neil, P.E. O'neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *ACM SIGMOD'93*.

[22] OSDL. *Iometer project*. http://www.iometer.org/, Online, 2004.

[23] R. Pagh and F. Rodler. Cuckoo hashing. *AlgorithmsESA 2001*, pages 121–133, 2001.

[24] Y. Park, S.H. Lim, C. Lee, and K.H. Park. Pffs: a scalable flash memory file system for the hybrid architecture of phase-change ram and nand flash. In *ACM SAC'08*.

[25] M.K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *IEEE/ACM Micro'09*.

[26] M.K. Qureshi, V. Srinivasan, and J.A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA'09*.

[27] M. Rosenblum and J.K. Ousterhout. The design and implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review*, 25(5):1–15, 1991.

[28] Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA'10*.

[29] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *USENIX FAST'10*.

[30] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *IEEE HPCA'10*.

[31] A.I.A. Wang, P. Reiher, G.J. Popek, and G.H. Kuenning. Conquest: Better performance through a disk/persistent-ram hybrid file system. In *USENIX ATC'02*.

[32] D. Woodhouse. Jffs: The journalling flash file system. In *Ottawa Linux Symposium*, volume 2001, 2001.

[33] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA'09*.

[34] Y. Zhou, J.F. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *USENIX ATC'02*.