A Page-Based Storage Framework for Phase Change Memory

Peiquan Jin[1, 2], Zhangling Wu[1, 2], Xiaoliang Wang[1, 2], Xingjun Hao[1, 2], Lihua Yue[1, 2]

[1]*School of Computer Science and Technology, University of Science and Technology of China*
No.96, Jinzhai Road, Hefei, 230027, China

[2]*Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences*
No.96, Jinzhai Road, Hefei, 230027, China
jpq@ustc.edu.cn

***Abstract*—Phase change memory (PCM) has emerged as a promising candidate for next-generation memories, owing to its low power consumption, non-volatility, and high storage-density. However, PCM has limited write endurance, i.e., it can only undergo a limited number of write operations, leading to a short lifecycle. Thus, it is an important issue to find out an efficient way to use PCM in memory hierarchy, so that we can take advantage of the merits of PCM and prolong its lifecycle. Although PCM can support byte accesses, currently it has to work on top of page-based HDDs or SSDs. Therefore, a feasible way is to use PCM as a page buffer in memory hierarchy. Based on this assumption, in this paper we propose an efficient page-based storage scheme for PCM. We propose to use both DRAM and PCM as page buffers, forming a hybrid page buffer for DBMSs. Particularly, we develop three new techniques for PCM storage management. First, we propose a dual-bucket list to organize PCM spaces. Second, we use a small DRAM cache managed by an age-based policy to cache writes to PCM pages. Third, we propose a new page allocation algorithm that considers both page migration and page swapping to reduce the writes to PCM. We conduct extensive experiments on a simulated PCM-based storage system over both synthetic and realistic traces. The results suggest the effectiveness of our proposal.**

Keywords- Phase change memory; Storage management; Buffer management; Page migration

I. INTRODUCTION

Phase change memory (PCM) as a new kind of storage media has received much attention from both academia and industries [1]. PCM is byte-addressable and has low access latency. In addition, PCM is non-volatile. Particularly, it has faster read/write speeds than flash memory and magnetic disks. These features make PCM much promising in achieving higher I/O performance in future storage systems [2]. However, PCM suffers from limited write endurance, i.e., it can only undergo a limited number of write operations. This endurance issue is critical to the practicability and even the viability of PCM to be used in future storage systems.

The byte-addressability and non-volatility features of PCM make it be suitable for both main memory and secondary storage. However, since the read/write latency of PCM is still higher than that of DRAM, PCM is not suitable to replace DRAM as process memory space. In other words, it is more suitable to be used as a page buffer for files or databases. This yields a hybrid page buffer including PCM and DRAM. Of course, there are also other choices of using PCM in the memory hierarchy, but these are beyond the scope of this paper.

This paper focuses on the storage management schemes for PCM used as a page buffer. We also assume that DRAM is still available, i.e., PCM and DRAM can both be used as page buffers.

There are some challenges for organizing and managing PCM as a page buffer, because we need to consider the endurance of PCM. First, existing techniques to prolong the write endurance of flash memory, such as Hot-DL [3], cannot be simply applied in PCM-based storage systems. This is because of the distinct features between these two media. Flash memory offers page-level read/write operations and block-level erase operations, but it has to use the erase-before-write policy when updating data, known as the out-of-place updating [4]. In contrast, PCM supports in-place updating due to its byte-addressability and bit-alterability. Second, PCM has the nanosecond-level access latency and can be used as an alternative of main memory. Thus, its space management policy needs to be re-designed, because most of existing studies on flash memory regard flash memory as secondary storage. Third, to improve the I/O performance, traditional HDDs and SSDs often use a small buffer to cache data. Various cache replacement strategies have been proposed to enhance the cache hit ratio by utilizing the inherent properties of workloads, such as access frequency and locality [5, 6]. However, in PCM-based storage systems, we have to consider other measures, e.g., how to avoid frequent data between PCM and DRAM.

In this paper, we propose an efficient page-based storage scheme for PCM-based hybrid memory. We develop a systematic framework that consists of three components, namely space management, cache management, and page allocation. We devise new structures as well as new algorithms for these components and experimentally demonstrate the feasibility and efficiency of our designs. Briefly, we make the following contributions in this paper:

(1) We propose a new structure, called *Dual Dynamic Bucket Lists*, to organize the spaces of the hybrid memory including DRAM and PCM. The dual lists maintain for each page the write count and age information, which are further used to improve the performance of buffer management and page allocation.

(2) We use a small DRAM buffer for PCM to improve the endurance of PCM. Particularly, we propose an *Age-based Lazy Caching* (ALC) policy for the management of the buffer. The key idea of ALC is to use an age-based LRU list to buffer old pages and replace cold and young pages. Compared with the traditional LRU algorithm, the ALC policy can reduce a great number of writes to PCM and prolong the lifecycle of PCM.

(3) We propose a new page allocation algorithm for PCM. It incorporates page migration and page swapping to reduce PCM writes and to control the write amplification of PCM.

(4) We implement a simulated PCM-based storage system and compare our proposal with three state-of-the-art methods including PTL [7], the bucket-based WL algorithm [8], and the random swapping algorithm [9]. Extensive experiments over both synthetic and real traces show that our proposal outperforms the compared methods in terms of various metrics.

The rest of the paper is organized as follows. In Section II, we give a short background and the related work on PCM. In Section III, we discuss the details of our method. In Section IV, we present the experimental results. Finally, we conclude the paper in Section V.

II. Related Work

In this section, we first describe the necessary background on PCM and then present the related work in the literature.

PCM is a kind of non-volatile semiconductor memories and is a promising candidate for the storage and the main memory. The basic unit of PCM, called a PCM cell, uses the phase change material to store a bit by switching between an amorphous state and a crystalline state with electrical pulses. Writing a PCM cell includes two operations: SET, which requires wild pulse and low current to crystallize the phase change material, and RESET, which is controlled by high-power pulse to make the material amorphous. Reading a PCM cell is done by sensing the resistivity of phase change material, which requires very low power. As such, PCM, being non-volatile and bit-addressable, also bears the advantages of having low idle power and low read latency. However, the long SET operation increases the write latency, and the PCM cell can only sustain a limited number of writes, between 10^6 and 10^8 times in general [10]. Therefore, frequently writing to PCM will not only deteriorate the I/O performance but also shorten the lifetime of PCM.

In view of the potential write latency of PCM, a hybrid PCM+DRAM memory system is proposed in the literature, where DRAM is used to store frequently accessed date or write-intensive data. For example, a small amount of DRAM is used in front of PCM to cache PCM data in [11; 12]; in [13; 14], PCM is used as an alternative main memory. In such hybrid memory systems, effective data partition methods and efficient page replacement/migration strategies are desirable. A wide range of buffer management policies have been proposed on top of different storage media and different kinds of architectures. For example, BPLRU [5] and CCF-LRU [15], variants of the classical LRU policy [16], are proposed for flash-based storages such as SSD, which nowadays are being increasingly deployed in the enterprise storage systems. On the other hand, lazy-write organization [11] and CLOCK-DWF [14] are proposed for PCM-based hybrid memory. The buffer management policies therein enforce a mandatory buffering of all requests, considering the sharp read/write latency gap between the buffer and the secondary storage. In this paper, we take into account the negative consequence that a request for a cold page could evict a frequently accessed page.

Another line of research focuses on dealing with the limited write endurance issue of PCM. Basically, the proposed approaches in the literature fall in two categories: the write-count reduction and the wear leveling. The hybrid PCM+DRAM approach mentioned above belongs to the former. Nevertheless, it cannot prolong the life-span of PCM when the writes to PCM are seriously localized. As such, the wear leveling approach is proposed as an alternative. Based on dynamic or static wear leveling, various policies such as DAC [17], PWL [18], and Hot-DL [3] have been proposed for flash memories. However, the wear leveling methods for flash memories cannot be directly applied in PCM-based systems, due to the distinguished features of PCM like byte-addressability and in-place updating.

Next, we focus on wear leveling methods for PCM. One line is the deterministic age-based swapping [12; 19; 20], where a page whose age exceeds the threshold could be swapped out. The age of a PCM page reflects the write count of the page. An adaptive multiple data swapping and shifting scheme is proposed for PCM in [12]. The write patterns are tracked and used to determine whether page-level swapping and line-level shifting should be performed. This scheme implements wear leveling in multiple granularity and performs well in general, but it brings heavy storage overhead for maintaining the write counts of PCM. Zhou et al. [19] proposed two methods, row shifting and segment swapping, to achieve wear leveling of PCM. Compared with [12], these two methods spend less storage to maintain metadata; yet, they introduce high cost of searching for candidate segments to be swapped when the capacity of PCM is large. In addition, they swap pages at a fix interval, which may expose wearing out PCM pages under the attacks of malicious processes. The general age-based wear leveling methods suffer from the space overhead, especially when they are used at a fine granularity. To this end, random-based wear leveling methods are proposed to reduce the space overhead, where a randomized algorithm is used to swap data to a randomly selected place [9; 20]. For such random-based swapping methods, the selection of an accurate randomized algorithm and the determination of the swapping interval are of significant importance, since these two factors greatly influence the time performance and the effect of wear leveling.

Recently, it is pointed out that both age-based swapping and random-based swapping algorithms could incur the problem of write amplification. DSA, a table-based wear leveling technique, is proposed to tackle the write amplification problem in [21]. Rather than swapping the physical spaces of hot and cold data, DSA maintains chunk-level write counts for recently used segments, and reallocates a new physical chunk if the write count of a chunk exceeds certain threshold. In [7], a PRAM translation layer (PTL), which serves to dynamically translate logical addresses to physical addresses, is proposed to avoid write amplification; however, in this approach, the age difference between read-only pages and frequently updated pages will become bigger.

III. Page-Based Storage Management for Phase Change Memory

Figure 1 shows the overall architecture of the PCM-based hybrid memory. The entire space consists of two parts, namely

the PCM space and the DRAM buffer. In this paper, we use the hybrid memory as the data page buffer for DBMSs; therefore both the DRAM buffer and PCM spaces are organized as page lists.

Particularly, the DRAM buffer is used to cache requests to PCM pages. If a page request is hit on the DRAM buffer, we simply process the request in the DRAM. If the requested page is in PCM, we will use a buffer management policy to determine whether to read the page from PCM into DRAM.

Figure 2 shows the detailed structure of our space management scheme for PCM. Let's now focus on the PCM space, which includes the data area and the meta-data area. The data area is the actual physical space available for data storage, while the meta-data area is responsible for maintaining two kinds of meta-data information of pages in the data area. The first kind of meta-data is the page age, which is determined by the write count of the page and is increased by one whenever the page is updated. The second kind of meta-data is a mapping table from the physical page number (PPN) to the logical page number (LPN). This mapping table is dynamically reconstructed when the device is initialized. The LPN-PPN mapping changes frequently. If we store the LPN-PPN mapping on PCM, it will be difficult to predict and reduce the wear counts of PCM cells. As a consequence, the mapping table area will be worn out if no wear leveling method with a finer granularity is applied. On the contrary, storing the PPN-LPN mapping on PCM will not incur

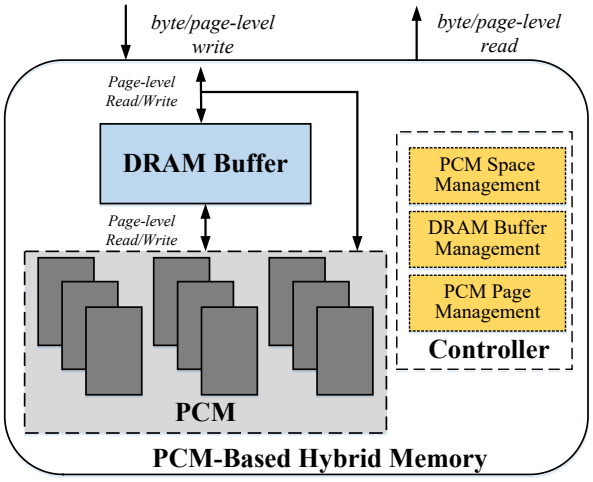

Fig. 1. Overall architecture of PCM-based hybrid memory

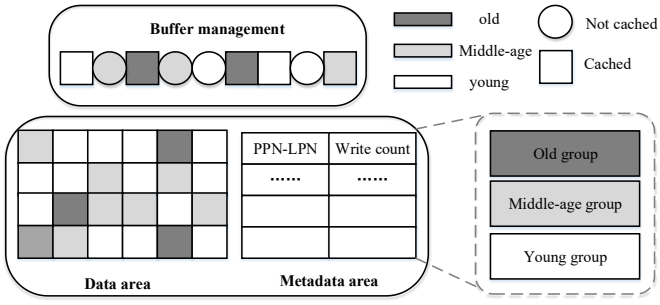

Fig. 2. Structure of the space management of PCM

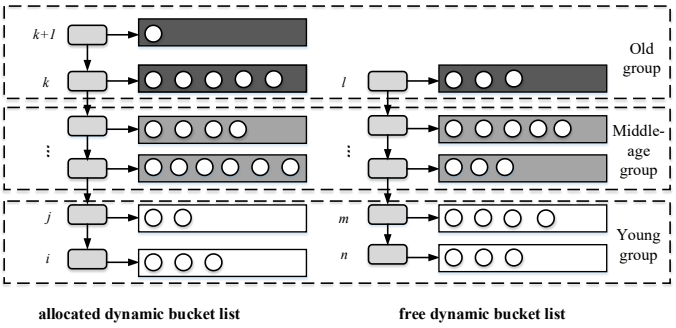

Fig. 3. Dual dynamic bucket lists

the wearing problem, because the write count of PCM cells storing the PPN-LPN mapping is always less than the wear count of the pages in the corresponding data area. In conclusion, the PPN-LPN mapping is suitable to be maintained in PCM, while the LPN-PPN mapping is suitable to be stored in DRAM.

Pages in the data area are divided into three categories based on their write counts: young group, middle-age group, and old group. The division of the three groups serves to facilitate the page allocation of PCM space and the buffer utilization.

In the following, we describe the above mentioned mechanisms in detail. First, to efficiently classify pages into three categories, we propose a new structure, called dual dynamic bucket lists, in Section III.A. Next, in Section III.B, we present the age-based lazy caching (ALC) policy, which serves to decide whether a non-buffered page is qualified to occupy the buffer space, and the page replacement policy, which is used to evict one buffered page in order to make room for the page that is newly allowed to be buffered. If the page to be purged from the buffer is dirty, it needs to be written back to the PCM. In this case, the system shall decide the placement of the page, i.e., being updated in-place or out-of-place, which depends on the category this page belongs to. And we present such page management algorithm in Section III.C.

A. PCM Space Management

The dual dynamic bucket list is used to organize all the pages in the data area of PCM. More specifically, we use two dynamic bucket lists to organize free pages (called *free dynamic bucket list*) and allocated pages (called *allocated dynamic bucket list*) respectively. The two lists share the same structure – a list of buckets – but with different lengths, as shown in Fig. 3. Each node in the list is a bucket; and each bucket is associated with a number indicating the age of pages in the bucket. Let w be a parameter of basic write count. When a page p is updated by n times ($(i-1)\times w \le n < i \times w$), then its age $a(p)$ is set to be i. That is, $a(p) = \lceil n/w \rceil$. All pages with the same age shall be put in the same bucket. We do not differentiate the order of pages within the same bucket. With the increasing of page write count, a page shall be moved to up-level buckets. In the beginning, there is only one bucket node in the free dynamic bucket list, encompassing all the available pages in PCM; correspondingly, the allocated dynamic bucket list is initially empty with no real data being stored in PCM.

The purpose of wear leveling is to prevent old pages from being worn out. As such, it is necessary to identify the oldness of pages in the data area. As mentioned above, we divide pages

in the lists into three groups: young group, middle-age group, and old group. Each group contains a set of pages and is circumscribed by the dotted boxes in Fig. 3. To partition pages into groups, we introduce a metric, the *average write count (AW)*, which is calculated by (3.1) and reflects the average write count of all pages in PCM. The value of this metric is achievable, since the write counts of pages are being tracked in the dynamic bucket list.

$$average\ write\ count = \frac{total\ write\ count}{total\ pages} \quad (3.1)$$

Let *TH* be a threshold of write count. A page is categorized into one of the three groups based on the following rules: i) if its write count is less than (AW-TH), it is put into the young group; ii) if its write count is greater than (AW+TH), it is put into the old group; iii) otherwise, it is put into the middle-age group. It is worth noting that the group to which a page belongs may change with the variation of *AW* value.

To achieve wear leveling, we can simply choose pages from the youngest bucket in the free dynamic bucket list, whenever any page allocation request arrives. However, this simple greedy policy may lead to poor effectiveness in scenarios where the free dynamic bucket list only contains old pages. Allocating old pages bring more writes to these pages, thereby deteriorating the write endurance of PCM. To alleviate this situation, we propose a new effective approach for page management in Section III.C.

B. DRAM Buffer Management

In our proposed PCM-based storage system, a small buffer is used to cache pages in PCM, which helps to reduce the write count of PCM, delay the updating of old pages, and improve the overall write performance of PCM.

Buffer management has been a common technique in computer systems to boost I/O performance, such as the traditional buffer used in HDDs and flash-based SSDs. However, the traditional buffer management strategies cannot be simply applied to PCM-based storage systems. This arises from the inherent differences between these two kinds of systems. In traditional DRAM buffer based storage system, the access latency between the DRAM buffer and the secondary storage is huge; and undoubtedly, using a DRAM buffer can evidently improve the I/O performance of the system. Nevertheless, in PCM-based storage systems, the difference of access latency in buffer and PCM is small. As a consequence, in PCM-based storage systems it is unnecessary to buffer all the requested pages from the perspective of reducing access latency; this is in stark contrast to the "buffering all" policy in traditional DRAM buffer. In fact, buffer in PCM-based storage systems undertakes the role of balancing the "wear-out" of PCM pages to a greater extent, rather than reducing the access latency. Therefore, the buffer management in PCM-based storage systems has a different design goal from that in traditional DRAM buffers.

We propose a new buffer management scheme, which consists of two parts: an age-based lazy caching policy (ALC) and a replacement strategy. The ALC policy determines whether a PCM page is qualified to be cached in the buffer. The replacement strategy decides which victim page to be purged from the buffer to make room for a newly buffered page. As highlighted in the above, the primary usage of buffer in a PCM-based storage system is as a vehicle to mitigate the "wear-out" of old pages. For this purpose, ALC gives higher priority to the buffering of pages with larger "age" values. In addition, ALC chooses to avoid buffering cold data so that hot data have chance to reside in the buffer for a longer time. Obviously, this reflects one of the "classical" goals of buffer management, i.e., achieving high utilization of the buffer space. For example, in Web caches, the caching decision policy often avoids caching the so-called "one-timers", the Web pages that are accessed only once by users, since caching one-timers benefits nothing. Similarly, ALC gives preference to buffering hot data. Finally, it is worth pointing out that the concept of "cold" and "hot" is with respect to the recency of page access in the buffer, while the division of the "young", "middle-age", and "old" (in Section III.A) group refers to the age of PCM pages. For example, if the data in a young PCM page is accessed frequently in recent time, this page is viewed to be hot; conversely, if the data in an old PCM page is seldom accessed recently, this page is viewed as a cold page.

Before describing the details of ALC, we first present the instrumental data structure for making caching decisions, *an age-based extended LRU list (A-eLRU)*, as shown in Fig. 4. The A-eLRU list maintains the information of recently accessed pages, each one being a record node in the list. Each record includes three types of information: i) the page age, ii) a status flag, indicating whether the page is cached in the buffer, and iii) the physical page address, if the page is cached. The record position of a page in the A-eLRU list represents the recency of that page. Specifically, the end of the list tagged with "*lru*" represents the least recently accessed page; the end of the list tagged with "*mru*" represents the most recently accessed page. In addition, we use a square and a cycle to represent a buffered and non-buffered page respectively. We also use three colors to represent the three groups into which a page has been categorized.

The procedure of handling a page request is described as follows. When a new request arrives, the system first decides the LPN of the page for the requested data. Next, it checks whether that page currently resides in the buffer with the aid of the A-eLRU list. If not, we need to decide whether this page should be brought to the buffer, controlled by the ALC policy. In case the buffer is not yet full (not a common situation), the requested page will be brought to the buffer so as to make full use of the buffer space. In most other cases, the buffer is full; and thus an existing buffered page needs to be purged if a positive caching decision is made for the newly requested page.

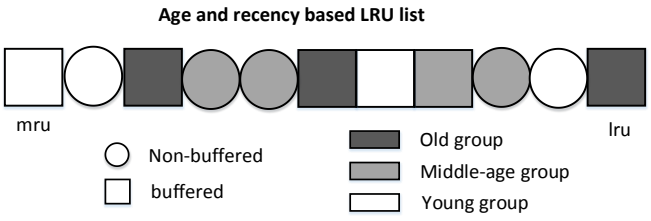

Fig. 4. Structure of the age-based extended LRU

Algorithm 1: Age-based Lazy Caching Policy

Input: logical page number *p*, operation type *op /*LPN the requested data belongs to*/*

Output: *record(p) /*a metadata information that represents page p is being accessed*/*

```
1:   record(p)=ListSearch(p) /*check whether the access record of LPN exist in A-eLRU list */
2:   if (record(p) exists or (op is write and p is mapped to a PCM page in the old group))
3:       if (record(p) does not exist) then
4:           create record(p) and update record(p).age;
5:       move record(p) in mru of the A-eLRU list;
6:       if (record(p).status="non-buffered") then
7:           q=page replacement policy();
8:           read page p from PCM and write to page q;
9:           record(p).status=buffered; record(p).ppa=q;
10:  else /*negative caching decision*/
11:      create record(p) and update record(p).age;
12:      place record(p) in the mru end of the A-eLRU list;
13:      record(p).status=non-buffered; record(p).ppa=null;
14:      if (op==write) then record(p).age+=1;
15:  return record(p);
```

Algorithm 1 describes the ALC policy, i.e., the overall procedure of making a caching decision. The caching decision of whether a page should be cached in the buffer is made based on the age of the corresponding physical page in PCM and the state of the LPN in the A-eLRU list. Suppose the LPN of the requested page is *p.* Recall that the supreme goal of introducing buffer is to alleviate the wear problem of PCM pages. As such, ALC gives old pages higher priority to be cached. Specifically, if the request is to write a page in the old group, ALC always makes a positive caching decision, no matter whether the page is cold or hot. In other cases, that is, the requested page being in either the young or middle-age group, ALC makes caching decisions based on the status of the A-eLRU list, which records the access information of pages. If the access record of *p* does not exist in the A-eLRU list (which also indicates the page p is currently non-buffered), it means the page *p* has not been accessed recently and thus *p* is viewed as a cold page. For a cold and non-old page, ALC makes a negative caching decision. For such a page, its read/write operations shall happen directly in PCM. The rationale behind this is that: on one hand, caching a cold page pays the price of evicting a warmer page, which may degrade the buffer utilization; on the other hand, there is no evident performance improvement of accessing a cold page in the buffer, compared with that in PCM. If the access record of *p* exists in the A-eLRU list, *p* is viewed as a hot page. Essentially, the position of a page's access record in the A-eLRU list reflects the recent Inter-Reference Recency (IRR) [22], which represents the number of other pages accessed between two consecutive accesses to a page. It is easy to see that the IRR value of a page *p* is smaller than that of pages behind it (towards the *lru* end) in the A-eLRU list. That is, a page *p* is warmer than those pages behind it. Thus, ALC makes a positive caching decision for *p*.

When ALC makes a positive caching decision for a page *p*, it means the page p shall be stored in the buffer. Specifically, the access record of p will be created if necessary and then be placed/moved in the *mru* end of the A-eLRU list (*line 3-5*). If the page p has already been cached in the buffer, no buffer eviction is needed. Otherwise, the page replacement policy is responsible for evicting certain page *q* to make room for page *p* (*line 7-8*). Meanwhile, the status flag and the physical buffer page address of *record(p)* in the A-eLRU list need to be updated (*line 9*). When ALC makes a negative caching decision for a page *p*, it means the page p shall be directly accessed in PCM. Although *p* is not cached in the buffer, we still create an access record of *p* and insert it into the *mru* end of the A-eLRU list (*line 11-12*). The status flag in the access record will be set "non-buffered" (*line 13*). This gives the currently cold page a chance to be cached in the buffer if it would be accessed again very soon in the future. Note that, whenever it is a write request, the write count of the page is increased by one (*line 14*), which keeps consistent the page age information in A-eLRU list and PCM metadata after the write request is completed.

Next, we describe the page replacement policy, which serves to free one buffer slot and is quite simple. The A-eLRU list maintains an invariant that the access record in the *lru* end corresponds to a buffered page. Recall that the access records in the A-eLRU could track non-buffered pages. To make room for the newly buffered page, the one represented by the access record of the *lru* end is purged from the buffer. If the evicted page is dirty, it needs to be written back to PCM. The write-back operation may incur page migration or swapping in PCM and we defer the discussion of this in Section III.C. It should be noted that to maintain the invariant mentioned above, after removing the *lru* end of the A-eLRU list, a certain number of neighboring records, i.e., corresponding to non-buffered pages, need to be deleted until a record of any buffered page is met.

Figure 5 shows how the buffer management scheme presented above works. Initially, we assume the buffer is full (i.e., the full capacity of buffer is 5 pages) and the A-eLRU list is depicted by Fig. 5(a). Suppose a read/write request for page E arrives and page E belongs to the young group. Since there is no access record of page E in the A-eLRU list, page E is viewed as a cold page and ALC makes a negative caching decision. Thus, page E is directly accessed in PCM; and a new access record of E is created on the fly and inserted into the *mru* end of A-eLRU, as shown in Fig. 5(b). Next, a read/write request for page B arrives. The non-buffered page B does not belong to the old group, but the access record of page B is found in the A-eLRU list, indicating page B to be hot. Based on Algorithm 1, ALC makes a positive caching decision and thus page B will be brought to the buffer. Before that, the page replacement policy chooses to evict page C, being referred to by the *lru* end of the A-eLRU list. To ensure the invariant of the *lru* end being buffered page, the access record of page D is removed from the list. Meanwhile, the access record of page B is created and placed in the *mru* end of the list. The result is shown in Fig. 5(c). Finally, a write request for page D arrives, neither page D being buffered nor its access record existing in the A-eLRU list.

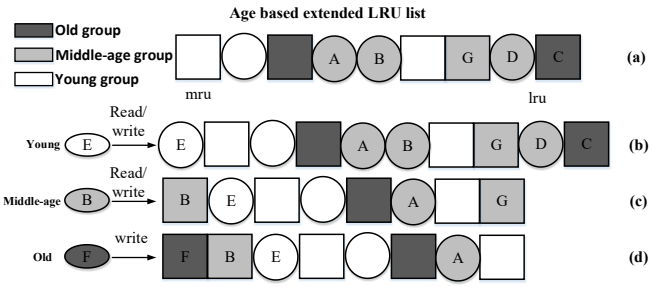

Fig. 5. An example of buffer managing scheme

Nevertheless, page D is an old page and thus ALC gives preference to caching it. Accordingly, page G, whose access record is located in the *lru* end, is evicted from the buffer. After that, the status of the A-eLRU list is captured by Fig. 5(d).

Note that the A-eLRU list is used to record accessing records of pages. In the first place, all the access records of pages in the buffer should appear in the list. In addition, the A-eLUR list should be long enough in order to accurately identify cold or hot pages.

C. PCM Page Management

In this section, we propose a novel page management scheme for PCM on the basis of the dual dynamic bucket lists that track the write counts of PCM pages.

The goal of wear leveling is to make the write count of each page close to the average write count, thereby lengthening the overall lifetime of PCM. To achieve such balance, young pages are expected to absorb more write requests. However, the access frequencies of logical pages from up-level programs are out of our control. But the knob we can twist is the mapping from logical page to physical page. Before presenting the page allocation algorithm, we use an example in Fig.6 to illustrate our basic idea. Suppose the logical page C has been evicted from the buffer. Page C being dirty, it needs to be written back to update the corresponding physical PCM page. Initially, page C was mapped to the physical page PPN10 that is an old page, as marked by (1) in Fig. 6. In this case, we can do an out-of-place updating. Rather than updating the content of logical page C still in PPN10, we select another free and young physical page PPN7 and update the page content there, as marked by (2) in Fig. 6. Then, the page mapping information in meta-data area will be updated accordingly to reflect such change. After that, the physical page PPN10 is reclaimed to be a free page. As we can see, with the above out-of-place updating, we avoid increasing the write count of the old page PPN10 by remapping the logical page C to a free and young physical page PPN7.

Generally, a page allocation request originates from two cases: (i) that a buffered dirty logical page is written back to an old physical page; and (ii) that a new logical page needs to be allocated with a free physical page. In the above, we have described the first case with Fig. 6. For the second case, due to the temporal locality of data access, a new logical page will often be accessed soon. In this case, allocating a young or middle-age physical page for this new logical page is preferred. However, at the time of allocation, it is possible that the free dynamic bucket list contains only old pages. We prefer to move an allocated young page that is occupied by a cold logical page

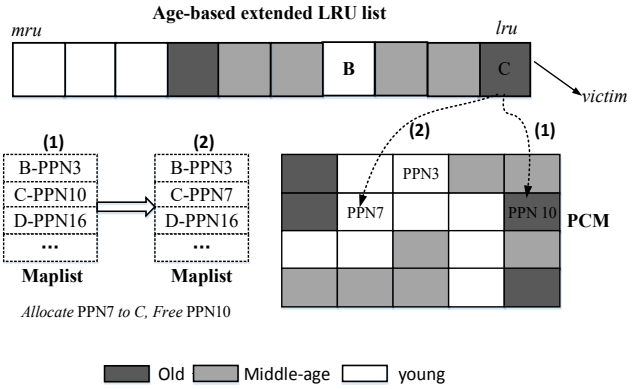

Fig. 6. The procedure of out-of-place updating

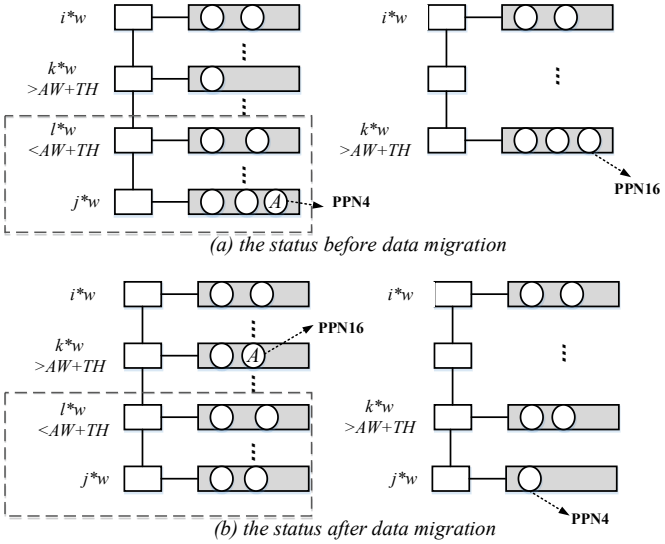

Fig. 7. An example of data migration incurred by allocation

to the free physical page. We use Fig. 7 to illustrate this. In Fig. 7, the left side is the allocated dynamic bucket list and the right side is the free dynamic bucket list. At the time of allocation in Fig. 7(a), all free physical pages are in the old group and the physical page PPN16 is selected to be allocated initially. We select a cold logical page A, which is currently mapped to the physical page PPN4, and move its content in PPN4 to PPN16. As shown in Fig. 7(b), the young physical page PPN4 is now a free page and becomes the hotbed of the newly allocated logical page.

Next, we summarize the page allocation procedure in Algorithm 2. When a request for a free page arrives, a page *p* in the youngest bucket of free dynamic bucket list is selected. If *p* belongs to either the young or the middle-age group, it is allocated directly (*line 1-3*). If *p* belongs to the old group, a page *q* is selected: (i) page *q* locates in the buckets of the allocated dynamic bucket list and belongs to young or middle-age group; (ii) the logical page mapped to page *q* is a cold page, i.e, there is no access record of q in the A-eLRU list. In order to allocate *q* to respond the page allocation request, first *p* is allocated to store the data of *q*; then the LPN of *q* is updated to link to *p* (*line 4-6*). Finally, *q* is released and reallocated (*line 7-8*).

Algorithm 2: Page Allocation

Output: a free page

```
1:  Select page p from the youngest bucket of the free bucket list;
2:  if (p.writecount − AW<TH) then
3:      allocate p and return p;
4:  else /*page p belongs to old group*/
5:      select cold page q from the youngest bucket of the
        allocated dynamic bucket list;
        /*q is in the young or middle-age group*/
6:      update the mapping from q to p and copy q to p;
7:      release q and add q into the free dynamic bucket list;
8:      allocate q and return q;
```

Previous PCM-based wear leveling policies, such as segment swapping [17], random swapping [9], and adaptive multiple data swapping and shifting scheme [12], often rely on various parameters like operation timing and the number of swapping pages. The accuracy of the estimation of these parameters has sensitive impact on the performance of wear leveling. In contrast to previous approaches, our page allocation does not rely on these parameters to do swapping or migration operations. In particular, we perform in-place updates on young pages and out-of-place updates on old pages.

D. Overhead Analysis

In this section, we summarize the storage overhead for managing 4 GB PCM. The information stored in the metadata area includes age information of all pages and the reverse mapping table from PPN to LPN (To reduce space overhead, we set the basic managing unit as a page). Since our system allows byte-level reading or writing, if the system updates several bytes, we increase the wear count of the pages which these updated bytes belong to. For a 4 GB PCM storage, the page size is set to 4 KB, so there are 2^{20} pages. We use 4 bytes to store the age information per page since the write limitation is 10^6-10^8, the space used to store age information is 4 MB. Meanwhile, 4 bytes are also enough to maintain the reverse mapping information per page, so another 4 MB is needed. Finally, the total space overhead to store metadata for a 4 GB PCM storage is 8 MB.

IV. Performance Evaluation

In this section, we present the evaluation results of our PCM management scheme described in Section III. We first describe the experimental settings on the workloads, metrics, and baseline algorithms. Then, we present comparative results with baseline algorithms with respect to various metrics.

A. Experimental Setup

We have implemented a PCM simulator, which incorporates all the components described in the architecture of Fig. 1.

We compare our proposal with three state-of-the-art approaches that have been proposed in the literature:

(1) *random swapping* [9], which swaps the page to be written with a randomly selected page for every 512 write operations to PCM.

(2) *the bucket-based WL algorithm* [8], which uses 500 buckets to maintain allocated and free pages separately; the write count difference of pages in the same buckets is 10.

(3) *PTL* [7], in which all pages are updated out-of-place. The original approaches of the three competitors did not use a buffer. To be fair, we implement a buffer for these methods and use the classic LRU as the cache replacement strategy. The page size of both buffer and PCM is set to 4 KB. The other parameter settings are described in Table I.

We use both synthetic and real traces in the following experiments. We use DiskSim [23] to generate two groups of traces, i.e., T1982 and T1955, by setting different read/write ratio and varying the locality. The ZIPF trace is generated using the algorithm in the literature [24]. The other group of traces, OLTP, is collected from PostgreSQL by recording the system accesses to the disk. Table II gives further details on these four traces. The memory footprint represents the number of distinct pages that are referenced by a trace. For the first four synthetic traces, manipulating the factor of locality is allowed, which influences the potential effectiveness of buffering. The locality of 80%/20% means that 20% of "hot" pages absorb 80% of requests.

TABLE I PARAMETERS IN THE EXPERIMENTS

Parameters	Value	
	Synthetic traces	Real traces
PCM size	12000 pages	52000 pages
Buffer size	1000 pages	
w	10	
TH	30	

TABLE II SYNTHETIC AND REAL TRACES USED IN THE EXPERIMENTS

Trace	Memory Footprint (#pages)	Read/Write Ratio	Locality	Total Requests
T1982	10,000	10% / 90%	80% / 20%	300,000
T1955	10,000	10% / 90%	Uniform	300,000
ZIPF	47,023	51% / 49%	80% / 20%	500,000
OLTP	51,880	77% / 23%	Unknown	607,390

B. Results

Impact of parameters *TH* and *w*. The parameter *w* determines the speed at which the age of a page is increased based on the write count. The parameter *TH* circumscribes the boundaries of categorizing pages into three groups: old, middle-age, and young. Our proposed PCM management scheme relies on the page age and page group to make caching decisions (in Section III.B) and implement page allocation for wear-leveling (in Section III.C). As such, we first study the impact of the parameter *TH* and *w* on the performance of our PCM management scheme.

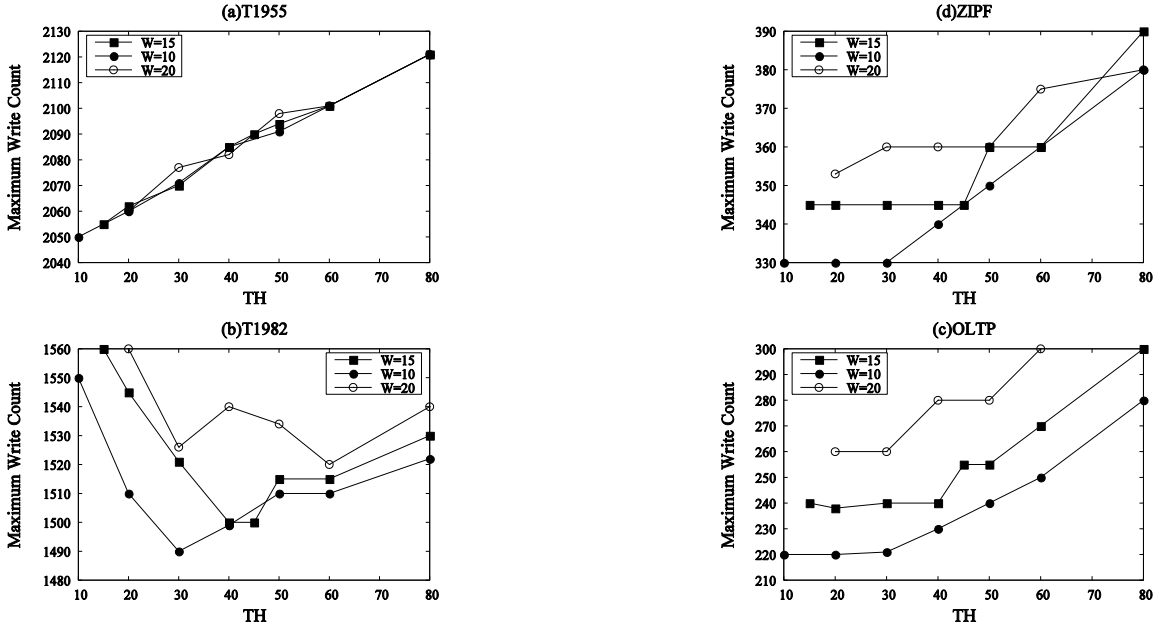

Fig. 8. Impact of TH and *w* on maximum write counts of PCM

We run our proposed method over each trace with varying values of *TH* and *w*. We observe the maximum write count experienced by any PCM page, which reflects the lifetime of PCM. The smaller the value of the maximum writes count, the better the performance of a method. Fig. 8 shows the results over four traces. We can see that the maximum write count increases with *w* and *TH* in most cases, especially when *TH* is over 30. This is because, with a large value of *TH* and *w*, a page will not be categorized into the old group until a large number of writes have occurred. This undermines the endeavor of ALC to prioritize buffering old pages to mitigate the worn out of these old pages, thereby yielding a large value of the maximum write count. On the other hand, if the value of *TH* and *w* are too small, then a page would be prematurely classified as an old page, which in turn might unnecessarily trigger an extra page swapping when an old dirty page is written back after being evicted from the buffer. To summarize, there is a trade-off in setting the value of the parameter *TH* and the parameter *w*. Based on the results observed here over four traces, we set *TH* to 30 and *w* to 10 in the following experiments.

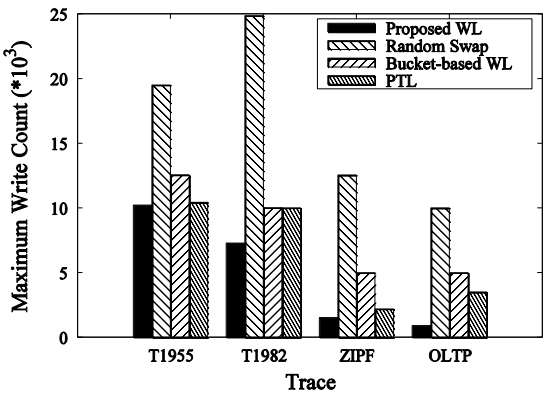

Fig. 9. Comparison of maximum write counts

Maximum write count on PCM. The goal of PCM management scheme is to balance the wearing degree of PCM pages. We evaluate the maximum write count of pages fed by each trace. As mentioned above, a larger value of the maximum write count indicates a worse performance achieved by a wear-leveling method. In this part, we compare our method with the three alternatives.

Figure 9 shows the maximum write counts of the four methods over various traces. Our proposed method outperforms the other three methods over all the four traces, indicating the effectiveness of our buffer management and PCM page allocation algorithm. Reducing maximum write count implies that our proposal has longer PCM lifetime than other methods.

We also note that the random swap algorithm yields the worst result, because it does not consider the age of physical pages when selecting a page for swapping. In addition, PTL has similar maximum write count with our approach when measured on trace T1955. This is because that the requests in T1955 are evenly distributed over pages.

Distribution of write requests. Next, we observe the distribution of write counts experienced by all PCM pages. In this part, all the four methods receive the same number of total requests over each trace. Since similar results are observed over the four traces, we only report results over the trace T1982 in the following.

Figure 10 shows the records of write count of each physical page for the four methods. Note that the range of *y*-axis values is different across different sub-figures. We make three observations. First, our method achieves the smallest gap between the maximum page write count and the minimum page write count, being less than 100. From Fig. 10 (b) and (c), we can also see that the gap value of PTL is smaller than that of the bucket-based WL. In addition, the random swapping method leads to the largest deviation of page write counts. As shown in Fig. 10 (d), a handful of pages receive 0 write count, while some

pages experience a startling 25000 write operations. Large deviation of write counts deviates from the primary goal of page management in PCM. The poor performance of random swapping derives from the random selection policy of page swapping, taking into no account the ages of PCM pages. As such, old pages may not be selected to be swapped and some empty pages may even not be allocated for page swapping. Second, the write counts of individual pages are evenly distributed in our method, which well meets the rudimentary goal of page wear leveling. Compared with random swapping and the bucket-based WL, PTL achieves better results in terms of balanced page write counts, due to its out-of-place updating. As shown in Fig. 10 (c), the majority of pages are worn evenly from 7700 to 8200, although some pages undergo a huge number of write operations. Third, with a closer look at Fig. 10 (a), we can see that there is a line formed by a large number of points gathering around the y-axis value 7247. In fact, this line reflects the boundary between the "old group" and the "middle-age group". In trace T1982, the average age is 7247; the page whose write count is greater than $ave + TH$ (i.e., 7247+30=7277) belongs to the old group.

First failure time of a PCM cell. The maximum write count is a metric evaluated when a fixed trace is fully served by a method. In contrast, the first failure time represents how many write requests from a trace can be served by a method before the first failure of any PCM cell takes place. Both these two are important metrics used in the literature. In this part, we compare our method with the other three alternatives in terms of the total write operations finished at the time of first PCM cell failure.

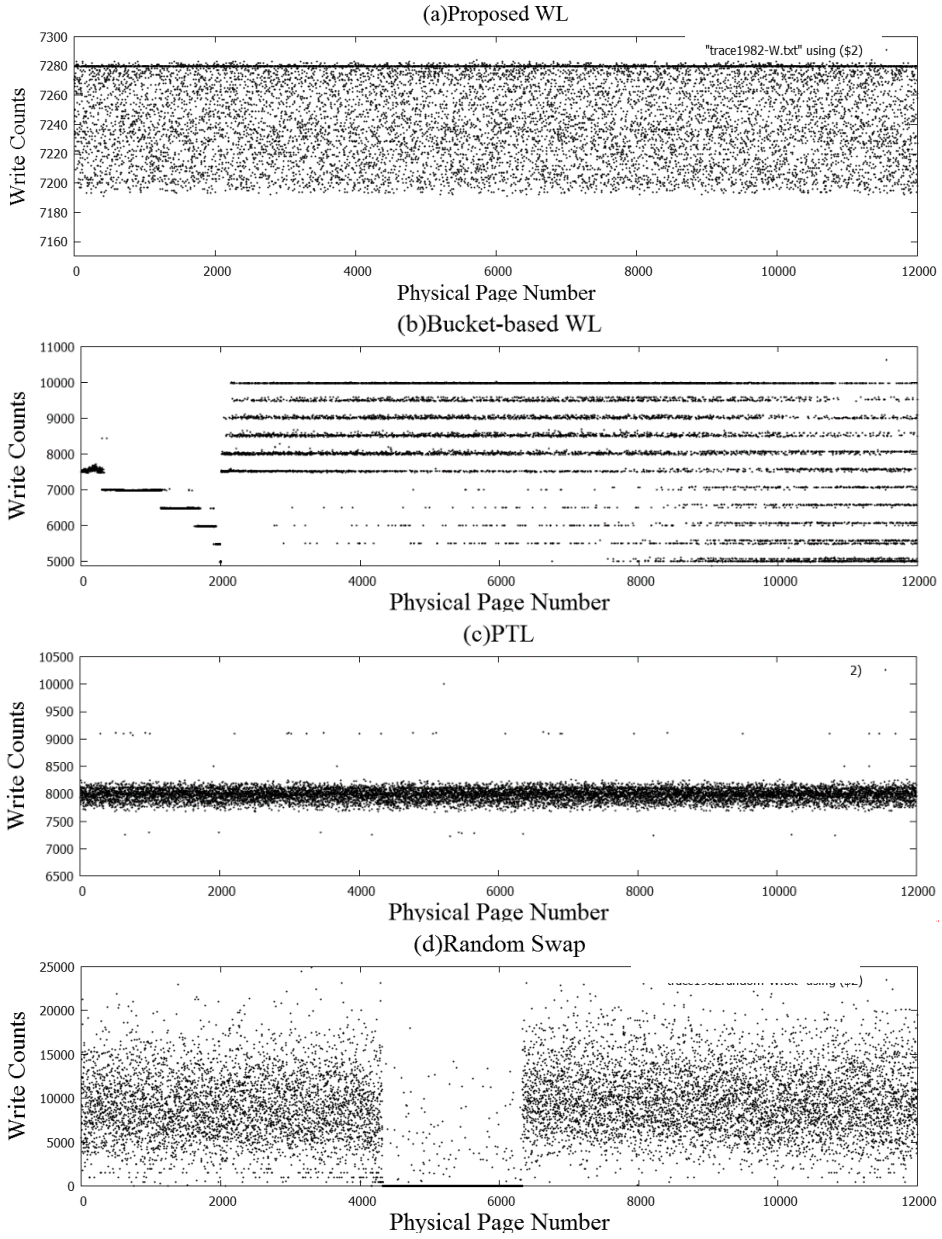

Fig. 10. Distribution of write count in PCM after applying T1982

The parameter settings of this group of experiments are described as follows. First, we set the write limitation of a PCM page to be 10^4. Although it has been reported that a PCM cell in general can tolerate 10^6~10^8 writes [10], we set this value for the sake of controlling the scale of experiments. But we do vary the values of write limitation to mitigate such impact. Second, we set the capacity of PCM to be 12000 pages. In the ideal case, 12000 pages can withstand 12000*10000 writes.

We report the total write count of the four methods over two traces, T1955 and T1982, before the first PCM page failure in Table III. As we can see, our method achieves the largest number of writes. In particular, it reaches about 99.5% and 96.9% of the ideal write count over T1955 and T1982 respectively. In contrast, over T1982, PTL, the bucket-based WL, and random swapping only achieve 80%, 72%, and 22% of the ideal write count, respectively.

TABLE III LIFETIME UNDER SYNTHETIC TRACES

Policies	Write count of wearing out PCM	
	T1955	T1982
Our proposal	119,511,349	116,328,780
PTL	117,628,266	95,740,849
Bucket-based WL	94,416,434	86,691,668
Random swap	62,941,008	26,001,132

We also study the sensitivity of the parameter, i.e., the write limitation of a PCM page, by varying its value from 10000 to 50000. Figure 11 shows the overall write counts of each method over different page write limitations. Our method yields less writes to PCM than the other three methods consistently over different values of write limitation. This also implies that out method can be applied to various kinds of PCM media that may possess different property in terms of page write limitation.

Impact of buffer management. In previous experiments, we study the performance of our PCM management scheme as a whole. In this part, we discuss the impact of buffer management on the overall performance.

The primary purpose of buffer in the hybrid PCM system is to mitigate the worn-out of old PCM pages. Therefore, it is neither the highest priority nor ultimate goal for the buffer management in PCM to blindly seeking the maximum cache hit rate. However, improving the utilization of buffer space does make sense. We compare our buffer management scheme, "ALC+AeLRU", with the traditional "ubiquitous caching + LRU" (where every page will be cached in the buffer and LRU serves as the cache replacement algorithm). We vary the buffer size from 500 pages to 3500 pages.

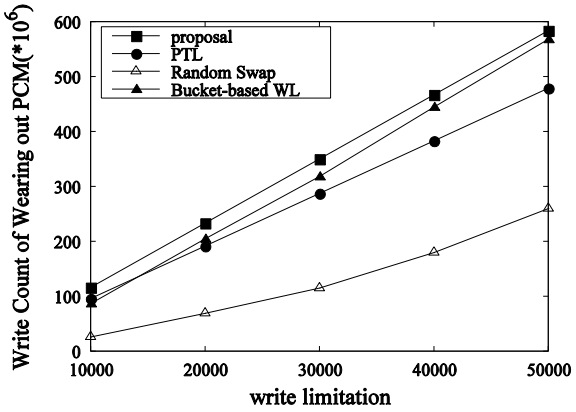

Fig. 11. Life time of PCM when varying the write limitation

Figure 12 shows the cache hit ratio of the two management schemes over four traces. We make two observations. First, in general a larger buffer size yields a higher cache hit ratio. However, the achieved cache hit ratio does not necessarily keep increasing as the buffer size increases. This is because the hit ratio is highly related to the locality of workload. For example, in T1982, 20% hot pages account for 80% requests. When the buffer size reaches 2000 pages (20% of the universe size), we see the hit ratio increases slowly, as shown in Fig. 12 (b). In Fig. 12 (a), since the distribution of page accesses in T1955 is uniformly distributed, the hit ratio increases linearly with the increasing of buffer size. Second, our buffer management scheme performs better than the classic method consistently over all traces and with different buffer sizes. Even with a small buffer size, our method brings moderate cache hit ratio, as shown in Fig. 12 (a) and (b). This arises from the fact that ALC avoids caching a cold page – according to the access records of pages maintained in the A-eLRU list – unless it belongs to the old group. The ALC caching policy is in stark contrast to the "ubiquitous caching", which might lead to the unwanted situation where a cold young page evicts a buffered hot page. Next, we study the impact of buffer management. In our method, some fraction of write requests are served directly in the buffer and we denote their number as "Proposed-BW". We denote "Proposed-PW" as the number of write operations to the PCM triggered by the following two parts: i) the fraction of write requests that are directly accessed in PCM, by-passing the buffer; ii) the dirty pages that need to be written back to PCM when evicted from the buffer. Note that, here we did not count the portion of extra writes to PCM due to page swapping. Similarly, with the classic "ubiquitous caching+LRU", "LRU-PW" represents the number of writes to PCM due to The write-back of dirty pages from buffer; "LRU-BW" represents the number of requests served in the buffer, which turns out to be the total number of requests in the workload. For brevity, we report the normalized value of "Proposed-PW", "Proposed-BW", and "LRU-BW" to "LRU-BW".

Figure 13 shows the results over four traces. We make two observations. First, when a larger buffer size is used, the normalized value of "Proposed-BW" increases, while the normalized value of "Proposed-PW" decreases. Second, "Proposed-PW" is consistently smaller than "LRU-PW" over all traces and with different buffer sizes. A smaller value implies that less writes are fed to PCM. It shows that our ALC caching policy performs better than the classic approach. In addition, in Fig. 13 (a), the gap between "Proposed-PW" and "LRU-PW" seems to be small; nevertheless, "Proposed-PW" is still smaller. This is because the trace T1955 does not present good workload locality, which is inherently not good for caching.

In summary, our ALC caching policy carefully avoids caching cold pages. This yields a high hit ratio and prevents writing hot pages to PCM frequently, which helps prolong the lifetime of PCM.

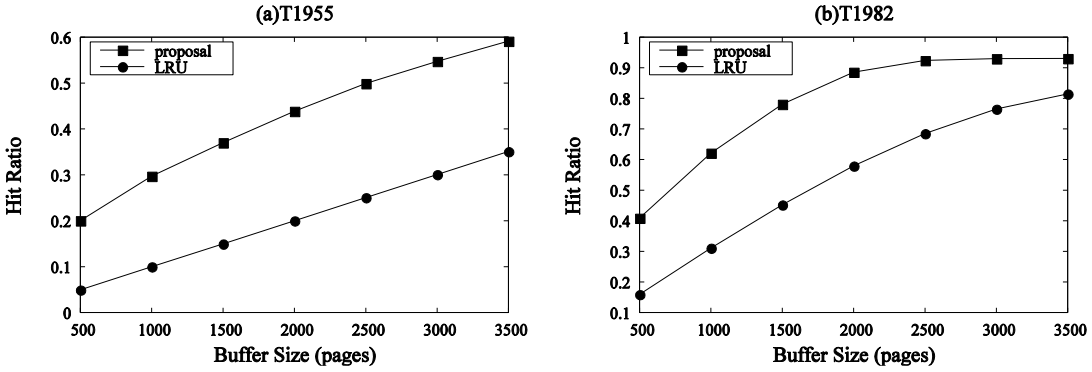

Fig. 12. Hit ratios under various buffer sizes

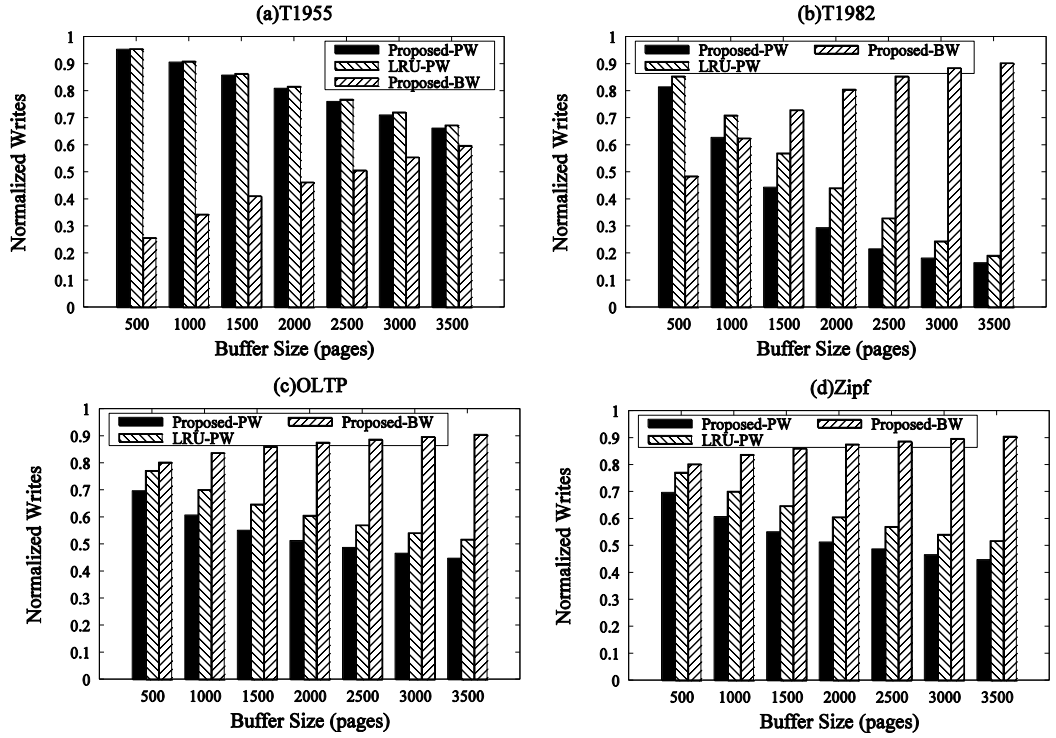

Fig. 13. Normalized writes under various buffer sizes

V. CONCLUSIONS

In this paper, we propose a new wear-leveling page management scheme for the hybrid PCM-based storage subsystem. It includes two parts: the ALC caching policy and the PCM page allocation scheme, which work together to balance the worn-out of PCM pages. Both the ALC caching policy and the page allocation algorithm respect the old pages in terms of preventing their further worn-out. To this end, a dual dynamic list is proposed to efficiently maintain the age information of PCM pages and partition pages into three groups (i.e., young, middle-age, and old group). Our page management scheme achieves the wear-leveling in two levels. In the buffer level, the ALC caching policy prioritizes buffering old pages to mitigate their worn-out and is inclined to cache hot pages to reduce the number of write operations to PCM. In the targeted PCM level, our page allocation algorithm wields the tool of page migration and page swap to balance writes to PCM pages, which seeks to achieve the wear-leveling while curbing the write amplification ratio. To evaluate the performance of our PCM page management scheme, we conduct extensive experiments on a simulated PCM-based storage system over both synthetic and realistic traces. The experimental results show that our method performs significantly better than the other three alternatives in various metrics, which puts our method in a competitive position as the page management scheme for the PCM-based storage subsystem.

There are a number of future works that need further investigating. First, we will try to find the optimal configuration of the DRAM buffer size for DRAM/PCM-based hybrid memory. Second, in this paper we did not consider the low-energy property of PCM. In future, we will study the impact of storage schemes on energy consumption. Finally, the non-volatility of PCM may introduce new potentials for building efficient in-memory computing systems. In future, we will focus on utilizing the non-volatility of PCM in hybrid memory

systems to improve the efficiency and consistency of in-memory computing and data management.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation of China (61472376 and 61672479). Peiquan Jin is the corresponding author of the paper.

REFERENCES

[1] Zhangling Wu, Peiquan Jin, Chengcheng Yang, Lihua Yue. 2014. APP-LRU: A New Page Replacement Method for PCM/DRAM-Based Hybrid Memory Systems. NPC. 84-95.

[2] Fei Xia, De-Jun Jiang, Jin Xiong, Ning-Hui Sun. 2015. A Survey of Phase Change Memory Systems. Journal of Computer Science and Technology. 30, 1, 121-144.

[3] Se Jin Kwon, Tae-Sun Chung. 2013. Hot-LSNs distributing wear-leveling algorithm for flash memory. ACM Transactions on Embedded Computing Systems. 12, 1, 62.

[4] Ke Lu, Peiquan Jin, Puyuan Yang, Shouhong Wan, Lihua Yue. 2014. Adaptive in-page logging for flash-memory storage systems. Frontiers of Computer Science. 8, 1, 131-144.

[5] Hyojun Kim, Seongjun Ahn. 2008. BPLRU: a buffer management scheme for improving random writes in flash storage. FAST. 239-252.

[6] Qingsong Wei, Cheng Chen, Jun Yang. 2014. CBM: a cooperative buffer management for SSD. MSST. 1-12

[7] Gyu Sang Choi, Byung-Won On, Kwonhue Choi, and Sungwon Yi. 2013. PTL: PRAM translation layer. Microprocessors and Microsystems. 37, 1, 24-32.

[8] Chi-Hao Chen, Pi-Cheng Hsiu, Tei-Wei Kuo, Chia-Lin Yang, and Cheng-Yuan Michael Wang. 2012. Age-based PCM wear leveling with nearly zero search cost. DAC. 453-458.

[9] Alexandre Peixoto Ferreira, Miao Zhou, Santiago Bock, et al. 2010. Increasing PCM main memory lifetime. DATE. 914-919.

[10] Hung-Sheng Chang, Yuan-Hao Chang, Pi-Cheng Hsiu, Tei-Wei Kuo, and Hsiang-Pang Li. 2015. Marching-Based Wear-Leveling for PCM-Based Storage Systems. ACM Transactions on Design Automation of Electronic Systems. 20, 2, 25.

[11] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, Jude A. Rivers. 2009(b). Scalable high performance main memory system using phase-change memory technology. ISCA. 24–33.

[12] Sung Kyu Park, Min Kyu Maeng, Ki-Woong Park, et al. 2014. Adaptive wear-leveling algorithm for PRAM main memory with a DRAM buffer. ACM Transactions on Embedded Computing Systems. 13, 4, 88.

[13] Dong-Jae Shin, Sung Kyu Park, Seong-Min Kim, Kyu Ho Park. 2012. Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory, RACS. 395-402.

[14] Soyoon Lee, Hyokyung Bahn, Sam H. Noh. 2011. Characterizing memory write references for efficient management of hybrid PCM and DRAM memory. MASCOTS. 168-175.

[15] Zhi Li, Peiquan Jin, Xuan Su, et al. 2009. CCF-LRU: a new buffer replacement algorithm for flash memory. IEEE Transactions on Consumer Electronics. 55, 3, 1351-1359.

[16] Edward G. Coffman Jr., Peter J. Denning. 1973. Operating Systems Theory. Prentice-Hall. Ch.6, 241-283.

[17] Mei-Ling Chiang, and Ruei-Chuan Chang.1999. Cleaning policies in mobile computers using flash memory. Journal of Systems and Software. 48, 3, 213-231.

[18] Fu-Hsin Chen, Ming-Chang Yang, Yuan-Hao Chang, and Tei-Wei Kuo.2015. PWL: a progressive wear leveling to minimize data migration overheads for nand flash devices. DATE. 1209-1212.

[19] Ping Zhou, Bo Zhao, Jun Yang, Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. ACM SIGARCH Computer Architecture News. 37, 3, 14-23.0

[20] Moinuddin K. Qureshi, John P. Karidis, Michele Franceschini, et al. 2009(a). Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. MICRO. 14-23.

[21] Soojun Im, Dongkun Shin. 2014. Differentiated space allocation for wear leveling on phase-change memory-based storage device. IEEE Transactions on Consumer Electronics. 60, 1, 45-51.

[22] Song Jiang, Xiaodong Zhang. 2002. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. ACM SIGMETRICS Performance Evaluation Review. 30, 1, 31-42.

[23] DiskSim, available at http://www.pdl.cmu.edu/DiskSim

[24] Donald E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, pp. 400, 1973