APP-LRU: A New Page Replacement Method for PCM/DRAM-Based Hybrid Memory Systems

Zhangling Wu, Peiquan Jin, Chengcheng Yang, Lihua Yue

APP-LRU: A New Page Replacement Method for PCM/DRAM-Based Hybrid Memory Systems

Zhangling Wu[1], Peiquan Jin[1, 2], Chengcheng Yang[1], Lihua Yue[1, 2]

[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]Key Laboratory of Electromagnetic Space Information, Chinese Academy of Sciences, China
jpq@ustc.edu.cn

Abstract. Phase change memory (PCM) has become one of the most promising storage media particularly for memory systems, due to its byte addressability, high access speed, and low energy consumption. In addition, hybrid memory systems involving both PCM and DRAM can utilize the merits of both media and overcome some typical drawbacks of PCM such as high write latency and limited lifecycle. In this paper, we present a novel page replacement algorithm called APP-LRU (*Access-Pattern-prediction-based LRU*) for PCM/DRAM-based hybrid memory systems. APP-LRU aims to reduce writes to PCM while maintaining stable time performance. Particularly, we detect read/write intensity for each page in the memory, and put read-intensive pages into PCM while placing write-intensive pages in DRAM. We conduct trace-driven experiments on six synthetic traces and one real OLTP trace. The results show that our proposal is able to reduce up to 5 times of migrations more than its competitors.

Keywords: Phase change memory, Page replacement policy, Hybrid memory

1 Introduction

Recently, the big data concept leads to a special focus on the use of main memory. Many researchers propose to use a large main memory to improve the performance of big data processing. However, the increasing capacity of main memory introduces many problems, such as increasing of total costs and energy consumption [1]. Both academia and industries are looking for new greener memory media, among which the Phase Change Memory (PCM) receives much attention [2]. PCM is one type of non-volatile memories, and provides better support for data durability than DRAM does. Further, it differs from other media such as flash memory in that it supports byte addressability. Because of the unique features of PCM, some people argue that PCM may replace DRAM in the future, as shown in Fig. 1(a). However, PCM has some limitations, e.g., high write latency, limited lifecycle, slower access speed than DRAM, etc. Therefore, it is not a feasible design to completely replace DRAM with PCM in current computer architectures.

A more exciting idea is to use both PCM and DRAM to construct hybrid memory systems, so that we can utilize the advantages from both media [2, 3]. PCM has the advantages of low energy consumption and high density, and DRAM can afford nearly unlimited writes. Specially, PCM can be used to expand the capacity of main memory, and DRAM can be used as either a buffer for PCM, as shown in Fig. 1(b) or

the secondary main memory like DRAM, as shown in figure 1(c). Presently, both the architectures illustrated in Fig. 1(b) and (c) are hot topics in academia and industries. Many issues need to be further explored, among which the most focused issue is the buffer management schemes for hybrid memory systems involving PCM and DRAM. The biggest challenge for PCM/DRAM hybrid memory systems is that we have to cope with heterogeneous media. Traditional management schemes yield some specific page replacement policies that are designed either for DRAM-only main memory or for the system shown in Fig. 1(b). However, in this paper we focus on the hybrid memory systems with the architecture shown in Fig. 1(c).

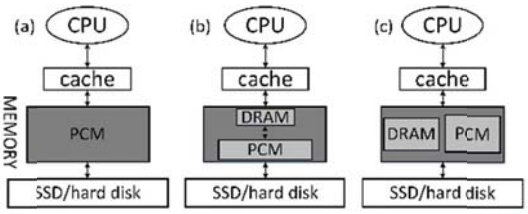

Fig. 1. Architectures of PCM-based memory systems [4]

The objective of this paper is to design an efficient page replacement scheme for PCM/DRAM-based hybrid memory systems as shown in Fig. 1(c). We propose a novel method called APP-LRU (Access-Pattern-Prediction-aware LRU). This method employs an algorithm to predict the access pattern changes and further uses the access patterns to reduce writes to PCM and keep stable time performance for PCM/DRAM-based hybrid memory systems. The main contributions of the paper are summarized as follows:

(1) We present a new page replacement method named APP-LRU for PCM/DRAM-based hybrid memory systems. APP-LRU records the access history of each page using a history table to identify the read and write intensity of pages. As a consequence, read-intensive pages are stored in PCM and write-intensive pages are saved in DRAM. Further, we propose an LRU-based on-demand migration algorithm to move pages between PCM and DRAM. (**Section 3**)

(2) We conduct trace-driven experiments in a simulated PCM/DRAM-based hybrid main memory environment under six synthetic traces and one real OLTP trace, and compare our proposal with several existing methods including LRU, CLOCK-DWF and Hybrid-LRU. The results show that our proposal reduces up to 5 times of total migrations more than its competitors. Meanwhile, it maintains comparable run time in all experiments. (**Section 4**)

2 Related Work

PCM is a kind of alternative memory devices because of its merits such as high density, low idle energy and so on. However, its limited life and long write latency is the main obstacles when implement traditional main memory management policies on PCM-based memory system.

There are many researches focus on reducing redundant writes to PCM, such as enhancing the fine-grained management approach [2, 3] with a Data Comparison Write (DCW) scheme that utilizes the bit alterability feature of PCM and only updates the changed bits [5-7]. However, these works are towards hardware design, while this paper employs a software-based research to reduce PCM writes. Moreover, Using DRAM to gather data writes are also a commonly used method for reducing the total number of writes to PCM [1, 8-11]. In this method, PCM is used as main memory, and thus we have to cope with heterogeneous memories in such hybrid memory systems. This hybrid architecture brings new challenges to buffer management schemes, because traditional page replacement policies mainly focus on improving hit ratios, while new policies for hybrid memory systems have to consider the unique features of different storage media in addition to keeping high hit ratios.

Recently, several page replacement policies have been proposed for PCM/DRAM-based hybrid memory systems. The page replacement policy denoted as the "*Hybrid-LRU*" method proposed by Hyunchul Seok et al. [10] and CLOCK-DWF proposed by Soyoon Lee et al. [11] are based on hybrid PCM/DRAM main memory. Hybrid-LRU monitors the access information of each page, assigns different weights to read and write operations, and predicts page access patterns. After that, it moves write-intensive data to DRAM and moves read-intensive data to PCM. However, inappropriate placement of a page when it is first read into memory will cause additional migrations between PCM and DRAM. The main idea of CLOCK-DWF is placing pages that are going to be updated to DRAM. If the data to be updated is currently stored in PCM, a migration is triggered to move the data from PCM to DRAM, and if DRAM is full at the same time, cold data stored in DRAM will be migrated to PCM. But CLOCK-DWF may cause a lot of unnecessary data migrations between PCM and DRAM since it often causes migrations if a page to be written is in PCM. As a consequence, both Hybrid-LRU and CLOCK-DWF introduce lots of data migrations between PCM and DRAM. This situation will degrade the overall time performance of buffer management schemes, because many additional CPU and memory operations are introduced.

The LRU approach has been widely used in the buffer management for flash memory based data management [14, 15]. Our work differs from these works in that we are mainly towards the architecture shown in Fig. 1(c). There are also some previous works focusing on hybrid storage systems involving flash memory and magnetic disks [16, 17]. These studies are orthogonal to our work, as we concentrate on the memory layer but they focus on the SSD/disk layer shown in Fig. 1.

3 The APP-LRU Method

In this section, we describe the details of APP-LRU. APP-LRU aims for reducing PCM writes but keeping stable time performance. For a tree-structured index, the leaf nodes receive more updates than the internal nodes do. Generally, file data accesses have certain access patterns. On the other side, the access patterns of data are usually stable during a certain time period [12]. This feature is used in our proposal to improve the performance of buffer management.

The overall architecture of the hybrid memory system that APP-LRU is towards is shown in Fig. 2. APP-LRU maintains three lists including one LRU list and two sub-lists (denote as "*List-PCM*" and "*List-DRAM*"). The LRU list is used to maintain the pages in both PCM and DRAM. A page is put in the MRU position of the LRU list when it is accessed. The structures of both *List-PCM* and *List-DRAM* are shown in Fig. 2. All the pages in *List-PCM* are divided into several groups, so are those pages in *List-DRAM*. The pages in the same group of *List-PCM* have the same local-write counts, and the pages in the same group of *List-DRAM* have the same local-read counts. The local-read (or local-write) count is the number of read (or write) operations aggregated since the page is stored in DRAM (or PCM). For example, if a new page is read from disk to PCM, its local-write count and total read/write count is 0, if a page is migrated from DRAM to PCM, its local-write count is reset to 0, but total read/write count does not change. Different groups are ordered by the local-read/write count of the pages in the groups. The pages in the head group among *List-PCM* (*List-DRAM*) have the maximum write (or read) counts. Whenever a page is read from disk or moved from DRAM to PCM (or from PCM to DRAM), it is placed to the tail of *List-PCM* (or *List-DRAM*). When a DRAM page is read or updated, its group will be changed, either from *List_PCM* to *List_DRAM* or vice versa. Basically, APP-LRU employs two algorithms. One is to predict page access patterns and the other is to perform page replacement and migration. The details are described below.

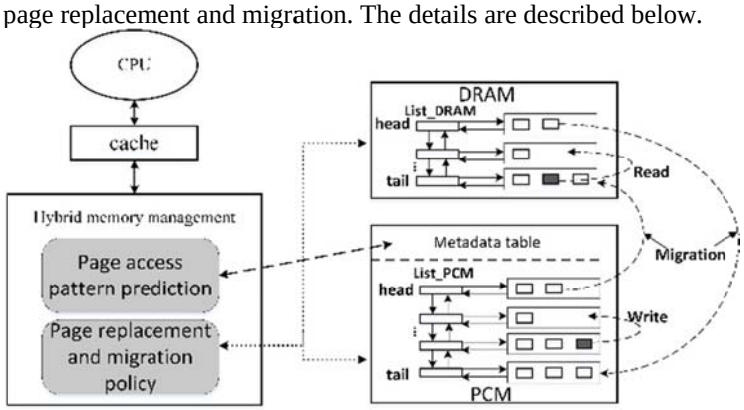

Fig. 2. Overall architecture of hybrid system

3.1 Page Access Pattern Prediction

Unlike DRAM-only memory systems, a hybrid memory system consists of both PCM and DRAM. First, storing data with frequent writes in PCM will introduce the problem of performance degradation, because the write operations to PCM spend much more time than DRAM does. This will also reduce the lifetime of PCM. Second, read and write amplification problems will occur because of data migrations between PCM and DRAM. In order to reduce the number of extra read and write operations caused by migrations, we propose a page access pattern prediction algorithm to predict the future assess patterns of pages.

The basic idea of access pattern prediction is to record the read and write counts for each logical page and then to distinguish read-intensive pages from write-intensive pages. For this purpose, we first maintain some metadata as shown in Fig. 3.

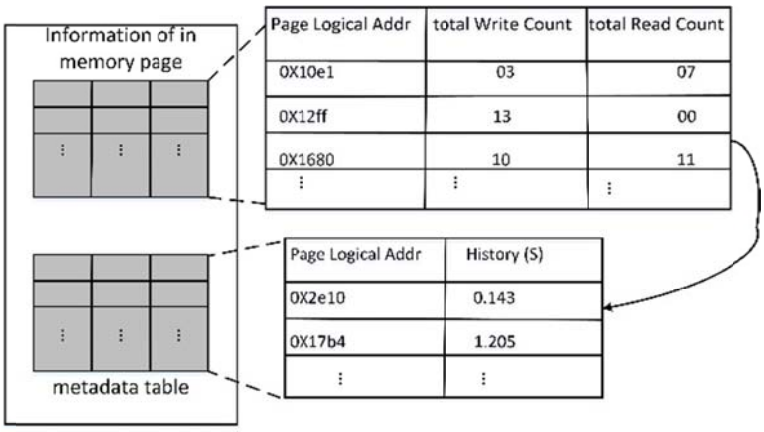

Fig. 3. Metadata for recording page access information

If a page is selected as a victim to be replaced, we process the recorded metadata using Equation (1).

$$S_{cur} = (1-\alpha) \times R / W + \alpha \times S_{pre} = S_{pre} + \beta \times (R / W - S_{pre}), \quad (\beta = 1-\alpha, \quad 0 \le \alpha < 0.5) \tag{1}$$

In Equation (1), R and W are the total counts of read and write operations to the replaced page during its staying in the memory. S_{pre} is the ratio of page's read count to its write count in the past. Therefore, if the S_{pre} of a page exceeds a certain threshold, we regard this page as read-intensive. Otherwise, the page is marked as write-intensive. Since the influence of access histories on the prediction of read and write intensity is becoming weak with time, we introduce a degrading factor α to adjust the influence of access histories to the prediction. On the other hand, the current reads and writes have a big impact on the future access pattern, thus, it is necessary to introduce a factor to reflect the importance of R/W in the prediction, as denoted as β in (1). It is reasonable to set this factor larger than α because of the recency feature in data accesses. In our method, we let $\beta = 1-\alpha$, and α is smaller than 0.5.

Each page's S_{pre} is stored in the metadata table, which is used to decide where to place if the page is accessed again in the future. The metadata table is stored in PCM, and can be found even after power failure accidents. We limit the memory space used for metadata table since the memory capacity is still small compared to disk. However, the concrete capacity of the metadata table is decided based on the actual environment. We also use LRU to manage the metadata table in order to remain relatively hot page access information in the metadata table. In order to alleviate wear out problem, we introduce a small SRAM to buffer metadata table, and the information stored in SRAM will be flush to PCM at set intervals. Since this method aims at logical pages, so we can get the access information from the OS level.

3.2 Page Replacement and Migration

In this section, we present the page replacement and migration procedure of APP-LRU. We maintain a LRU list and two sub-lists. These lists are used to select victims for replacement, as well as to perform page migrations.

When a page fault occurs, the space allocation for the faulted page is based on the access history information in the metadata table (if exists). The page allocation algorithm is shown in **Algorithm 1**. The function *get_free_page*() in Algorithm 1 return a free memory page. If this function is called with a parameter *dram* (or *pcm*), the function will allocate a DRAM (or PCM) page (if exist), but if there is no free DRAM (or PCM) page, the function will allocate a PCM (or DRAM) page (if exist), or allocate the selected victim page from LRU position of LRU list (Before the victim is evicted, we calculate its read/write ratio based on Equation (1), and store the value in metadata table). If the faulted page does not have access histories after looking through the metadata table (Line 1), we call the function without parameter which means faulted page has no specific medium type requirement (Line 17). Otherwise, if the read/write ratio of the faulted page exceeds a certain threshold *R_W_Threshold*, it means that the faulted page probably tends to be read and should be placed in PCM, so we call *get_free_page*() function with parameter *pcm* (Line 4). If the selected victim page is in DRAM, we move the page that is in the head group of List-PCM to the location occupied by the page to be replaced in DRAM (Line 6 ~ 8). Similarly, if the ratio is less than the threshold, it means that the faulted page possibly is write-intensive and should be stored in DRAM (Line 10 ~ 16). As a consequence, we get a free page for accommodating the faulted page.

Algorithm 1: *Page_Allocation*

Input: faulted page addr p
Output: an empty memory page *q*

```
1:  history(p) = get access history of page p in metadata table;
2:  if (history(p) ≠ null) then   /* the page p has been accessed before*/
3:    if (history(p).S_cur > R_W_Threshold) then
4:      q = get_free_page(pcm);
5:      if (q belongs to PCM) then return q;
6:      else   /*q belongs to DRAM*/
7:        select r from the head of List-PCM;
8:        move r to q and insert q to the tail of List-DRAM;
9:        return q=get_free_page(pcm); /*r is empty, r belongs to PCM*/
10:   else
11:     q = get_free_page(dram);
12:     if (q belongs to DRAM) then return q;
13:     else   /*q belongs to PCM*/
14:       select r from the head of List-DRAM;
15:       move r to q and insert q to the tail of List-PCM;
16:       return q=get_free_page(dram); /*r is empty, r belongs to DRAM*/
17:   return q = get_free_page();
```

Next, we explain the page replacement algorithm of APP-LRU, as shown in **Algorithm 2**. If a requested page is not found in memory, we allocate a new space for it using Algorithm 1. We also put the page to the MRU position in the LRU list (Line 1 ~ 5). If the page request is a read request and belongs to DRAM, we increment the read count of the page and adjust the page's position in *List_DRAM*. If the page request is a write request and belongs to PCM, we increment the write count of the page, set a dirty mark, and adjust the page's position in *List_PCM*. (Line 6 ~ 13).

The access pattern of normal data is not likely to change dramatically, so the page's read/write ratio can accurately reflect the access tendency after a long time accumulation based on the theory of statistics. Why we don't choose the read/write ratio as the assessment standard of the migration? It is because the read/write ratio of in memory pages is a short-term computed result, so have no statistical. Even more, the pages that have a similar read/write ratio value may reflect different access frequency, but the warmer page's ratio is much more accurate if the moment when they are read into main memory is close.

Algorithm 2: *Page_Replacement*

Input : page *p* logical address, operation type *op*

```
1:  if (miss) then /* page fault */
2:     q= Page_Allocation(p);
3:     insert p to q and adjust the position of q in LRU list;
4:  else
5:     adjust the position of q in LRU;
6:  if ( op is read) then
7:     read_count(p)++;
8:     if (p is in DRAM) then
9:        adjust the position of q in the List-DRAM;
10: else
11:    dirty(p)=1; write_count(p)++;
12:    if (p is in PCM) then
13:       adjust the position of q in the List-PCM;
```

4 Experimental Results

In the experiments, we use the LRU policy [13] as the baseline method, and also compare two different state-of-the-art approaches including CLOCK-DWF [11] and Hybrid-LRU [10]. Both CLOCK-DWF and Hybrid-LRU are designed for DRAM/PCM-based hybrid memory systems.

4.1 Experimental Setup

We develop a hybrid memory system simulator to evaluate the performance of page replacement policies. The system adopts unified addressing mode, DRAM takes the low-end addresses and PCM takes the high-end addresses. The page size is set to 2 KB. The total size of memory space is constant, and we vary the size of PCM used

in the hybrid memory system ranging from 50% to 86%, which corresponds to the ratio of PCM to DRAM from 1:1 to 1:6 to evaluate the performance.

We use both synthetic and real traces in the experiments, as shown in Table 1. Memory footprint in the table refers to the amount of different pages that the traces reference. There are six synthetic traces used with different localities and read/write ratios. For example, the trace T9182 means that the read/write ratio in this trace is 90% / 10%, i.e., 90% reads plus 10% writes, and the reference locality is 80% / 20%, indicating that 80% requests are focused on 20% pages. The real trace is a one-hour OLTP trace in a bank system and contains 470,677 reads and 136,713 writes to a 20GB CODASYL database (the page size is 2KB).

Table 1. Synthetic and real traces used in the experiments

Trace	Memory Footprint	Read/Write Ratio	Locality	Total Accesses
T9182	10,000	90% / 10%	80% / 20%	300,000
T9155	10,000	90% / 10%	50% / 50%	300,000
T1982	10,000	10% / 90%	80% / 20%	300,000
T1955	10,000	10% / 90%	50% / 50%	300,000
T5582	10,000	50% / 50%	80% / 20%	300,000
T5555	10,000	50% / 50%	50% / 50%	300,000
OLTP	51,880	77% / 23%	~	607,390

4.2 Results on the Synthetic Traces

We use Equation (1) to predict defaulted pages' access patterns. Before we conduct the comparison experiments, we have to first determine the appropriate value of β to minimize the total PCM writes. Fig. 4 shows the total PCM write counts under the T5555 trace when we vary β from 0.5 to 1. It shows an obvious decrease and increase trend of PCM writes when the value of β increases, and the write count is minimized when β is 0.7. Therefore, we set the value of β as 0.7 in the following experiments.

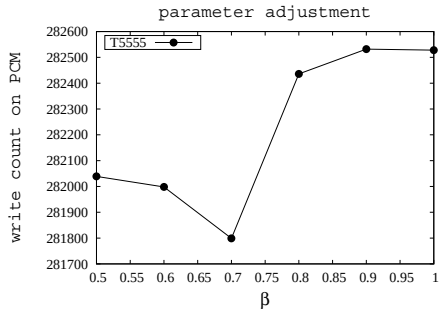

Fig. 4. PCM write counts when varying the parameter β

Figure 5 shows the number of total PCM writes induced by page faults, write operations of traces and migrations between PCM and DRAM. From the figures, we can see APP-LRU reduces maximum 11% total PCM writes with few migrate operations

compared to LRU. This is because that APP-LRU can effectively distinguish write-intensive pages and store them in DRAM, making these pages' write operations take place on DRAM at the beginning. By doing so, it not only eliminates needless migrations but also reduce PCM writes. As APP-LRU has no history information to predict pages' read/write intensity when a page is first accessed, the improvement is limited, but we can get much more reduction as time goes by. The gap of PCM write counts between APP-LRU and LRU increases gradually as PCM/DRAM size ratio increases that means proposed policy perform much better when the PCM/DRAM ratio increases. However, APP-LRU incurs more PCM writes than CLOCK-DWF in most cases, that is because the PCM writes in CLOCK-DWF are only incurred by migration and page fault, and every write operation from workloads only happens on DRAM no matter where the page located in, which will induce a large number of migrations when the write operation is hit in PCM and have a significant effect on memory access latency.

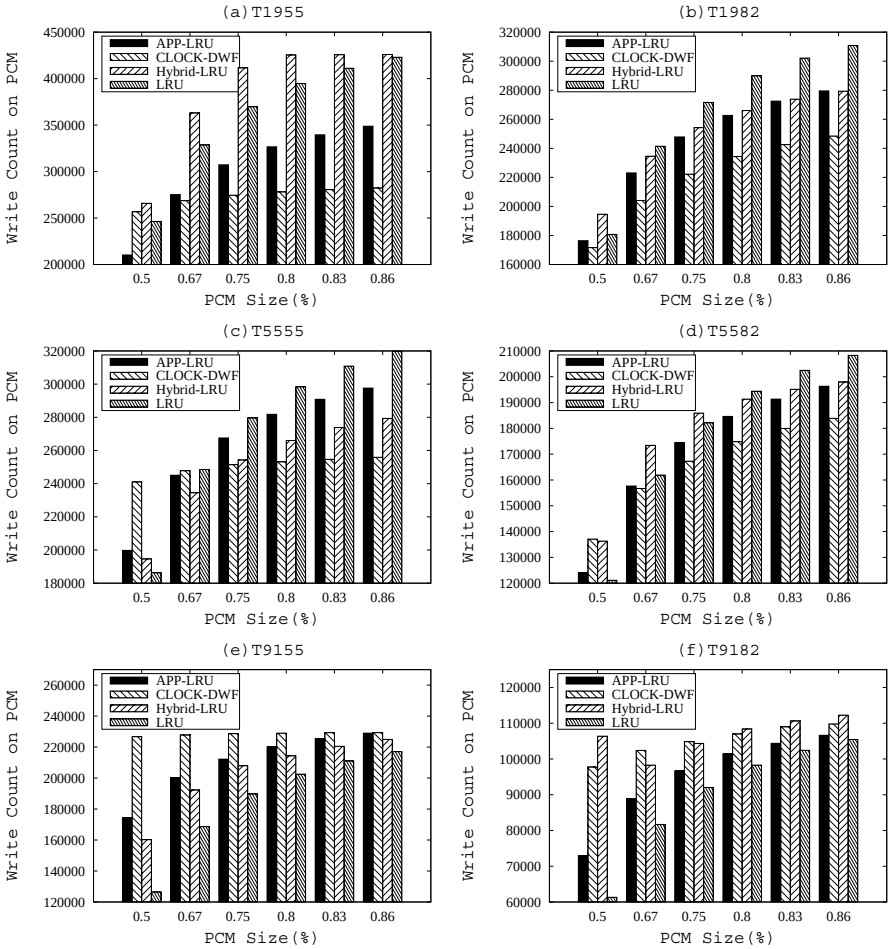

Fig. 5. Total PCM write counts on synthetic traces

Figure 6 shows the total migrations between PCM and DRAM of various replacement algorithms. Figure 5 shows CLOCK-DWF incurs minimum PCM writes compared to others, but Fig. 6 shows that it takes much more migrations in most cases which will introduce extra memory writes and reads. From this figure, both CLOCK-DWF and Hybrid-LRU incur much more migrations in most cases, but APP-LRU reduces nearly up to five times total migrations more than CLOCK-DWF. The migrations of our proposal on T9155 and T9182 are a bit larger than CLOCK-DWF. This is because that the migrations of CLOCK-DWF are only triggered by write operations, but in T9155 and T9182 there are only 10% write operations.

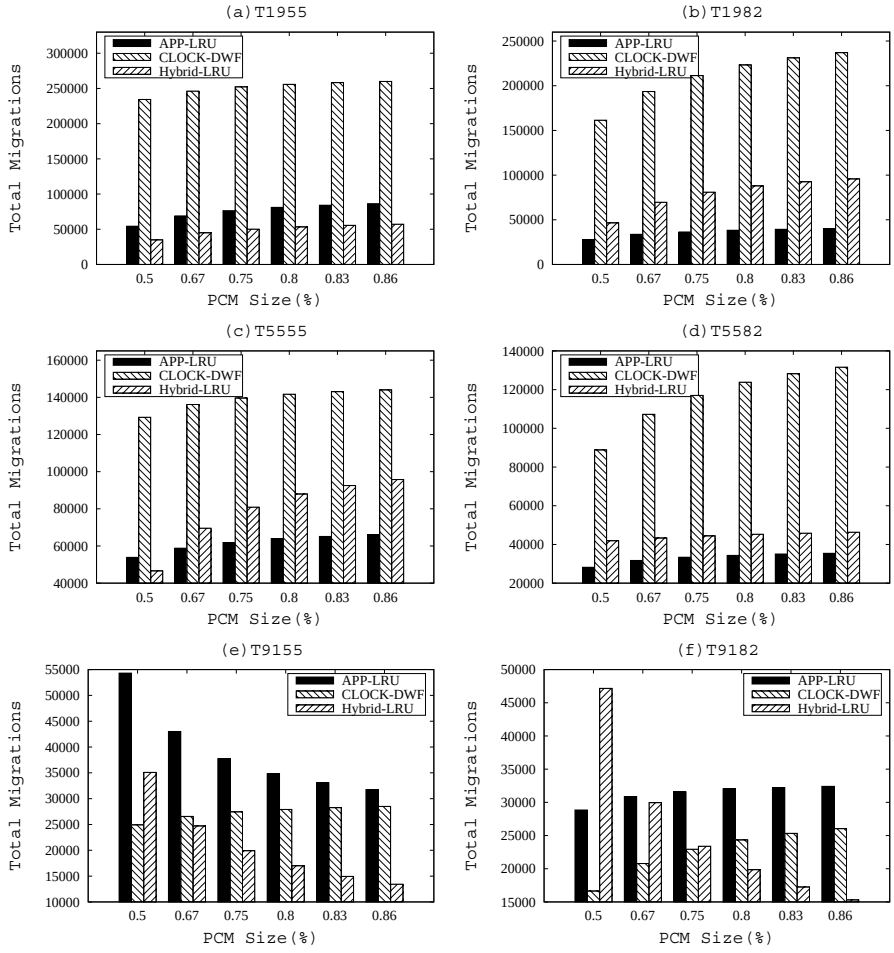

Fig. 6. Total migrations on the synthetic traces

4.3 Results on the Real OLTP Trace

Figure 7 shows the results on the real OLTP trace. The left part of Fig. 7 shows the total writes on PCM. The OLTP trace exhibits a read-incentive pattern and its read locality is much higher compared with write locality. These characteristics make

APP-LRU cannot distinguish the write-intensive page since most pages are read-intensive with only few write operations. From the figure, we still can identify that APP-LRU reduce PCM's writes compared with LRU, which means APP-LRU policy is effective against reducing trace's write operations located on PCM. In conclusion, the APP-LRU algorithm has poor effect on reducing PCM write counts, but is better than both CLOCK-DWF and LRU. Furthermore, APP-LRU still can reduce the total PCM write counts as the size of PCM is larger than DRAM.

The right part of Fig. 7 shows total migrations for real OLTP trace. From the figure, we can see that the migrations of our proposal decrease as the PCM/DRAM ratio augments, and while the migrations of its competitors grow. Our method can reduce average 2 times total migrations against its competitors to reduce writes of PCM, while CLOCK-DWF incurs maximum migrations but cannot obtain any reduction of the total writes of PCM. The total performance of APP-LRU outperforms both Hybrid-LRU and CLOCK-DWF because the miss rate and the large number of migrations of both Hybrid-LRU and CLOCK-DWF are larger than others.

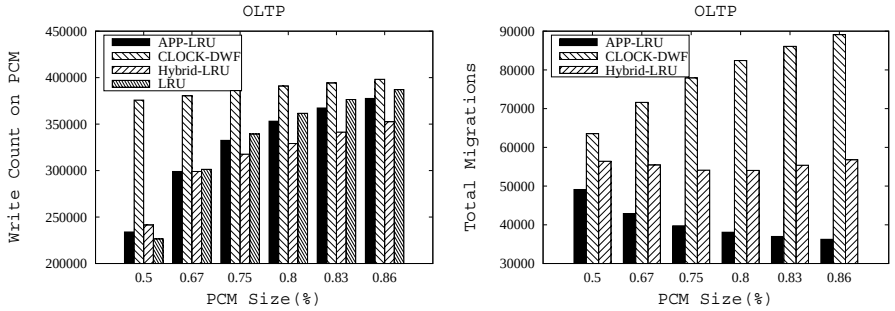

Fig. 7. PCM write counts and total migrations on the real OLTP trace

5 Conclusion

This paper proposes an efficient page replacement policy called APP-LRU for PCM/DRAM-based hybrid memory systems. APP-LRU introduces a metadata table to record the access histories of pages and propose to predict the access patterns of the pages in the memory. Based on the predicted access patterns, either read-intensive or write-intensive, APP-LRU determines to put pages in PCM or DRAM. Through comprehensive experiments on six synthetic traces and one real trace, we demonstrate that our proposal can effectively reduce PCM writes with few migrations.

6 Acknowledgement

This paper is supported by the National Science Foundation of China (No. 61073039, 61379037, and 61272317) and the OATF project funded by University of Science and Technology of China.

7 References

1. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. IEEE Computer, 36(12):39–48, 2003

2. Moinuddin K. Qureshi, Srinivasan Vijayalakshmi, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In Proc. of ISCA, New York: ACM, 2009: 24–33
3. B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. In Proc. of ISCA, New York: ACM, pp. 2-132009.
4. Shimin Chen, Phillip B. Gibbons, Suman Nath. Rethinking database algorithms for phase change memory[C]. Proc. of CIDR, pp. 21-31, 2011
5. Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, et al. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In Proc. of ISCAS. New Orleans, USA, 2007: 3014-3017
6. P. Zhou, B. Zhao, J. Yang, and Y. Zhang, A durable and energy efficient main memory using phase change memory technology. In Proc. of ISCA, New York: ACM, pp.14–23, 2009
7. Sangyeun Cho and Hyunjin Lee. Flip-N-Write: A simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In Proc. of MICRO. New York: ACM, 2009: 347-357
8. Hyunsun Park, Sungjoo Yoo, Sunggu Lee. Power management of hybrid dram/pram-based main memory. In Proc. of DAC. New York: ACM, pp. 59-64, 2011
9. Dong-Jae Shin, S. K. Park, S. M. Kim and K. H. Park. Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory. In Proc. of ACM RACS. New York: ACM, pp. 395-402, 2012
10. H. Seok, Y. Park, K. Park, and K. H. Park. Efficient Page Caching Algorithm with Prediction and Migration for a Hybrid Main Memory. ACM SIGAPP Applied Computing Review 11(4): 38-48, 2011
11. Lee, S., Seoul Bahn, H. Noh S. CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. Computers, IEEE Transactions on, PP(99): 1, 2013
12. Songbin Liu, Xiaomeng Huang, et al. Understanding Data Characteristics and Access Patterns in a Cloud Storage System. In Proc. of CCGrid, pp. 327-334, 2013
13. E.G. Coffman and P.J. Denning, Operating Systems Theory, Prentice-Hall, ch.6, pp.241-283, 1973.
14. P. Jin, Y. Ou, T. Haerder, Z. Li, ADLRU: An Efficient Buffer Replacement Algorithm for Flash-based Databases, Data and Knowledge Engineering (DKE), Elsevier, Vol.72, 83-102, 2012
15. Z. Li, P. Jin, X. Su, K. Cui, L. Yue, CCF-LRU: A New Buffer Replacement Algorithm for Flash Memory, IEEE Trans. on Consumer Electronics, 55(3), 1351-1359, 2009
16. P. Yang, P. Jin, L. Yue, Hybrid Storage with Disk Based Write Cache. In Proc. of DASFAA Workshops 2011, pp. 264-275, 2011
17. P. Yang, P. Jin, S. Wan, L. Yue, HB-Storage: Optimizing SSDs with a HDD Write Buffer. In Proc. of WAIM Workshops 2013, pp. 28-39, 2013