http://dx.doi.org/10.5573/JSTS.2015.15.1.068 JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.15, NO.1, FEBRUARY, 2015

LDF-CLOCK: The Least-Dirty-First CLOCK Replacement Policy for PCM-based Swap Devices

Seunghoon Yoo, Eunji Lee, and Hyokyung Bahn

Abstract—**Phase-change memory (PCM) is a promising technology that is anticipated to be used in the memory hierarchy of future computer systems. However, its access time is relatively slower than DRAM and it has limited endurance cycle. Due to this reason, PCM is being considered as a high-speed storage medium (like swap device) or long-latency memory. In this paper, we adopt PCM as a virtual memory swap device and present a new page replacement policy that considers the characteristics of PCM. Specifically, we aim to reduce the write traffic to PCM by considering the dirtiness of pages when making a replacement decision. The proposed replacement policy tracks the dirtiness of a page at the granularity of a sub-page and replaces the least dirty page among pages not recently used. Experimental results with various workloads show that the proposed policy reduces the amount of data written to PCM by 22.9% on average and up to 73.7% compared to CLOCK. It also extends the lifespan of PCM by 49.0% and reduces the energy consumption of PCM by 3.0% on average.**

Index Terms—**Phase-change memory, swap device, replacement policy, virtual memory, CLOCK**

I. INTRODUCTION

Recently, phase-change memory (PCM) has been drawing considerable interest as a memory medium of future computer systems [1-4]. In particular, PCM is considered as a replacement of DRAM due to its various advantages such as low-power consumption, high density, byte-addressability, and non-volatility [1-3]. However, PCM has weaknesses to become a main memory medium as its access time is relatively slower (about 1-5x read, 5-25x write) compared to DRAM and it has limited write endurance. Hence recently, it is being considered as a high-speed storage medium (like swap device) or long-latency memory that is to be used with a DRAM buffer [1, 3, 4, 11, 12]. This paper presents a new page replacement policy for the system that uses PCM as a swap device.

The primary goal of page replacement policies is to reduce the number of page faults because accessing traditional swap devices (i.e., hard disks) costs several orders of magnitudes larger than accessing DRAM memory. Meanwhile, PCM is known to possess significantly different physical characteristics from hard disks. As a result, PCM specific file systems [11, 12] and PCM specific memory management techniques [1, 3, 4] have been extensively studied. Unlike these studies, research on virtual memory systems that use PCM as a swap device is in its infancy.

As Table 1 shows, the write latency of PCM is 5-25 times slower than that of DRAM and the number of write operations allowed for each PCM cell is also limited to 10^6-10^8 [1, 3]. To cope with this situation, studies on PCM memory use additional DRAM as shown in Figs. 1(a) and (b). Though details of the architectures are different, the role of additional DRAM is commonly to hide the slow write operations of PCM and also increase the lifespan of PCM by absorbing frequent write operations. Studies on PCM specific file systems also

Manuscript received May. 12, 2014; accepted Dec. 27, 2014
A part of this work was presented in Korean Conference on Semiconductors, Seoul in Korea, Feb. 2014
Ewha Womans University
E-mail : bahn@ewha.ac.kr

Table 1. Comparison of memory technologies

	DRAM	PCM
Cell size	*6F2*	*5F2*
MLC	N/A	4
Read Latency	20 ns	20-100 ns
Write Latency	20 ns	100-500 ns
Endurance	10^{16}	10^{6}-10^{8}
Retention	Volatile (64 ms)	Non-volatile (10yr)

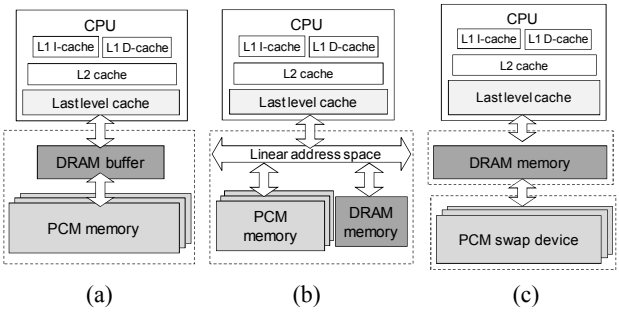


Fig. 1. Different memory architecture with PCM.

aim to reduce write traffic to PCM [11, 12].

This paper reduces the write traffic to PCM-based swap devices by judiciously managing virtual memory page replacement policies. Specifically, we design a novel page replacement policy for virtual memory systems by making use of the fine-grained dirtiness management of a page. Our replacement policy keeps track of the dirtiness of a page (e.g., 4KB) at the granularity of a sub-page (e.g., 256B) and replaces the least dirty page among pages not recently used. This technique, which we call LDF-CLOCK (Least-Dirty-First CLOCK), is effective in reducing the write traffic to PCM significantly by replacing pages incurring small writing.

Moreover, in contrast to the LRU (Least Recently Used) replacement policy that needs to perform list manipulations or time-stamping on every memory reference, LDF-CLOCK does not require neither time-stamping nor list manipulations unless page faults occur. This makes LDF-CLOCK suitable for virtual memory environments that allow OS controls only upon page faults. The other strength of LDF-CLOCK is that it does not degrade the page fault ratio significantly even though it reduces large write traffic to PCM. This is because LDF-CLOCK does not evict recently referenced pages similar to the original CLOCK algorithm.

Experimental results show that LDF-CLOCK reduces the write traffic to PCM by 22.9% on average and up to 73.7% compared to the CLOCK algorithm. We also show that the lifespan of PCM is extended by 49.0% and the energy consumption of PCM is reduced by 3.0% on average by using LDF-CLOCK.

II. REPLACEMENT POLICIES FOR VIRTUAL MEMORY SYSTEMS

Page references in a virtual memory environment have temporal locality in that a more recently referenced page is more likely to be referenced again soon. In terms of the hit ratio, the LRU policy is known to be optimal for references which exhibit this property [13]. LRU aligns all pages in the memory in the order of their most recent reference time, and replaces the least recently used page whenever free page frames are needed. It is the most popular replacement algorithm in various caching environments including file system buffer cache since it performs well but has constant time and space overheads.

Nevertheless, LRU has a critical weakness in virtual memory environments. On every memory reference, LRU needs to move a page to the most recently used position in the list. This involves some list manipulations which cannot be handled by the paging unit hardware. Thus, list implementation of LRU is generally used in file system buffer cache, in which list manipulation by OS is possible on every memory references. Though LRU can also be implemented by hardware, this is not feasible in virtual memory systems as it should maintain the time-stamp of each page and update it upon every memory references. Thus, hardware implementation of LRU is usually adopted in cache memory systems, which have limited associativity.

Due to this reason, in virtual memory systems, CLOCK was introduced as an efficient way to approximate the workings of LRU [14]. Instead of keeping pages in the order of reference time, CLOCK only monitors whether a page has recently been referenced or not with one reference bit per page. Whenever a page is referenced (i.e., read or written), the reference bit is set to 1 by the paging unit hardware. CLOCK resets this bit to 0 periodically to ensure that a page will have reference bit set only if it has been accessed at least once from the duration of the last reset.

To do this, CLOCK maintains pages in a circular list. Whenever free page frames are needed, CLOCK sequentially scans through the pages in the circular list, starting from the current position, that is, next to the position of the last evicted page. This scan continues until a page with a reference bit of 0 is found and that page is then replaced. For every page with reference bit of 1 in the course of the scan, CLOCK clears the reference bit to 0, without removing the page from the list.

The reference bit of each page indicates whether that page has recently been accessed or not; and a page which is not referenced until the clock-hand comes round to that page again is certain to be replaced. Even though CLOCK does not replace the LRU page, it replaces a page that has not been referenced recently, so that temporal locality is exploited to some extent. In addition to this, since it does not require any list manipulation on memory hit, CLOCK is suitable for virtual memory environments.

III. THE LDF-CLOCK POLICY

1. System Architecture

Fig. 1(c) shows the target system architecture of LDF-CLOCK. LDF-CLOCK is adopted as a page replacement algorithm in the main memory layer on top of the PCM swap device. The main memory transfers data to/from LLC (Last Level Cache) in a sub-page (or a cache block) granularity, while communicates with the PCM swap device in a page granularity. In our study, the size of a page is set to 4KB and the size of a sub-page to 512B by which a page consists of 8 sub-pages.

For each page, LDF-CLOCK maintains a *reference bit* and a *valid bit* to indicate whether the page is recently accessed and contains valid data, respectively. A page also needs a *dirty bit* to represent whether it has been modified after entering memory so that changes can be reflected to the swap device when it is evicted. LDF-CLOCK maintains a dirty bit for each sub-page to quantify the expected write traffic by eviction of each page. The dirty bit is set when the LLC cache block is written back to the memory page. The dirty bit of 1 indicates that the sub-page should be flushed to PCM swap device when the page it belongs to is replaced from memory since it has been changed while resident in the memory.

To implement the LDF-CLOCK algorithm, sub-page dirty information should be maintained in each page table entry. As TLB is a cache of the page table, it also needs sub-page dirty information. This necessitates the modification of TLB entries as well as page table entries. This may be a hurdle to adopt LDF-CLOCK in current systems promptly, but as the underlying hardware device is changed from traditional swap storage (i.e. HDD) to new medium (PCM), such modifications are essential to utilize the full advantages of PCM.

2. Algorithm Details

LDF-CLOCK selects a replacement victim based on the state of the reference bit and dirty bits of each page to reduce the number of page faults and the write traffic to PCM simultaneously. LDF-CLOCK sets the reference bit of a page to 1 when the page is accessed, and it also sets the dirty bit of the accessed sub-page when the access is write.

Similar to CLOCK, LDF-CLOCK also uses a clock-hand that traverses in one direction over the circular list of pages. Whenever replacement is needed to accommodate a new page, LDF-CLOCK considers two important metrics: recency and dirtiness. Recency is important for virtual memory workloads, which exhibit strong temporal locality. The reference bit of each page is an indication of recency. In order to reduce the number of page faults, LDF-CLOCK restricts victim candidates for replacement to those pages with the reference bit of 0. To reduce the write traffic to PCM as well, LDF-CLOCK also considers the dirtiness of a page, which is defined as the number of dirty sub-pages within a page.

During the victim selection process, LDF-CLOCK checks the reference bit of the page pointed by the clock-hand. If the reference bit is 1, it is reset to 0, and the clock-hand is advanced to the next page. This step is repeated until the clock-hand comes across a page with the reference bit of 0, and then, stops at that point. Now, the set of all pages, which currently have their reference bit of 0 is considered to be victim candidates. Of these, LDF-CLOCK selects the page with the minimum number of dirty sub-pages as a victim. This policy allows pages referenced recently to remain in memory, while reducing

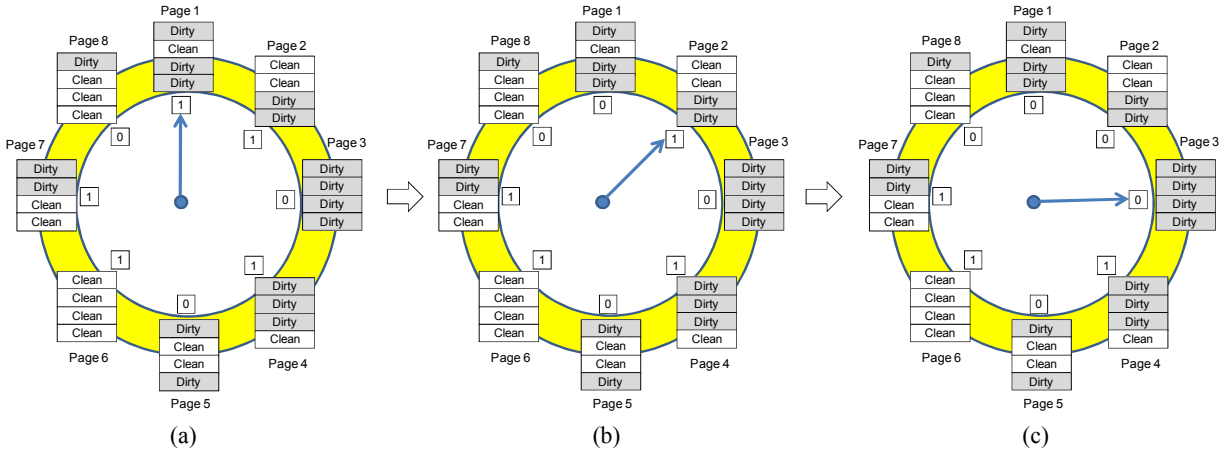


Fig. 2. Working of the LDF-CLOCK policy.

write traffic to PCM by evicting a page that does not incur much PCM writing. If there exist multiple pages that have the same number of minimum dirty sub-pages, the page whose reference bit was least recently reset to 0 is evicted.

As all pages with the reference bit of 0 are victim candidates, a naive implementation of LDF-CLOCK requires the time overhead of O(n), where n is the number of page frames in memory. To eliminate such inefficiency, we use multiple list structures that separate pages with different number of dirty sub-pages. In particular, we use 9 lists, and each list manages pages with the dirty sub-pages of 0 to 8 in the FIFO order. Thus, when the clock-hand comes across a page with the reference bit of 0, LDF-CLOCK replaces the first page in the list with minimum dirty sub-pages. Note that these operations can be performed in a constant time complexity.

Fig. 2 shows the workings of LDF-CLOCK. This example assumes that a page consists of four sub-pages. When replacement is needed, LDF-CLOCK checks the reference bit of the page pointed by the clock-hand (Fig. 2(a)). As the reference bit is 1, it is reset to 0, and the clock-hand is advanced to the next page (Fig. 2(b)). Again, as the reference bit of Page 2 is 1, it is cleared and the clock-hand is advanced to Page 3 (Fig. 2(c)). Now, the clock-hand stops at this location as the reference bit of Page 3 is 0. In this situation, Pages 1, 2, 3, 5, and 8 are considered as candidates for eviction as their reference bit is 0. Of these, Page 8 is finally selected as a victim as it has the smallest number of dirty sub-pages. As a result, LDF-CLOCK reduces write traffic to PCM to one fourth compared to original CLOCK that possibly evicts Page 3 with no awareness of dirtiness.

Table 2. PCM characteristics used in the experiments

Read/Write Latency	50 / 500 (ns/cache block)
Read/Write Energy	0.2 / 1.0 (nJ/bit)
Static Power	0.1 (W/GB)
Endurance	10^7

IV. Performance Evaluation

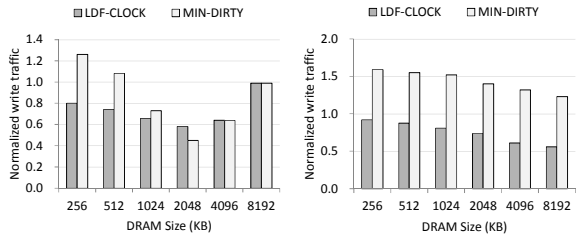
1. Experimental Setup

We now present the performance evaluation results to assess the effectiveness of LDF-CLOCK. A trace-driven simulation is performed to manage the replacement algorithm of a virtual memory system. The size of a page is set to 4KB which is common to most operating systems. The size of a sub-page is set to 512B considering the block size of the last-level cache. The characteristics of PCM used in our experiments are summarized in Table 2 [3, 4, 20].

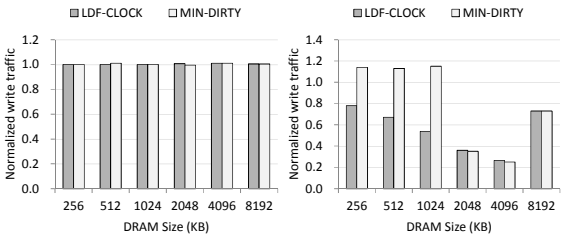
The traces were acquired by a modified version of the Cachegrind tool from the Valgrind 3.2.3 toolset [15]. We capture virtual memory access traces from four different applications used on Linux Xwindows, namely, the freecell game, the kghostview PDF file viewer, the firefox web browser, and the gnuplot graphing utility. We filter out memory references that are accessed directly from the CPU cache memory and also reflect the write-back property of the cache memory. The characteristics of these traces are described in Table 3.

Table 3. Memory usage and reference count for each workload.

workload	footprint (KB)	memory access count		
		I-read	D-read	D-write
freecell	2,521	114,233	315,902	60,040
firefox	8,887	1,886,877	2,848,847	483,677
gnuplot	3,182	3,193	112,600	110,286
kgview	4,347	380,609	1,061,986	103,540



(a) freecell (b) firefox



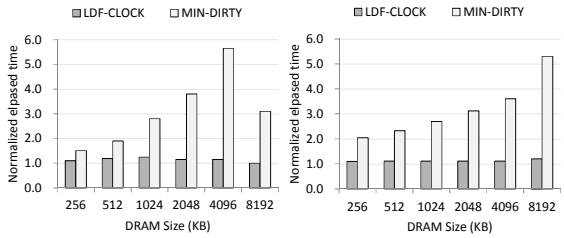
(c) gnuplot (d) kghostview

Fig. 3. Write traffic to PCM for LDF-CLOCK and MIN-DIRTY normalized to CLOCK as the size of main memory is varied.

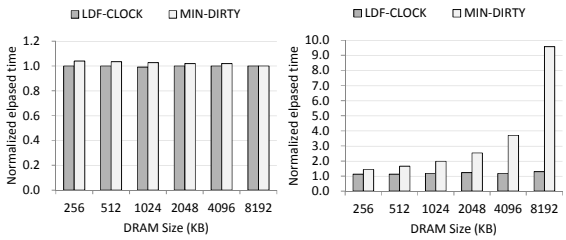
The performance of LDF-CLOCK is compared with CLOCK and MIN-DIRTY. MIN-DIRTY is an algorithm that replaces the page containing minimum number of dirty sub-pages, which is devised for the comparison purpose.

2. Experimental Results

Fig. 3 shows the amount of data written to PCM for LDF-CLOCK and MIN-DIRTY normalized to CLOCK as the memory size is varied. As shown in the figure, LDF-CLOCK significantly reduces the write traffic to PCM compared to original CLOCK for a wide range of memory sizes and a variety of workloads. Specifically, LDF-CLOCK reduces the write traffic to PCM by an average of 22.9% and up to 73.7% compared to CLOCK. The write traffic of MIN-DIRTY is expected to be less than that of LDF-CLOCK as it preserves pages with more dirty data in memory as much as possible. However,



(a) freecell (b) firefox



(c) gnuplot (d) kghostview

Fig. 4. Total elapsed time of LDF-CLOCK and MIN-DIRTY normalized to CLOCK as the size of main memory is varied.

it performs worse than LDF-CLOCK in most cases. Specifically, MIN-DIRTY incurs much more PCM writes than LDF-CLOCK when the DRAM size is small. This implies that dirty pages maintained by MIN-DIRTY are cold pages that do not incur writes again while some evicted pages by MIN-DIRTY must be hot pages which are written again although they are less dirty. This will be quantified more clearly through the total elapsed time and the energy consumption results in the next paragraphs.

Fig. 4 shows the total elapsed time of LDF-CLOCK and MIN-DIRTY normalized to CLOCK. As shown in the figure, the performance degradation of LDF-CLOCK is very small compared to CLOCK though it reduces write traffic to PCM significantly. This is because LDF-CLOCK selects a victim page with the same conditions with respect to the re-reference likelihood compared to original CLOCK. That is, both LDF-CLOCK and CLOCK replace a page only after its reference bit becomes 0. In contrast, the performance of MIN-DIRTY is degraded significantly because it does not consider the reference recency of pages.

Let us now move on to the power issue. Power consumption in PCM can be divided into static and active power consumption. Static power consumption is necessary even when PCM is idle. Due to its non-volatile characteristics, PCM does not need refresh operations,

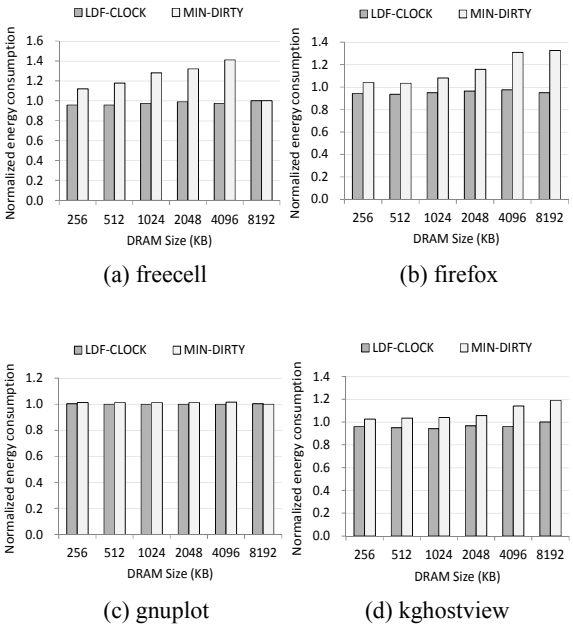


Fig. 5. Energy consumption of LDF-CLOCK and MIN-DIRTY normalized to CLOCK as the size of main memory is varied.

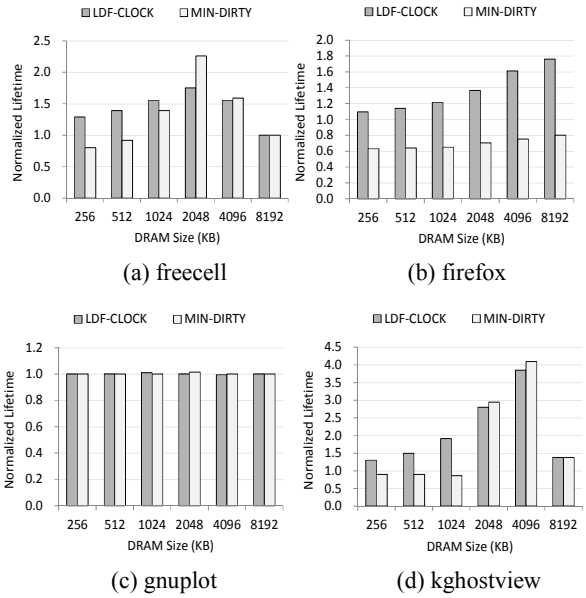


Fig. 6. PCM lifetime for LDF-CLOCK and MIN-DIRTY normalized to CLOCK as the size of main memory is varied.

and thus its static power consumption is very small compared to that of DRAM. Active power consumption refers to the energy dissipated when data is being read and written. In our experiments, total power consumption P_{total} is calculated as

$$P_{total} = P_{static} + P_{active}$$

where

$$P_{static} = Unit\_static\_power\ (W/GB) * memory\_size\ (GB)$$
$$P_{active} = (N_{read} * E_{read}(J) + N_{write} * E_{write}(J)) / (N_{read} * L_{read}(ns) + N_{write} * L_{write}(ns)).$$

Unit_static_power is the static power per capacity, and N_{read} and N_{write} are the number of reads and writes, respectively. E_{read} and E_{write} refer to the energy required for read and write operations, respectively. L_{read} and L_{write} are the average latency of a read and a write operation, respectively.

Fig. 5 shows the energy consumption of PCM when using LDF-CLOCK and MIN-DIRTY normalized to CLOCK. The detailed parameters used in this experiment are listed in Table 3. As shown in the figure, the energy-savings of LDF-CLOCK is 3.0% on average and up to 6.9%, compared to CLOCK. The energy-savings here are small as most workloads are read-intensive.

Finally, we show the effectiveness of LDF-CLOCK with respect to the expected lifetime of PCM. We calculate the expected lifetime of PCM assuming that all writes are equally distributed to PCM. Equal distribution may seem like an unrealistic assumption. However, as we do not deal with the management of PCM, this is a fair approach when we consider only page replacement policies. Furthermore, recent technologies indicate that the wear-leveling of PCM aiming to evenly distribute PCM writes, can be supported within a PCM controller [1, 3]. Hence, we consider this to be a valid assumption. Fig. 6 shows the lifetime of PCM for LDF-CLOCK and MIN-DIRTY normalized to CLOCK. As can be seen, LDF-CLOCK extends the lifetime of PCM by a large margin compared to CLOCK. Specifically, the lifetime is extended by 49.0% on average and up to 279.9%, compared to CLOCK.

V. RELATED WORKS

PCM hardware technology has already reached a certain level of maturity. Specifically, as of 2013, PCM has been commercialized and has been equipped in certain types of smartphones. Patents published recently by Intel describe a detailed micro-architecture to support PCM as memory and/or a storage device, implying that PCM based computer architectures are imminent [16, 17].

Two major PCM manufacturers, Micron and Samsung, are forecasting that the primary interfaces for PCM is likely to be DIMM and PCI-e rather than other block I/O interfaces [18]. This is because existing block I/O interfaces such as SATA or SAS are not fast enough to support high-performance PCM devices, limiting the full advantages that PCM conveys. In addition, numerous activities are underway to re-architect interfaces and the underlying software to maximally utilize the developments that are happening with PCM technologies [19].

Studies on using PCM as main memory have focused on enhancing write performance and endurance of PCM. There are three categories of research that aims to achieve these goals.

The first category uses a certain amount of DRAM to reduce the number of writes that occurs on PCM [1, 4]. Dhiman et al. present a hybrid PCM and DRAM memory architecture called PDRAM [4]. They focus on balancing the write count of PCM by moving data located at a PCM page to a DRAM page if the write count of the PCM page becomes large. However, they do not consider the placement or replacement issues. Qureshi et al. present an architecture that uses DRAM as the last level cache memory of PCM main memory [1]. This architecture caches both clean and dirty pages in DRAM cache. Zhou et al. present cache partitioning and replacement algorithms under this architecture [9]. Their algorithms aim at reducing the cache miss ratio as well as writebacks from the DRAM cache. They also consider the balance of PCM write queues in the design of replacement algorithms. Seok et al. predicts page access patterns and tries to migrate read-intensive pages to PCM and write-intensive pages to DRAM [10]. For prediction of the read and write access pattern, they calculate the weighting value using the ratio of writes to total references. Their research is similar to our research in maintaining data to be written again on non-PCM partitions. However, their algorithm requires exact time information for every reference, which is a requirement difficult to fulfill in virtual memory systems.

The second category is to reduce the number of PCM writes by programming only the cells whose contents have been changed. This technique could enhance the endurance of PCM but it accompanies comparison overhead. Qureshi et al. present the line-level write-back (LLWB) technique that writes only dirty cache lines within a PCM page [1]. To do this, they use a dirty bit within each cache line that retains whether the cache line is modified or not. Yang et al. present the data comparison write that compares each bit in a PCM page and then writes only modified bits [6]. Similar work is also performed by Zhou et al. [5]. Cho and Lee present the flip-n-write technique, which flips all bits in a page if it incurs less number of bit writes [7]. Wongchaowart et al. present a content-aware block placement algorithm that selects a free block among those with similar contents in the free block lists [8].

The third category is the wear-leveling technique to evenly distribute PCM writes. Coarse-grained and fine-grained wear-leveling techniques have been separately studied. For coarse-grained wear-leveling, Zhou et al. present the segment swapping technique that swaps old and young pages periodically [5]. For fine-grained wear-leveling, Zhou et al. present the row shifting technique, which shifts the position of bits in a page in order to balance the number of bit writes within a page [5]. Qureshi et al. present a similar technique called FGWL (fine grained wear-leveling) that stores the lines of each page in PCM in a rotated manner [1].

VI. CONCLUSION

This paper presented a new page replacement policy, called LDF-CLOCK, for PCM swap devices. LDF-CLOCK reduces write traffic to PCM by replacing the least dirty page among pages not recently referenced. We have shown that LDF-CLOCK reduces the write traffic to PCM by 22.9% on average and up to 73.7% compared to CLOCK. We also show that the lifespan of PCM is extended by 49.0% and the energy consumption of PCM is reduced by 3.0% on average compared to CLOCK.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation (NRF) grant funded by the Korea government (MEST) (No. 2011-0028825).

REFERENCES

[1] M. Qureshi, V. Srinivasan, and J. Rivers, “Scalable

high performance main memory system using phase-change memory technology," *Proc. IEEE ISCA Conf.,* pp. 24-33, 2009.

[2] E. Lee, H. Bahn, and S.H. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," *Proc. USENIX FAST Conf.*, pp. 73-80, 2013.

[3] S. Lee, H. Bahn, and S. H. Noh, "CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2187-2200, 2014.

[4] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: a hybrid PRAM and DRAM main memory system," *Proc. ACM/IEEE Design Automation Conf.*, pp.664-559, 2009.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *Proc. IEEE ISCA Conf.*, pp.14-23, 2009.

[6] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu, "A low power phase-change random access memory using a data-comparison write scheme," *Proc. IEEE Symp. Circuit and Syst.*, 2007.

[7] S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," *Proc. IEEE Symp. Microarchitect.*, 2009.

[8] B.Wongchaowart, M. Iskander, and S. Cho, "A content-aware block placement algorithm for reducing PRAM storage bit writes," *Proc. IEEE MSST Conf.*, pp.1-11, 2010.

[9] M. Zhou, Y. Du, B. Childers, R. Melhem, and D. Mosse, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM Trans. Architect. Code Optimization*, vol. 8, no. 4, 2012.

[10] H. Seok, Y. Park, K. Park, and K. Park, "Efficient page caching algorithm with prediction and migration for a hybrid main memory," *Applied Comput. Review*, vol. 11, no. 4, 2011.

[11] E. Lee, J. Jang, T. Kim, and H. Bahn, "On-demand snapshot: an efficient versioning file system for phase-change memory," *IEEE Trans. Knowledge & Data Engineering*, vol. 25, no. 12, pp.2841-2853, 2013.

[12] E. Lee, S. Yoo, J. Jang, and H. Bahn, Shortcut-JFS: a write efficient journaling file system for phase change memory, *Proc. IEEE MSST Conf.*, 2012.

[13] E. Coffman and P. Denning, *Operating Systems Theory*, Prentice-Hall, pp.241-283, 1973.

[14] R. Carr and J. Hennessy, "WSCLOCK—a simple and effective algorithm for virtual memory management," *Proc. ACM SOSP Conf.*, pp.87-95, 1981.

[15] Valgrind, http://valgrind.org/

[16] B. Nale, R. Ramanujan, M. Swaminathan, and T. Thomas, "Memory channel that supports near memory and far memory access," PCT/US2011/ 054421, Intel Corporation, 2013.

[17] R. Ramanujan, R. Agarwal, and G. Hinton, "Apparatus and method for implementing a multi-level memory hierarchy having different operating modes," US 20130268728 A1, Intel Corporation, 2013.

[18] PCM product, http://www.micron.com/products/ phase-change-memory, Micron, 2013.

[19] R. L. Coulson, "Co-optimizing systems, OS, applications, SSDs and NVM," *Proc. Non-Volatile Memories Workshop*, 2012.

[20] E. Lee, H. Bahn, S. Yoo, S. H. Noh, "Empirical study of NVM storage: an operating system's perspective and implications," *Proc. IEEE MASCOTS Conf.*, 2014.

Seunghoon Yoo received the BS degree in computer science from Korea Air Force Academy in 2006, and the MS degree in computer engineering from Seoul National University in 2011, and is currently working toward the PhD degree in computer engineering at Seoul National University. He is also with the Korea Air Force Academy. His research interests include flash memory, storage system, and virtualization software.

Eunji Lee received the PhD degree in computer engineering from Seoul National University in 2012. She was a visiting scholar at the Department of EECS, the University of Michigan, Ann Arbor, and a senior engineer at the Samsung Electronics, Co., Ltd. She is currently an assistant professor in the software department, Chungbuk National University, Korea. Her research interests include operating systems, embedded systems, and storage systems. She has published more than 40 papers in leading conferences and journals in these fields, including IEEE Trans. Computers, IEEE Trans. Knowledge & Data Engineering, and ACM Trans. Storage. She also received the Best Paper Awards at USENIX FAST in 2013.

Hyokyung Bahn received the BS, MS, and PhD degrees in computer science from Seoul National University, in 1997, 1999, and 2002, respectively. He is currently a full professor of computer engineering at Ewha University, Korea. His research interests include operating systems, storage systems, embedded systems, and real-time systems. He received the Best Paper Awards at the USENIX Conference on File and Storage Technologies in 2013.