JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.17, NO.1, FEBRUARY, 2017 ISSN(Print) 1598-1657
https://doi.org/10.5573/JSTS.2017.17.1.007 ISSN(Online) 2233-4866

A Working-set Sensitive Page Replacement Policy for PCM-based Swap Systems

Yunjoo Park and Hyokyung Bahn

***Abstract*—Due to the recent advances in Phage-Change Memory (PCM) technologies, a new memory hierarchy of computer systems with PCM is expected to appear. In this paper, we present a new page replacement policy that adopts PCM as a high speed swap device. As PCM has limited write endurance, our goal is to minimize the amount of data written to PCM. To do so, we defer the eviction of dirty pages in proportion to their dirtiness. However, excessive preservation of dirty pages in memory may deteriorate the page fault rate, especially when the memory capacity is not enough to accommodate full working-set pages. Thus, our policy monitors the current working-set size of the system, and controls the deferring level of dirty pages not to degrade the system performances. Simulation experiments show that the proposed policy reduces the write traffic to PCM by 160% without performance degradations.**

***Index Terms*—Phase-change memory, working-set, replacement policy, virtual memory, CLOCK**

I. INTRODUCTION

Due to the wide speed gap between DRAM and hard disk drives, the primary goal of virtual memory systems has been the minimization of page fault frequency. However, with the recent advances in fast storage technologies such as flash memory and PCM (Phase-Change Memory), the extremely large speed gap has become narrow [1]. The typical access time of flash memory is less than 50 milliseconds, and thus the speed gap between storage and memory is reduced to three orders of magnitude. This trend is accelerated by the emergence of PCM, of which the access time is about 1-5x and 5-25x slower than DRAM in read and write operations, respectively.

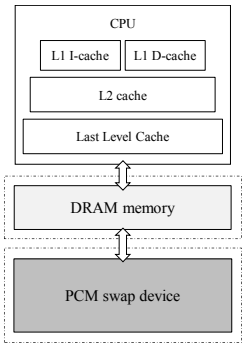


Fig. 1. A DRAM memory and PCM swap architecture.

Actually, PCM was considered as a replacement of DRAM memory due to its various advantages such as low-power consumption, high density, and byte-addressability [2]. However, PCM has weaknesses to substitute DRAM memory in its entirety as its access time is relatively slower compared to DRAM and it has limited write endurance of 10^6–10^8. Thus, PCM is recently considered as a high-speed secondary storage medium as well as far memory that is to be used along with DRAM [3-5].

In this paper, we adopt PCM as a high-speed swap device as shown in Fig. 1 and propose a new page replacement policy for PCM-based swap systems.

Manuscript received Oct. 18, 2016; accepted Feb. 3, 2017
A part of this work was presented in Korean Conference of Semiconductors, Seoul in Republic of Korea, Feb. 2016.
Department of Computer Science & Engineering, EWHA Womans University, Seoul 120-750, Korea
E-mail : bahn@ewha.ac.kr

Although PCM provides high performance and byte-addressability, its write operation is slow and it accommodates only limited endurance cycles. To cope with this situation, our policy tracks the dirtiness of a page at the granularity of a sub-page and defers the eviction of dirty pages in proportion to their dirtiness. To do so, we use a circular list in selecting a victim page and overlook dirty pages although they have not been accessed recently. However, excessive preservation of dirty pages in memory may deteriorate the page fault rate, especially when the memory capacity is not enough. Thus, our policy monitors the current working-set size of the system, and controls the deferring level of dirty pages not to degrade the system performances. Simulation experiments show that the proposed policy reduces the write traffic to PCM by 160% without performance degradations.

The remainder of this paper is organized as follows. Section II explains some background of this research. Section III describes the proposed replacement policy for PCM-based swap systems. Section IV presents the experimental results, and finally Section V concludes the paper.

II. BACKGROUNDS

1. Phase-change Memory Technologies

PCM stores data by making use of a material called GST. GST has two different phases called amorphous and crystalline, which can be set by controlling heating time and temperature. As each state provides different resistance when the electric current is passed, data can be differentiated by reading the resistance on the cell. While detecting the resistive value on the cell is fast, changing the phase in a cell takes longer, and hence a write operation is slower than a read operation in PCM. The endurance limit, that is, the number of writes possible on a PCM cell currently ranges 10^6–10^8. As this limitation is not sufficient for harnessing PCM as main memory, methods to use it in conjunction with a DRAM buffer have been proposed [6]. As a storage device, however, this endurance limitation is not a serious issue when considering the fact that MLC NAND only allows around 10^4 writes per cell.

Comparing memory technologies with respect to capacity, PCM is expected to exceed DRAM, and even NAND flash memory. DRAM is hard to fabricate beyond 20 nm and NAND flash memory has almost reached its scalability limit. In case of NAND flash memory, when the cells are smaller than 20 nm, it is no longer cost-effective as the cost of patterning is too large. Due to these reasons, V-NAND Flash and 3D-DDR3 DRAM attempt to scale the capacity by leveraging the 3D stacking technology rather than scaling down the chip size [7, 8]. In contrast, it is expected that PCM will have stable characteristics in 5 nm node [9]. In 2012, Micron and Samsung announced 45 nm 1 Gb PCM and 8 Gb 20 nm PCM, respectively. Considering that recent NAND flash on the market are on 25 nm cells, it would not be too long before PCM finds its place in storage systems.

In addition to the advances in micro-fabrication processes, multilevel cell (MLC) technologies are also accelerating the density enhancement of PCM. Although most PCM prototypes are being produced as a single-level cell (SLC), which considers only the two crystalline and amorphous states, recent research has demonstrated that additional intermediate states are representable enabling the MLC feature with PCM [9]. MLC can store multiple bits in a cell by choosing multiple levels of electrical charge. This allows PCM to have an order of magnitude higher density than other nonvolatile memory media, such as FeRAM (ferroelectric RAM) and MRAM (magnetic RAM), that are structurally hard to provide MLC. Thus, major semiconductor manufacturers, including Samsung and Intel, are now preparing for the commercialization of PCM technologies.

In reality, PCM hardware technology has already reached a certain level of maturity. Specifically, PCM has been commercialized and also 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 [10, 11]. 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. 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.

2. Replacement Policies for Virtual Memory Systems

Replacement policies for virtual memory systems usually exploit the temporal locality property, which refers to the fact that a more recently accessed page is more likely to be accessed again. In terms of the page fault ratio, the LRU (Least Recently Used) replacement algorithm is known to be optimal for access patterns with this property [12]. LRU sorts all the pages in memory by their last access time, and replaces the oldest page whenever free page frames are needed. It is the most popular replacement policy in various caching systems including file system buffer cache as it performs well but has only constant time and space overheads.

Nevertheless, LRU has a critical weakness in virtual memory systems. On every memory access, LRU needs to move a page to the most recently accessed position in the list. This involves some list manipulations which cannot be handled by the MMU (Memory Management Unit) hardware. Thus, list implementation of LRU is generally used in file system buffer cache, in which list manipulation by operating systems is possible on every accesses. 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 accesses. Thus, hardware implementation of LRU is usually adopted in on-chip cache, which has limited associativity.

Due to this reason, the CLOCK algorithm has been widely used in virtual memory systems to efficiently approximate the workings of LRU [13]. Instead of sorting pages in order of their last access time, CLOCK only monitors whether a page has recently been accessed or not with one reference bit per page. Whenever a page is accessed (i.e., read or written), the reference bit is set to one by the MMU hardware. CLOCK resets this bit to zero periodically to ensure that 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 zero is found and that page is then replaced. For every page with the reference bit of one in the course of the scan, CLOCK clears the reference bit to zero, 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 accessed until the clock-hand comes round to that page again is certain to be replaced. Even though CLOCK does not replace the oldest page, it replaces a page that has not been accessed 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 systems.

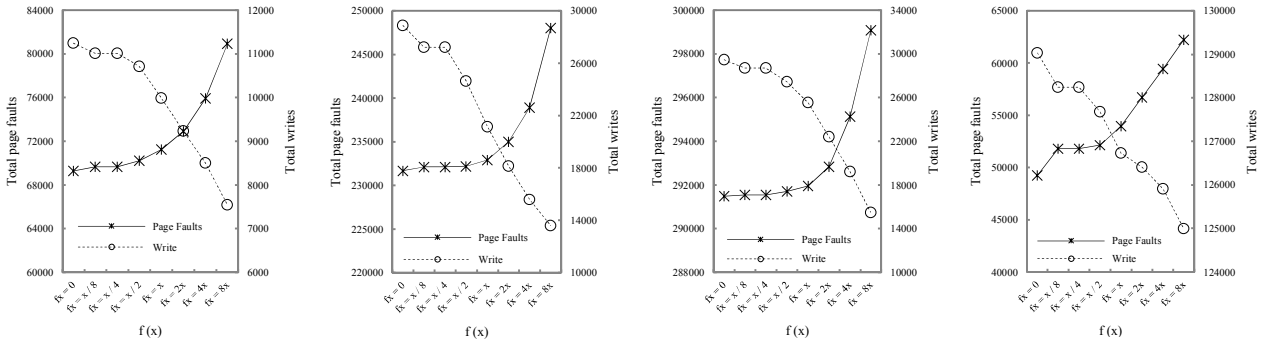
III. A Working-set Sensitive Page Replacement Policy

Fig. 1 shows the target system architecture of the proposed replacement policy called WS-CLOCK (Working-Set Sensitive CLOCK). As shown in the figure, WS-CLOCK is adopted as a page replacement policy in the main memory layer on top of the PCM swap device. Main memory transfers data to/from LLC (Last Level Cache) in a sub-page granularity, while communicates with the PCM swap device in a page granularity.

For each page, WS-CLOCK maintains a *reference bit* to indicate whether the page is recently accessed or not. 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. WS-CLOCK maintains a dirty bit for each sub-page to quantify the expected write traffic by the eviction of each page. The dirty bit is set when an 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 when the page containing this sub-page is replaced from main memory as it has been modified while resident in memory.

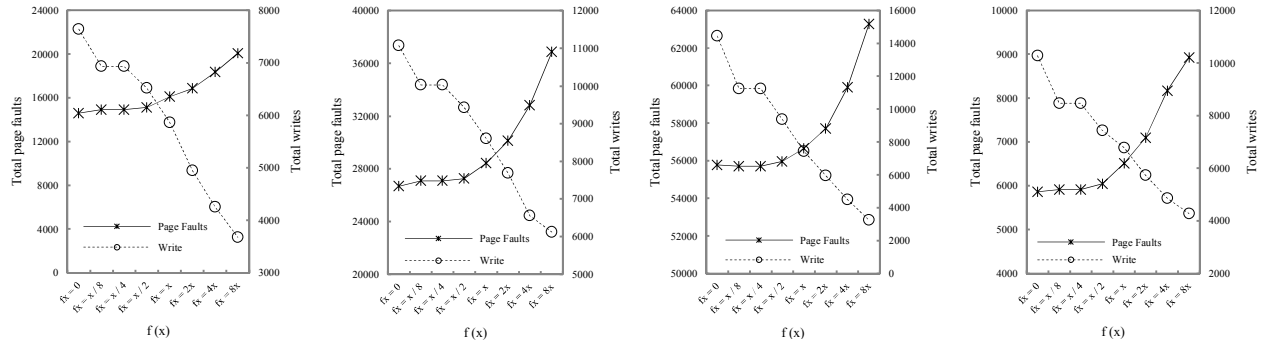
WS-CLOCK selects a replacement victim based on the state of the reference bit and dirty bits of each page to reduce the number of page faults and the write traffic to PCM simultaneously. The reference bit of a page is set to 1 when the page is accessed, and the dirty bit of an accessed sub-page is set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

Similar to CLOCK, WS-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, WS-CLOCK checks the *reference bit* and the *dirtiness* of the page pointed by the



(a) freecell (b) gedit (c) kghostview (d) xmms

Fig. 2. Total page faults and total writes as $f(x)$ is varied (memory size is set to 10% of footprint).



(a) freecell (b) gedit (c) kghostview (d) xmms

Fig. 3. Total page faults and total writes as $f(x)$ is varied (memory size is set to 30% of footprint).

clock-hand. (Note that the dirtiness of a page is defined as the number of dirty sub-pages within the page.) If the reference bit is 1, WS-CLOCK resets it to 0, and the clock-hand is advanced to the next page. Otherwise, WS-CLOCK investigates whether the page is dirty. In WS-CLOCK, a dirty page can be deferred to be evicted for a certain number of clock cycles even though the page's reference bit is 0. To do this, WS-CLOCK maintains the deferring count of each page. If it does not exceed the deferring level given to the page, the clock hand is advanced to the next page. This step is repeated until WS-CLOCK finds a victim page.

WS-CLOCK assigns different deferring levels to dirty pages depending on their dirtiness. That is, a dirtier page has a higher deferring level. Let x be the number of dirty sub-pages in a page. Then, the function f defines the deferring level of a dirty page in proportion to the dirtiness of the page. We define f as a monotonic increasing function to maintain dirtier pages longer in memory.

However, excessive preservation of dirty pages in memory may deteriorate the page fault rate, especially when the memory capacity is not enough to accommodate full working-sets. To find an appropriate deferring level for a given memory status, we perform some preliminary experiments. The experiments are conducted with the virtual memory access traces of four Linux applications: freecell a game, gedit a text editor, kghostview a PDF file viewer, and xmms a music player. Characteristics of each trace are listed in Table 1. In the experiments, the page size is set to 4 KB and the sub-page size is set to 512 bytes.

Fig. 2 and 3 show the total number of page faults and the total write traffic to PCM as the function f is varied. The memory size is set to 10% and 30% of the memory footprint for each workload, respectively. As shown in the figures, a higher deferring level reduces the total write traffic to PCM while the performance is degraded.

Table 1. Characteristics of each workload

Workload	Memory Usage(KB)	Memory Reference Frequency		
		Total	Read	Write
freecell	10,080	490,175	430,135	60,040
gedit	14,460	1,733,763	1,600,941	132,822
kghostview	17,390	1,546,135	1,442,595	103,540
xmms	8,050	1,168,939	190,697	978,242

This indicates that there is a trade-off relation between the total number of page faults and the total write traffic to PCM. This implies that the optimal deferring level cannot be simply fixed since it depends on workloads and system status. Thus, our policy monitors the current working-set size of the system, and controls the deferring level of dirty pages not to degrade the system performances. For example, when the current working-set size of the system is large, WS-CLOCK keeps the deferring level low to focus on reducing page faults for the overall system performances. In contrast, when the available memory space in the system is sufficient (i.e., small working-set), WS-CLOCK maintains a high deferring level to focus on reducing write traffic to PCM.

The system status can be monitored by the difference of page fault rates between the current system and a system with some additional memory. Though this is not a simple problem in online memory management, we can estimate it by the Belady's lifetime function, which is well-known for approximating the hit ratio of references as the memory size is varied. In this paper, we modify the original function to model the page fault rate of the system. With the memory size i, the page fault A_i can be estimated by

$$A_i = c * i^{-k} \quad (1)$$

where c and k are control parameters. The control parameters determine the degree of temporal locality; as c becomes small or k becomes large, the degree of temporal locality increases. Algorithm 1 shows each step of WS-CLOCK.

Algorithm 1 Working-Set Sensitive CLOCK

```
1:   p is the page pointed by clock-hand
2:   procedure WS-CLOCK( )
3:     while reference_bit(p) or !IS_VICTIM(p) do
4:       if reference_bit(p) is 1 then
5:         reference_bit(p) = 0;
6:       else
7:         deferring_count(p) ++;
8:       end if
9:       advance the clock-hand;
10:    end while
11:
12:  procedure IS_VICTIM(p)
13:  Expiration = # of dirty sub-pages * coefficient
14:  if deferring_count(p) > Expiration then
15:    return 1;
16:      else return 0;
17:      end if
```

IV. PERFORMANCE EVALUATIONS

In this section, we present the performance evaluation results to assess the effectiveness of WS-CLOCK. We compare WS-CLOCK with CLOCK, CAR, CART, CFCLOCK, CRAW, and CLOCK-W. CLOCK is a traditional policy that considers the recency of references to select a victim page. CAR considers both recency and frequency of references by making use of two different CLOCK lists and adjusts their sizes through virtual pages; CART improves CAR by applying temporal filtering [14]. CFCLOCK also uses a CLOCK list to capture recency but it preferentially evicts clean pages for a certain range of the list in order to reduce write traffic to storage [15]. CRAW separately maintains CLOCK lists for read and write operations to consider the different eviction cost of the two operations [16]. CLOCK-W behaves identical to CLOCK except for the use of dirty bits instead of reference bits in order to consider the recency of write operations [3]. We can classify the aforementioned policies into two groups. One including CFCLOCK, CLOCK-W, CRAW, and WS-CLOCK takes into account whether a page has been modified or not after entering memory. That is, these policies consider the asymmetric eviction cost of dirty pages that should be written to storage before their eviction and clean pages that can simply be discarded. The other group including CLOCK, CAR, and CART does not consider the eviction cost when selecting a replacement victim but focuses only on the re-reference likelihood of pages.

1. Experimental Setup

Traces used in our experiments were extracted by the Cachegrind tools of Valgrind 3.2.3 toolset [17, 18]. We capture the virtual memory access traces from four applications used on Linux Xwindows, namely, the freecell game, the gedit text editor, the kghostview PDF file viewer, and the xmms music player. 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 1. In the experiments, the page size is set to 4 KB and the sub-page size is set to 512 bytes by which a page is composed of 8 sub-pages.

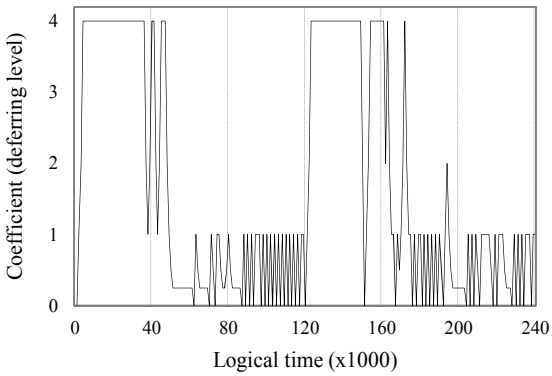
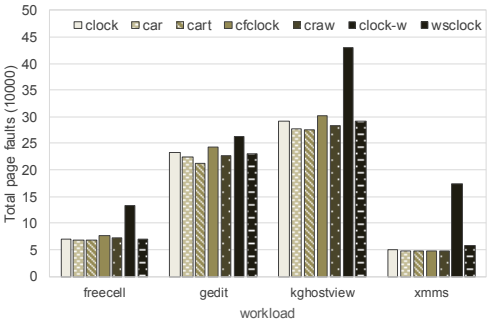
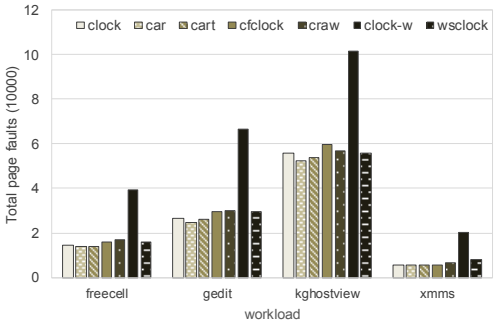


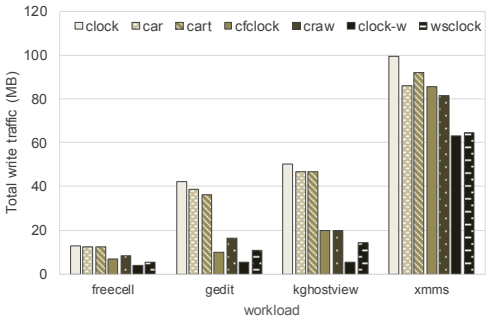
Fig. 4. Coefficient for the deferring level as the system status changes.



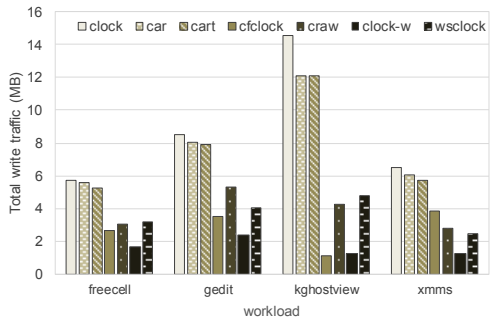
(a) Total page faults under 10% memory



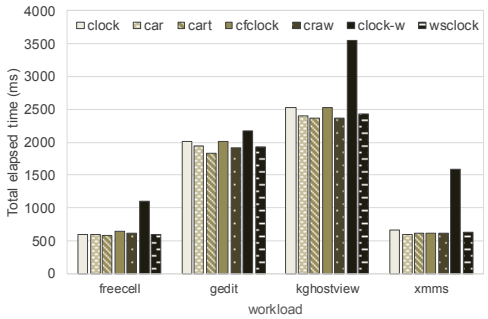
(b) Total page faults under 30% memory



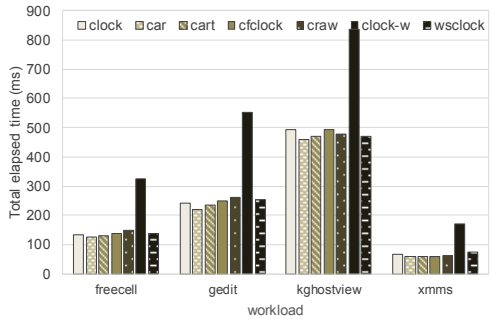
(c) Total write traffic under 10% memory



(d) Total write traffic under 30% memory



(e) Total elapsed time under 10% memory



(f) Total elapsed time under 30% memory

Fig. 5. Comparison of replacement policies with respect to page faults, write traffic, and total elapsed time.

2. Experimental Results

Fig. 4 shows the coefficient of the function *f* as time progresses and the system status changes. In this experiment, the total memory size is set to 20% of the smallest footprint among running applications. We control the working-set size by varying the number of running applications to 1, 2, 3, 1, 4, and 4 as time progresses. As shown in the figure, the deferring level becomes low as the working-set size becomes large and vice versa.

Fig. 5 shows the number of page faults, write traffic, and elapsed time for different replacement policies when the memory size is 10% and 30% of the total memory footprint, respectively. 10% represents the situation that the system does not have enough memory when considering the applications' footprint. In contrast, 30% represents a relatively large memory capacity. As shown in the figure, WS-CLOCK reduces write traffic to PCM significantly in comparison with CLOCK, CAR, and CART that do not consider write references. This is because WS-CLOCK delays the eviction of dirty pages aggressively. CFCLOCK, CRAW, and CLOCK-W also take into account the dirtiness of pages, and thus they reduce write traffic similar to WS-CLOCK. However, they do not consider the dynamic change of system situations, and thus they increase page fault counts when the memory size is not sufficient. This also leads to the degradation of total elapsed time. WS-CLOCK resolves this problem by monitoring the system status and controlling the deferring level of dirty pages adaptively.

Another noteworthy result in our experiments is that CLOCK-W reduces much more write traffic than WS-CLOCK. This is because CLOCK-W checks the reference recency of pages by dirty bits instead of reference bits in order to estimate future write operations, thereby maintaining pages likely to be re-written in memory aggressively. However, it significantly increases page faults as it ignores read operations.

In summary, WS-CLOCK reduces write traffic to PCM by 160% on average and up to 286% in comparison with CLOCK.

VI. CONCLUSIONS

In this paper, we proposed a working-set sensitive page replacement policy, called WS-CLOCK, for the PCM-based swap systems. WS-CLOCK reduces write traffic to PCM by deferring the eviction of dirty pages in proportion to their dirtiness and controls the deferring level depending on the system status. We showed that WS-CLOCK reduces write traffic to PCM by an average of 160% and up to 286% compared to CLOCK without performance degradations.

ACKNOWLEDGMENT

This work was supported by the Basic Science Research program through the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2016R1A2B4015750).

REFERENCES

[1] E. Lee and H. Bahn, "Caching Strategies for High Performance Storage Media," *ACM Transactions on Storage*, Vol. 10, No. 3, 2014.

[2] P. Zhou, B. Zhao, J. Yang, Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Computer Architecture News*, Vol. 37, No. 3, pp. 14-23, 2009.

[3] Y. Park and H. Bahn, "Management of Virtual Memory Systems under High Performance PCM-based Swap Devices," Proc. of *the 39th IEEE Computer Software and Applications Conference (COMPSAC)*, Vol. 2, 2015.

[4] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, Vol. 37, No. 3, pp. 24-33, 2009.

[5] E. Lee, J. E. Jang, T. Kim, and H. Bahn, "On-demand snapshot: An efficient versioning file system for phase-change memory," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, No. 12, pp. 2841-2853, 2013.

[6] 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 Transactions on Computers*, Vol. 63, No. 9, pp. 2187-2200, 2014.

[7] C, Weis, N. Wehn, L. Igor, and L. Benini, "Design space exploration for 3d-stacked drams," Proc. of *Design Automation & Test in Europe*, pp.1-6, 2011.

[8] J. Elliot, and E. S. Jung, "Ushering in the 3D Memory Era with V-NAND," Proc. of *Flash Memory Summit*, pp. 1-14, 2013.

[9] C. D. Wright et al., "Can We Reach Tbit/sq.in. Storage Densities with Phase-Change Media?" Proc. of *European Phase Change and Ovonics Symposium (EPCOS)*, 2006.

[10] B. Nale, R. K. Ramanujan, M. P. Swaminathan, T. Thomas, and T. Polepeddi, "Memory Channel that Supports near Memory and Far Memory Access," US 9342453, Intel Corporation, 2013.

[11] R. K. Ramanujan, R. Agarwal, and G. J. Hinton, "Apparatus and Method for Implementing a Multi-level Memory Hierarchy Having Different Operating Modes," US 20130268728 A1, Intel Corporation, 2013.

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

[13] R. W. Carr and J. L. Hennessy, "WSCLOCK—a simple and effective algorithm for virtual memory management," Proc. of *the 8th ACM SIGOPS Operating Systems Review*, Vol. 15, No. 5, pp.87-95, 1981.

[14] S. Bansal and D. S. Modha, "CAR: Clock with Adaptive Replacement," Proc. of *the 3rd USENIX Conference on File and Storage Technologies (FAST)*, pp. 187-200, 2004.

[15] S, Park, D. Jung, J. Kang, J. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," Proc. of *the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 234-241, ACM, 2006.

[16] H. Lee and H. Bahn, "Characterizing virtual memory write references for efficient page replacement in NAND flash memory," Proc. of *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 1-10, 2009.

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

[18] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," *Electronic Notes in Theoretical Computer Science*, Vol. 89, No. 2, 2003.

Yunjoo Park received the BS degree in computer science and engineering from Ewha Womans University in 2015. She is currently a MS candidate of computer science and engineering at Ewha Womans University, Korea. Her research interests include operating systems, storage systems, embedded systems, and real-time systems.

Hyokung 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, Republic of 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.