JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.15, NO.5, OCTOBER, 2015 ISSN(Print) 1598-1657
http://dx.doi.org/10.5573/JSTS.2015.15.5.476 ISSN(Online) 2233-4866

Efficient Management of PCM-based Swap Systems with a Small Page Size

Yunjoo Park and Hyokyung Bahn

Abstract—**Due to the recent advances in non-volatile memory technologies such as PCM, a new memory hierarchy of computer systems is expected to appear. In this paper, we explore the performance of PCM-based swap systems and discuss how this system can be managed efficiently. Specifically, we introduce three management techniques. First, we show that the page fault handling time can be reduced by attaching PCM on DIMM slots, thereby eliminating the software stack overhead of block I/O and the context switch time. Second, we show that it is effective to reduce the page size and turn off the read-ahead option under the PCM swap system where the page fault handling time is sufficiently small. Third, we show that the performance is not degraded even with a small DRAM memory under a PCM swap device; this leads to the reduction of DRAM's energy consumption significantly compared to HDD-based swap systems. We expect that the result of this paper will lead to the transition of the legacy swap system structure of "large memory – slow swap" to a new paradigm of "small memory – fast swap."**

Index Terms—**Phase-change memory, swap system, paging size, virtual memory**

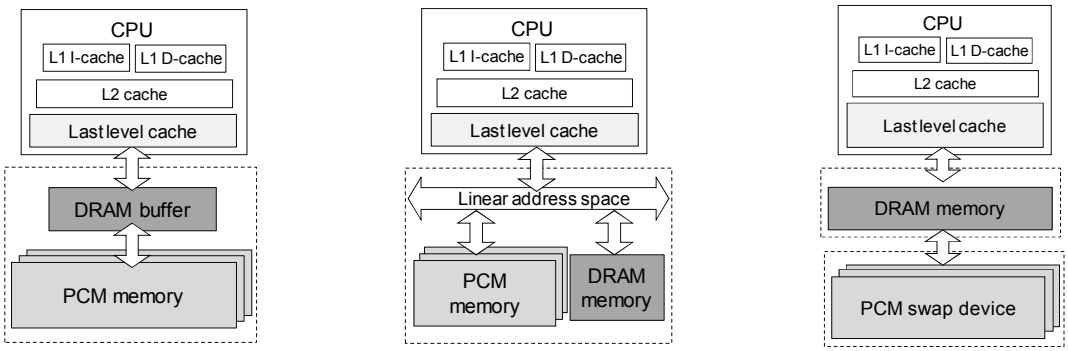
I. INTRODUCTION

To alleviate the widening speed gap between processors and secondary storage, large DRAM memory is used in modern computer systems. In particular, due to the surge of memory-intensive applications and the advent of multi-core technologies, the demand of DRAM grows even rapidly. The enlarged memory capacity seriously increases the energy consumption of computer systems as DRAM memory cells need consistent refresh operations to retain data. Such situations become even worse as the fabrication of DRAM technology reaches its limit at 20nm, and thus it is difficult to enhance the density of DRAM any longer.

As a solution of DRAM's energy consumption and density limit, next-generation memory technologies such as PCM (Phase Change Memory) and STT-MRAM (Spin Torque Transfer Magnetic RAM) have been drawing considerable interest [1-5]. Specifically, PCM hardware technology has already reached a certain level of maturity. As of 2013, PCM has been commercialized and has been equipped in certain types of smartphones [6]. 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 [8, 20].

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 [6, 21]. 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 [22].

Manuscript received Apr. 16, 2015; accepted Jun. 8, 2015
A part of this work was presented in Korean Conference on Semiconductors, Seoul in Korea, Feb. 2015.
Ewha Womans University
E-mail : bahn@ewha.ac.kr



(a) PCM memory with DRAM buffer (b) DRAM-PCM hybrid memory (c) PCM as a swap device

Fig. 1. Different memory architecture with PCM

PCM has been considered as a replacement of DRAM memory due to its various advantages such as low-power consumption, high density, and byte-addressability [1, 2, 4]. However, PCM has weaknesses to substitute DRAM memory in its entirety as its access time is relatively slower (about 2-5x read, 8-50x write) compared to DRAM and it has limited write endurance of 10^7–10^8. To cope with this situation, studies on PCM memory [4] use additional DRAM as shown in Fig. 1(a) and (b). Though details of the architectures are different, the role of additional DRAM memory is commonly to hide the slow write operations of PCM and also to increase the lifespan of PCM by absorbing frequent write operations.

More recently, PCM is being considered as a high-speed storage medium (like swap device) as well as far memory that is to be used along with DRAM [1, 4, 5, 18, 19]. In this paper, we adopt PCM as a high-speed swap device as shown in Fig. 1(c) and investigate the management issues and potential benefits of PCM swap storage. In the traditional HDD-based swap systems, a large amount of adjacent data are loaded together when a page fault occurs since the seek time of a hard disk drive is too large. To improve the CPU utilization, the process that incurs a page fault becomes blocked and the CPU is switched to other process while handling the page fault. However, this may not be efficient for a PCM-based swap system that is sufficiently fast and does not have seek time.

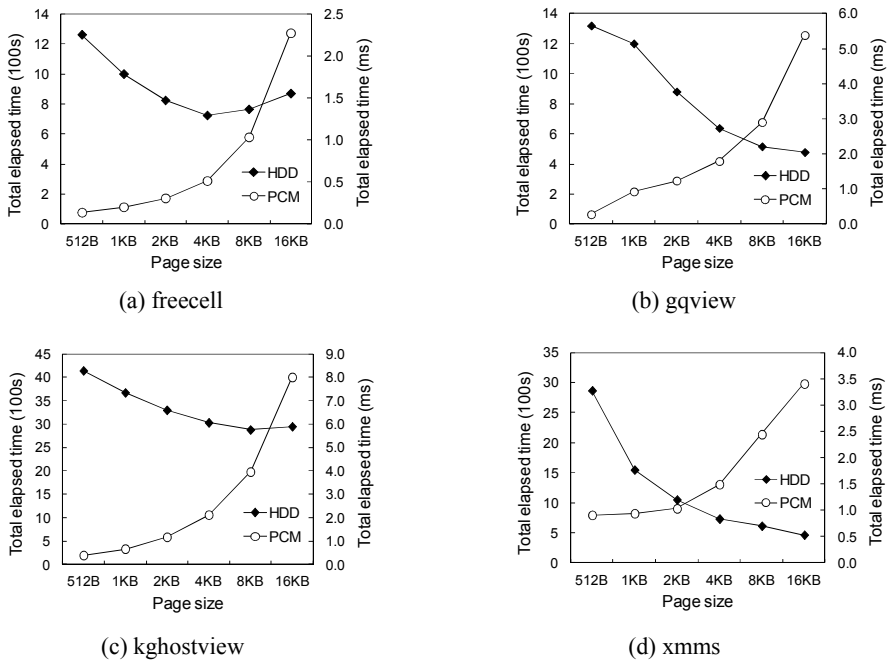
In this paper, we explore the performance of PCM-based swap systems and discuss how this system can be managed efficiently. Specifically, we introduce three management techniques for PCM-based swap systems. First, we reduce the page fault handling time by attaching PCM on DIMM slots and eliminate the software stack overhead of block I/O; we also improve performance by omitting the context switch while handling the page fault. Second, we show that it is effective to reduce the page size and turn off the read-ahead option under the PCM swap system where the page fault handling time is significantly small. Third, we show that the performance is not degraded even with a small DRAM memory size under a PCM swap device; this leads to the reduction of DRAM's energy consumption significantly compared to HDD-based swap systems.

We expect that the result of this paper will lead to the transition of the legacy swap system structure of "large memory – slow swap" to a new paradigm of "small memory – fast swap."

The remainder of this paper is organized as follows. Section II briefly describes the motivation of this research. Section III shows the results of experiments performed in this paper and analyzes them. Section IV summarizes researches related to this study. Finally, Section V concludes this paper.

II. MOTIVATIONS

In this section, we perform motivational experiments to analyze the effectiveness of PCM-based swap systems. The experiments were conducted with the virtual memory access trace of four Linux applications, namely freecell, gqview, kghostview, and xmms. (Details of the workload characteristics will be described in Section III.) For a comparison purpose, we also measure the performance of HDD-based swap systems as well as that of the PCM-based swap system.



(a) freecell (b) gqview

(c) kghostview (d) xmms

Fig. 2. Total elapsed time of HDD and PCM swap systems as a function of the page size

Fig. 2 shows the total elapsed time of each workload as the page size of the target system is varied. For readers' convenience, we use different scales for PCM and HDD in the *y*-axis.

As shown in the figure, the performance of the PCM-based swap system is improved significantly as the page size becomes smaller in all cases. This result is obviously different from the HDD-based swap system case, in which the best performance is obtained when the page size is 4 KB or more. This indicates that the page size of 4 KB, which is most commonly used in modern computer systems, is not efficient for the PCM-based swap system.

However, reducing the page size is not a simple issue since the number of page table entries should be increased as the page size shrinks. This would eventually increase the memory requirement for the page table dramatically. However, it is likely that the total main memory capacity can be reduced when a small page size is used because a variety of contents can be accommodated even with a small DRAM capacity, and the page fault handling time is fast enough under the PCM storage. Therefore, we argue that the space overhead of the page table can be relieved by shrinking the total DRAM size of the system. Through our experiments, we show that our system consisting of small memory and fast swap storage exhibits competitive performance and is also energy-efficient.

III. PERFORMANCE EVALUATIONS

In this section, we perform various experiments to find efficient management techniques for PCM-based swap systems.

1. Experiment Setup

Traces used in our experiments were extracted by the Cachegrind tools of Valgrind 3.2.3 [23]. We capture the virtual memory access traces from four applications used on Linux Xwindows, namely, the freecell game, the kghostview PDF file viewer, the gqview image editing program, 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.

With these traces, we perform simulation experiments to evaluate the performance of PCM-based swap systems. We assume that PCM is put on DIMM slots and there is no context switch while handling page faults. In the simulation, the size of DRAM memory is varied from

Table 1. Memory usage and reference count for each workload.

workload	memory usage (KB)	ratio of operations	memory access count			
			total	instruction read	data read	data write
xmms	8,050	reads : writes = 1 : 5.13	1,168,939	65,048	125,649	978,242
gqview	7,430	reads : writes = 1 : 1.30	610,685	93,242	172,044	345,399
freecell	10,080	reads : writes = 7.16 :1	490,175	114,233	315,902	60,040
kghostview	17,390	reads : writes = 13.93 : 1	1,546,135	380,609	1,061,986	103,540

5% to 100% of the total memory footprint of each workload. The DRAM size of 100% implies the infinite memory capacity where all pages referenced in the trace can be loaded simultaneously, and thus replacement is not needed. This is an unrealistic condition but presented to show the complete performance trend while varying the memory size. The page size is also varied from 512 bytes to 128 KB. We use 512 bytes as the minimum page size because the page size cannot be smaller than the cache block size managed in the last-level cache (LLC). The CLOCK algorithm, which is widely used in virtual memory systems, is adopted as a baseline page replacement algorithm in our experiments [7].

The access time of a PCM device is composed of T_{ACCESS} and $T_{TRANSFER}$, where T_{ACCESS} is a static time component needed for each independent access and $T_{TRANSFER}$ is a time component proportional to the request size. In our experiments, we set the values modeled in the previous study [24]. Suppose that the proportional time component of a write operation is $T_{TRANSFER\_write}$, that of a read operation is $T_{TRANSFER\_read}$, and the size independent time component of read and write operations is T_{ACCESS_read} and T_{ACCESS_write}, respectively. When the total number of read operations is N_{read}, the total number of write operations is N_{write}, and the page size is $SIZE_p$, then the PCM access time by each request T_{EACH} and the total elapsed time T_{TOTAL} can be calculated as follows.

$$T_{EACH} = T_{ACCESS} + T_{TRANSFER}$$
$$T_{TOTAL} = N_{read}(T_{ACCESS\_read} + T_{TRANSFER\_read} * SIZE_p) + N_{write}(T_{ACCESS\_write} + T_{TRANSFER\_write} * SIZE_p)$$

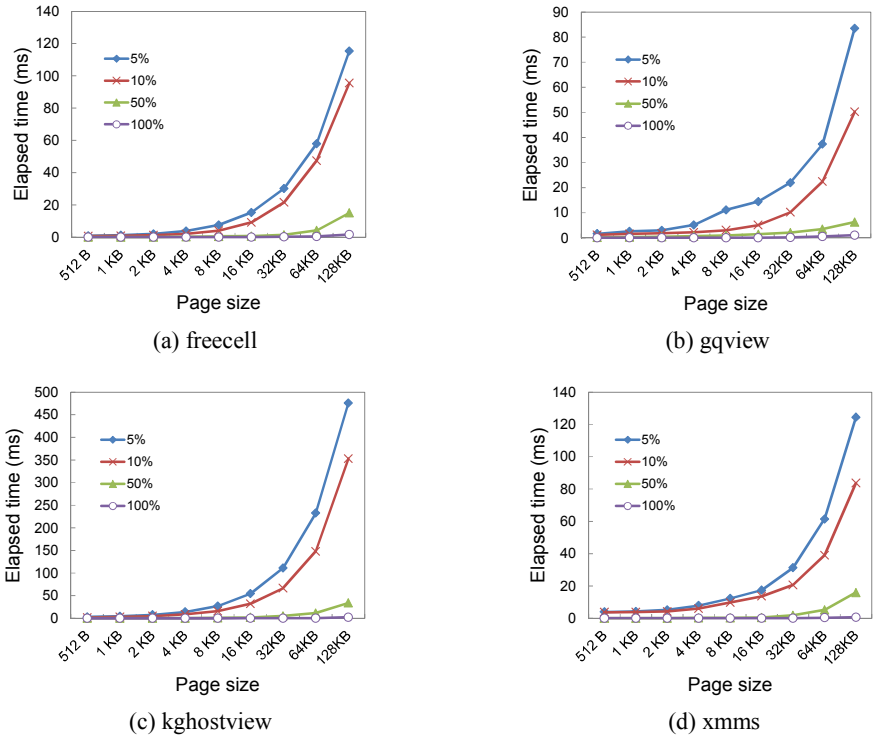
2. Effects of the Page Size

Fig. 3 shows the total elapsed time of each workload as a function of the page size. For each configuration, we also vary the DRAM memory size from 5% to 100% of the footprint of each workload. As shown in the figure, the total elapsed time is improved as the page size becomes smaller in all cases. This is because only the necessary part to execute the program is loaded into memory. Note that a hard disk drive has a significant portion of size-independent time component to access data, whereas PCM does not do so, and thus the transfer size is critical to PCM performances. As we eliminate the software I/O stack overhead and the context switch overhead from the page fault handling process, reducing the transfer size is important in minimizing the total cost of a page fault. When the page size is reduced from 4KB to 512 bytes, the performance is improved by 58.9% on average.

3. Effects of Read-ahead

Modern operating systems such as Linux use a read-ahead technique that loads several adjacent pages as well as the requested page itself when a page fault occurs. This is effective in disk storage systems to minimize the head movement as storage accesses frequently exhibit sequential behavior. In this section, we analyze the effectiveness of read-ahead on the performance of PCM-based swap systems.

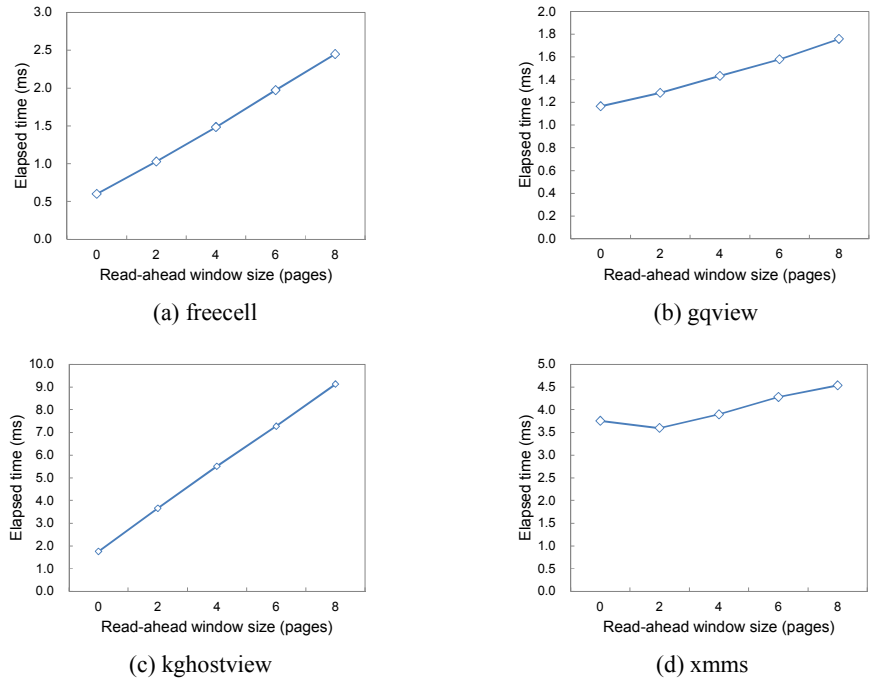
Fig. 4 shows the total elapsed time of each workload as the read-ahead window size is varied. In this experiment, the page size is set to 512 bytes based on the result of Section III-2. The total memory size is set to 10% of the memory footprint for each workload. As shown in Fig. 4, the performance is improved significantly when the read-ahead option is turned off. Specifically, the total elapsed time is improved by 51.6% on average and up to 80.7% when the read-ahead option is turned off, in comparison with the read-ahead window size of 8. This is because the cost of loading unnecessary pages through read-ahead degrades the performance of PCM-based swap systems significantly.



(a) freecell (b) gqview

(c) kghostview (d) xmms

Fig. 3. Total elapsed time of each workload as the page size of PCM swap systems is varied



(a) freecell (b) gqview

(c) kghostview (d) xmms

Fig. 4. Total elapsed time of each workload as a function of the read-ahead window size

4. Effects of Page Replacement Algorithms

In this section, we assess the effects of page replacement algorithms in PCM-based swap systems. CLOCK is a representative replacement algorithm for virtual memory systems [7]. To support the replacement algorithm in modern computer systems, a reference bit and a dirty bit are provided for each memory page that

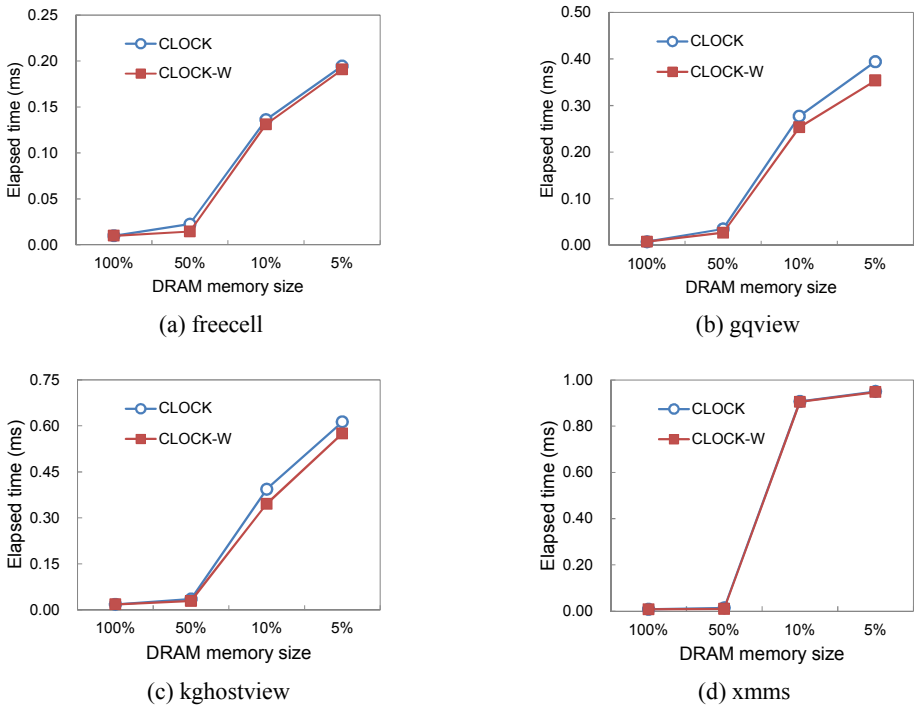


Fig. 5. Total elapsed time of each workload as a function of the DRAM memory size

are set by the paging unit hardware. Specifically, on a hit to a page, the paging unit hardware sets the reference bit of the page to 1 when a read or a write reference for that page occurs, and sets the dirty bit to 1 when a write reference occurs. Then, pages are maintained 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, the position next to the last evicted page. This scan continues until a page with a reference bit of zero is found, and that page is then replaced. If the dirty bit of the victim page is set to 1, the page is written back to swap storage since it has been modified while resident in memory. In the course of the scan, for every page with the reference bit of 1, CLOCK clears it to 0, without removing the page from memory. The reference bit of each page is an indication of whether that page has recently been accessed or not; and pages not referenced upon the return of the clock-hand to that page will be replaced.

Though CLOCK estimates future page accesses by considering the recency of references, it does not have the ability to predict future write accesses. Though CLOCK uses the dirty bit, this bit simply indicates that the page was written at least once while resident in memory. There is no way to convey the information that the page has been recently written or not. Since a write operation of PCM is significantly slower than a read operation, it is necessary to predict future write accesses and maintain pages likely to be written again in memory.

To this end, we devise a new algorithm CLOCK-W, by simply changing the original CLOCK algorithm such that the *dirty bit* is used to capture the recency of write references for page replacement instead of the *reference bit*. Specifically, CLOCK-W checks the dirty bit of the page to which the clock-hand points. If the dirty bit is 1, CLOCK-W clears the dirty bit and backs it up to a kernel variable in the page structure. By so doing, OS still knows that it is a dirty page that should be written back to storage when selected as an eviction victim. If a page with the dirty bit of zero is found, that page is replaced since it is not a recently written page.

Fig. 5 shows the total elapsed time of CLOCK and CLOCK-W as a function of the DRAM memory size. In the figure, 100% memory size is an unrealistic condition where the complete memory footprint can be loaded at the same time, not incurring any page replacement. In this case, the two algorithms perform the same. As shown in the figure, CLOCK-W performs slightly better than original CLOCK for call cases, even though it does not consider the recency of read references. This is

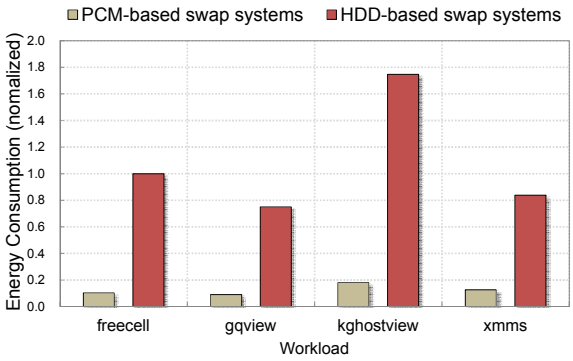


Fig. 6. Comparison of total energy consumption

because the cost of a write operation is even larger than that of a read operation in PCM.

Though the performance of CLOCK-W is not much better than CLOCK in our simple experiments, we can state that the design of an efficient replacement algorithm for the PCM-based swap system is needed to consider the re-reference likelihood of write operations.

5. Effects of Reducing DRAM Memory Size

The energy consumption of DRAM memory is becoming a dominant portion of the total system energy due to the increasing memory capacity driven by memory-intensive applications and multi-core processors. In this section, we investigate the effects of reducing the DRAM memory size when PCM-based swap storage is adopted. Regarding the performance aspect, we have already seen in Fig. 3 that a reasonably good performance can be obtained even with the 10% DRAM size of the total footprint when we use the page size of 512 bytes.

Fig. 6 shows the total energy consumption of PCM-based swap systems with the 10% DRAM size in comparison with the disk-based conventional swap system with the 100% DRAM size. As shown in the figure, the PCM-based swap system consumes 88.1% less energy than the HDD-based swap system. This result indicates that the new swap architecture of "small memory – fast swap" will be effective in reducing the energy consumption of future computer systems.

IV. Related Work

Studies on adopting PCM in the memory hierarchy of computer systems have focused on enhancing the write performance and endurance of PCM. There are three categories of research that aims at achieving these goals.

The first category uses a certain amount of DRAM to reduce the number of writes that occurs on PCM [1, 5]. Dhiman et al. present a hybrid PCM and DRAM memory architecture called PDRAM [5]. 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 [15]. 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 [16]. For prediction of read and write access patterns, they calculate the weighting value using the ratio of writes to total references. One problem with this scheme is that the algorithm requires exact time information for every memory reference, which is difficult to be implemented 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]. A similar technique is also presented by Lee et al. [11]. These two techniques 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 [12]. Similar work is also performed by Zhou et al. [9]. Cho and Lee present the flip-n-write technique, which flips all bits in a page if it incurs less number of bit writes [13]. Wongchaowart et al. present a content-aware block placement algorithm that selects a free block with similar contents among those in the free block lists [14].

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 [9]. Seong et al. present the security refresh technique that prevents wear-out from malicious attackers by constantly migrating physical locations inside the PCM, obfuscating the actual data placement from users and system software [10].

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 [9]. 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].

Wear-leveling techniques given above are table-based translation schemes that require tables to track write counts and to perform logical to physical address mappings. To obviate overhead that comes from table-based translation, Qureshi et al. propose the Start-Gap wear-leveling method by using an algebraic mapping between physical and logical addresses [17]. Without table structures, Start-Gap uses just two registers, named Start and Gap, to perform wear-leveling and still achieves good PCM lifetime.

V. CONCLUSION

In this paper, we explored the empirical characteristics of PCM-based swap systems and showed how this system can be managed efficiently. Specifically, we demonstrated that reducing the page size and turning off the read-ahead option are effective in the PCM-based swap system as the page fault handling time is significantly small. We also showed that the performance is not degraded but the DRAM's energy consumption can be reduced significantly when we use only a small amount of DRAM memory. We expect that the result of this paper will lead to the transition of the legacy swap system structure of "large memory – slow swap" to a new paradigm of "small memory – fast swap" in the future.

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] J. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," *Proc. USENIX HotOS Workshop*, 2009.

[4] 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.

[5] 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.

[6] Phase Change Memory Product, http://www.micron.com/products/phase-change-memory, Micron, 2013.

[7] F.J. Corbato, "A paging experiment with the multics system," *In Honor of P.M. Morse*, MIT Press, 1969.

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

[9] 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.

[10] N. Seong, D. Woo, and H. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically

randomized address mapping," *Proc. IEEE ISCA Conf.*, pp. 383-394, 2010.

[11] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," *Proc. IEEE ISCA Conf.*, pp. 2-13, 2009.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," *Proc. IEEE Symp. Microarchit.*, pp. 14-23, 2009.

[18] 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.

[19] E. Lee, S. Yoo, and H. Bahn, "Design and implementation of a journaling file system for phase-change memory," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1349-1360, 2015.

[20] 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.

[21] J. Condit, E. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," *Proc. ACM SOSP Conf.*, 2009.

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

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

[24] H. Yoon et al., "Techniques for data mapping and buffering to exploit asymmetry in MLC PCM," *SAFARI Technical Report No. 2013-002*, 2013.

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.

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.