Efficient Page Caching Algorithm with Prediction and Migration for a Hybrid Main Memory

Hyunchul Seok, Youngwoo Park, Ki-Woong Park, and Kyu Ho Park
KAIST
Daejeon, Korea
{hcseok, ywpark, woongbak}@core.kaist.ac.kr and kpark@ee.kaist.ac.kr

ABSTRACT
Emerging next generation memories, NVRAMs, such as Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM) are rapidly becoming promising candidates for large scale main memory because of their high density and low power consumption. Many researchers have attempted to construct a main memory with NVRAMs, in order to make up for the limits of NVRAMs. However, we find that the preexisting page caching algorithms, such as LRU, LIRS, and CLOCK-Pro, are often sub-optimal for NVRAMs due to its DRAM-oriented design including uniform access latency and unlimited endurance. Consequently, the algorithms cannot be directly adapted to the hybrid main memory architecture with PRAM.

To mitigate this design limitation, we propose a new page caching algorithm for the hybrid main memory. It is designed to overcome the long latency and low endurance of PRAM. On the basis of the LRU replacement algorithm, we propose a prediction of page access pattern and migration schemes to maintain write-bound access pages to DRAM. The experiment results have convinced us that our page caching algorithm minimizes the number of the write access of PRAM while maintaining the cache hit ratio. The results show that we can reduce the total write access count by a maximum of 52.9% and the consumed energy by 19.9%. Therefore, we can enhance the average page cache performance and reduce the endurance problem in the hybrid main memory.[4]

Categories and Subject Descriptors
B.3.2 [Memory Structures]: Design Styles Cache memories; D.4.2 [Operating Systems]: Storage Management - *Allocation /deallocation strategies, Main memory*

General Terms
Page cache algorithm

Keywords
Page Cache, Hybrid Main Memory, PRAM, Migration

1. INTRODUCTION
The main memory of today's computer systems is showing significant change with the emergence of next generation types of memory, NVRAMs, such as Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM) [20][24][14]. Several studies have observed that DRAM based main memory, while it significantly increases the computing power, also greatly increases the cost of a computer system [4]. Therefore, many trials have been made to replace DRAM. One such trial attempted to use hybrid memory by including NVRAMs.

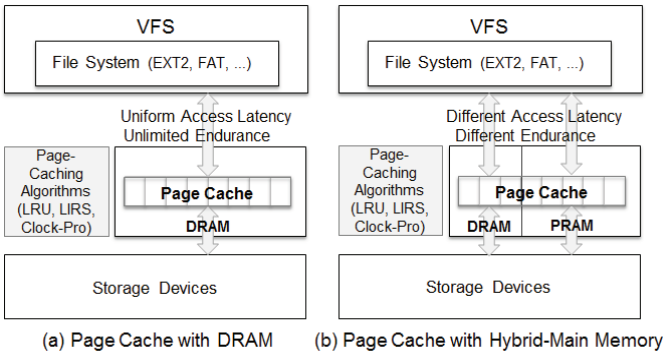


Figure 1. Page caching system architecture

Among these memories, PRAM is the most promising candidate for uses in large scale main memory because of its high density and low power consumption [20]. While DRAM stores each bit of data in a separate capacitor, PRAM uses the phase of the material to represent each bit. PRAM density is expected to be much greater than that of DRAM (about four times). Also, PRAM is a non-volatile memory because the phase of the material does not change after power-off. It has negligible leakage energy regardless of the size of the memory. Table 1 shows the properties of PRAM compared to those of DRAM [17]. Though PRAM has attractive features such as high density, non-volatility, low idle power, and good scalability, the access latency of PRAM is still not comparable to DRAM latency. PRAM also has a worn-out problem caused by limited write endurance. In previous research, the hybrid main memory approaches of DRAM and PRAM have been adopted to make up for the latency and endurance limits of PRAM [10][20][18]. Those researchers proposed several hardware and software schemes to manage the hybrid main memory.

On the other hand, in modern computer systems, a large part of main memory is used as a page cache to hide disk access latency, as can be seen in Figure 1(a). Many page caching algorithms such as LRU, LIRS [12], and CLOCK-Pro [19] have been developed and show good performance for current DRAM based main memory. However, such previous page caching algorithms only considered the main memory with uniform access latency and

[4] This work is based on an earlier work: SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computing, Copyright 2011 ACM 978-1-4503-0113-8/11/03.

http://doi.acm.org/10.1145/1982185.1982312

unlimited endurance. They cannot be directly adapted to the hybrid main memory architecture with DRAM and PRAM, as shown in Figure 1(b).

This paper is an extended work related to our previous work in [22]. The previous work mainly focused on a page caching algorithm design to enhance the average page cache performance and to reduce the endurance problem. In this paper, attempts are made to integrate a prediction and page migration mechanisms as the key primitive for additional performance enhancement.

The main objective of this work is to propose a new page caching algorithm for the hybrid main memory. The algorithm is designed to overcome the long latency and endurance problem of PRAM. On the basis of conventional cache replacement algorithms, we propose a prediction of the page access pattern by page monitoring and migration schemes to move write-bound access pages to DRAM. In particular, we present analytic metrics for PRAM endurance, energy, and latency, and illustrate that existing page caching algorithms such as LRU, LIRS, and CLOCK-Pro are suboptimal in the hybrid main memory. We present improved algorithms for enhancing the page cache performance on the hybrid main memory. It minimizes the write access of PRAM while maintaining the cache hit ratio. Therefore, we can enhance the average page cache performance and reduce the endurance problem in the hybrid main memory. The remainder of the paper is organized as follows. In Section 2, we briefly summarize the conventional page caching algorithms and their limits when used in hybrid memory. In Section 3 we present the design of our algorithm and describe the prediction algorithm and migration strategy. The performance evaluation results are given in Section 4, and we briefly discuss future work in Section 5. Section 6 concludes this paper.

2. BACKGROUND

A proper page caching algorithm can have a significant effect on improving performance of I/O by hiding the long latency of disks. Many studies have been undertaken to make efficient page cache algorithms. However, most page cache replacement algorithms have been designed for memory with uniform access latency and unlimited endurance. In this paper, we evaluate the page caching algorithms for hybrid memory, which has different read and write latencies and different endurance. We propose a new page caching algorithm that includes prediction and migration schemes for efficiently hiding the bad effects derived from the different properties of hybrid memory and compare the results. Although we implement our algorithm based on the LRU replacement algorithm, our technique can be utilized with the conventional page caching algorithm and can improve the performance of page caching in hybrid memory.

LRU (Least Recently Used) has been widely used as a cache management and page cache replacement algorithm [6][2][8][9] [11]. When the cache is full and a miss occurs, this algorithm selects the page where it is in the LRU position as a victim. One of the advantages is that it is very simple to implement this algorithm. The algorithm also has constant time and space overhead. But it has some disadvantages. It is known that LRU shows the best performance on the workloads of Stack Depth Distribution (SDD) [6]. However, LRU cannot operate well with an access pattern with weak locality, such as sequential scans, a cyclic pattern with slightly larger than cache size.

LIRS (Low Inter-Reference Recency Set) was proposed by Jiang and Zhang in 2002 in order to solve the problems of LRU [12]. This algorithm uses Inter-Reference Recency (IRR) to determine which page should be replaced. The objectives of LIRS are both to effectively address the limits of LRU and to retain the low overhead of LRU. As the history information of each page, IRR is defined as the number of other pages accessed between two consecutive references to the page. The algorithm assumes that if the IRR of a page is large, the next IRR of the page is likely to be large. Therefore, LIRS selects pages with large IRRs for replacement.

Table 1. Properties of PRAM

Attributes	DRAM	PRAM
Non-volatility	No	Yes
Cost/TB	Highest (~4x PRAM)	Low
Read Latency	50ns	50-100ns
Write Latency	20-50ns	~ 1us
Read Energy	~ 0.1nJ/b	~ 0.1nJ/b
Write Energy	~ 0.1nJ/b	~ 0.5nJ/b
Idle Power	~ 1.3W/GB	~ 0.05W
Endurance	∞	10^8 for write

CLOCK-Pro was proposed by Jiang, Chen, and Zhang in 2005 [19]. It is based on CLOCK which is a simple approximation of the LRU replacement algorithm [7][5]. CLOCK can reduce the overheads of the LRU algorithm, which are related to the overhead of moving a page to the MRU position on every page hit [3]. The objectives of CLOCK-Pro are to obtain a low computational requirement and to remove the disadvantages, which are the same disadvantages as those of LRU with the workload of weak locality. Therefore, CLOCK-Pro was designed to include the way in which LIRS and CLOCK work.

In addition to the replacement algorithms mentioned above, there are many caching algorithms, most of which have the goal of improving the performance problems of LRU. In studies of such methods, FBR [21], LRU-2 [16], 2Q [13], LRFU [15], and MQ [25] were proposed and researchers tried to combine "recency" (LRU) and "frequency" (LFU) in order to compensate for the disadvantages of LRU. These disadvantages are derived from the access patterns such as the sequential stream and a cyclic pattern with larger than cache size. However, these methods cannot consider the physically different properties of hybrid memory. For example, PRAM has about ten times larger write latency than DRAM does, as can be seen in Table 1. Therefore, if some pages are frequently accessed by write accesses and these pages are in PRAM memory, the overall performance will degrade. In such a case, we can improve the performance if the page cache algorithm can make a decision to put the write-bound pages into DRAM rather than into PRAM. In this case, the performance is affected by the position of the pages.

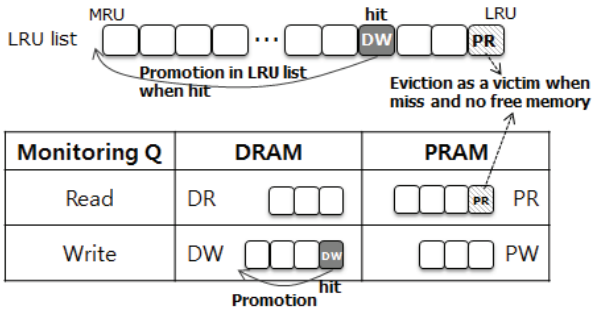


Figure 2. Overview of page caching algorithm

3. PAGE CACHING ALGORITHM FOR HYBRID MAIN MEMORY

In this section, we describe the design of a new page caching algorithm for the hybrid main memory; this design consists of DRAM and PRAM. Although the conventional cache algorithms show good performance, they cannot be directly adapted to the hybrid main memory. Because the hybrid main memory uses two different types of memories, it is important to consider their properties in order to maximize the performance and efficiency. As can be seen in Table 1, PRAM has a very long latency compared to DRAM when writing a page. If the page cache algorithm causes a lot of writes to PRAM, the average page cache performance gets worse. In addition, PRAM has low endurance compared to DRAM. If there are many write accesses, PRAM will be worn-out quickly. Consequently, the page cache algorithm on the hybrid main memory should be designed to overcome the long latency and endurance problem of PRAM. To solve these problems, we propose a new page caching algorithm with prediction of page access pattern and migration schemes.

3.1 Basic Operation

To satisfy the requirements, we designed the new page caching algorithm according to the conventional cache algorithm. We add a prediction of page access pattern and migration schemes. Although any conventional cache replacement algorithm can be adapted, we chose the LRU replacement algorithm, which is very simple but works well. In order to monitor the pages and to adapt the migration scheme, we use four monitoring queues, which consist of a DRAM read queue, a DRAM write queue, a PRAM read queue, and a PRAM write queue, as shown in Figure 2. When one page block is accessed, it is retained into both the LRU list and one of the four queues by its access pattern and the memory type where it is located.

Figure 2 shows the basic operation of our algorithm. When a page fault occurs, we put the page into the MRU position in the LRU list with the LRU replacement algorithm. In addition, we put the page into one of the monitoring queues according to the page access type. For example, if the page's access request is read and a selected memory page is on DRAM, we put the page into the DRAM read queue. If there is no free memory when a miss occurs, the least recently used page block is selected as a victim page, which also follows the LRU replacement algorithm. At the same time, we evict the page related to the victim page. For example, if a type of the victim page is read cache and resides in PRAM, we eliminate the page from the PRAM read queue by evicting the page in LRU list. When the page hits, the page in both the LRU list and the monitoring queue is promoted, as can be seen in Figure 2.

Algorithm 1 : Page Migration Function

input: page address and request type
W_{cur}: Current weight value
Tr_{mig}: Threshold value for determining migration
Tr_q: Threshold value for determining movement between read and write queues

```
Calculate W_cur
if page in PRAM then
    if W_cur ≥ Tr_mig then
        migrate the page to DRAM
    else if W_cur ≥ Tr_q and page in read queue then
        the page moves to write queue
    else if W_cur ≤ −Tr_q and page in write queue then
        the page moves to read queue
    end if
else if page in DRAM then
    if W_cur ≤ −Tr_mig then
        migrate the page to PRAM
    else if W_cur ≤ −Tr_q and page in write queue then
        the page moves to read queue
    else if W_cur ≥ Tr_q and page in read queue then
        the page moves to write queue
    end if
end if
```

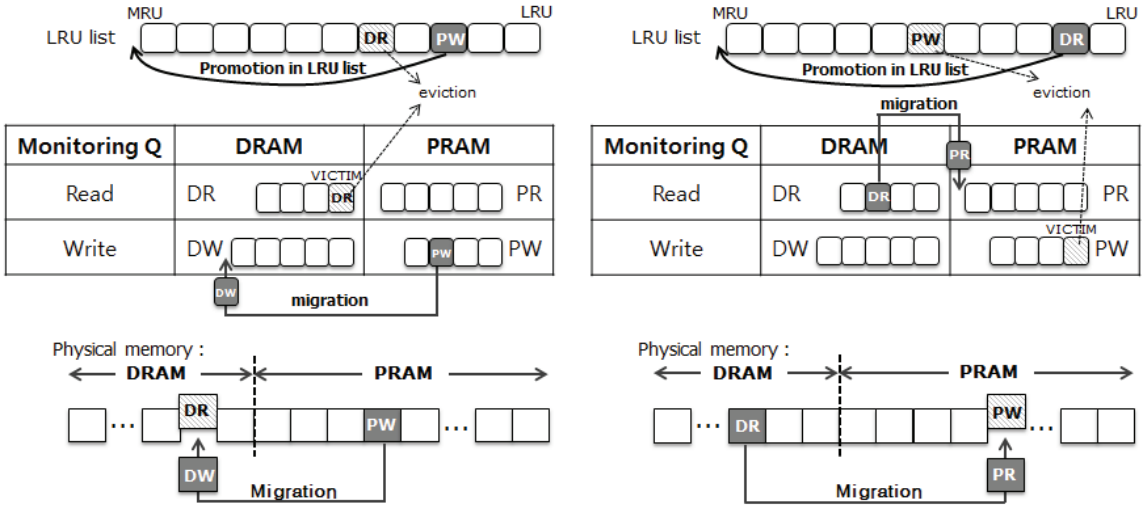
3.2 Prediction and Page Migration

The write-bound pages on PRAM cause performance degradation and worn-out problem because of PRAM's long latency and low endurance. To solve these problems, we need to move the write-bound pages from PRAM to DRAM. Additionally, we must move the read-bound pages from DRAM to PRAM because we have to gather the read-bound pages to PRAM. In order to effect the efficient migration of pages, we have to know which pages are write-bound and which pages are read-bound. For prediction of the access pattern, we calculate the weighting values, which indicate how close the values are to write-bound or read-bound. By monitoring the request type of the page requests, the weighting value can be calculated by using a moving average with weight $\alpha \in [0,1]$ as follows:

$$W_{cur} = \alpha W_{prev} + (1-\alpha)RT \quad (1)$$

, where RT means the requested type of the page; its value is 1 if the page request is write and -1 if the page request is read. When a page fault occurs, the page is inserted into both the LRU list and one of the monitoring queues. We selected the value of α as 0.5. We will explain in detail in section 4.2.1.

There are two migration cases, as shown in Figure 3: one is the migration of write-bound pages from PRAM to DRAM; the other is the migration of read-bound pages from DRAM to PRAM. To determine when migration occurs, we calculate the W_{cur} value at every request. According to equation 1, this value is increased



(a) Migration of the write-bound page

(b) Migration of the read-bound page

Figure 3. Migration of cached pages between DRAM and PRAM

when write requests occur and is decreased at every read request. Algorithm 1 shows how the cached pages are migrated. When a page request hits, W_{cur} of the hit page is calculated by equation 1 and we determine whether migration occurs. For example, if the write access is hit on a page in the PRAM write queue, PW, as shown in Figure 3(a), and its weighting value is over Tr_{mig}, this page will migrate to the DRAM write queue. Similarly, as can be seen in 3(b), if a page in the DRAM read queue, DR, is hit by a read request and its W_{cur} value is under $-Tr_{mig}$, this page will migrate to the PRAM read queue. We use two threshold values for determining the migration and the movement between read and write queues in the same memory. Tr_{mig} is the threshold value for determining whether a page is migrated and Tr_q is the threshold value for determining movement between read and write queues.

Figure 3(a) shows an example of migration from PRAM to DRAM. If there is no free space in DRAM, we have to select a victim page on DRAM. In this case, we select the victim page from the bottom of the DRAM read queue and remove it from DRAM. The write-bound page in PRAM is moved to the DRAM where the victim page was located. In the DRAM write queue, this page is put into the top of the queue. If there is no element in the DRAM read queue when we find a victim page for migration, we choose a victim page from the bottom of the DRAM write queue, which means that the victim page is the least recently used. The migration of read-bound pages is similar to the migration of write-bound pages, as shown in Figure 3(b). To select a victim page, we select the bottom page of the PRAM write queue first, and if there are no pages in the PRAM write queue, we choose the bottom page of the PRAM read queue. When we remove a victim page for migration, we just remove it from the memory, the LRU list, and the monitoring queue. The reason why we do not change the pages between the victim page for migration and the page that will migrate is that such an action would cause an additional write on PRAM.

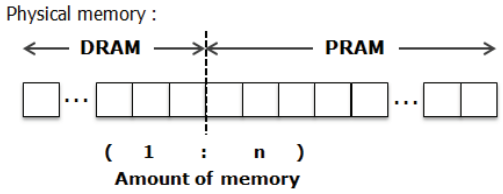


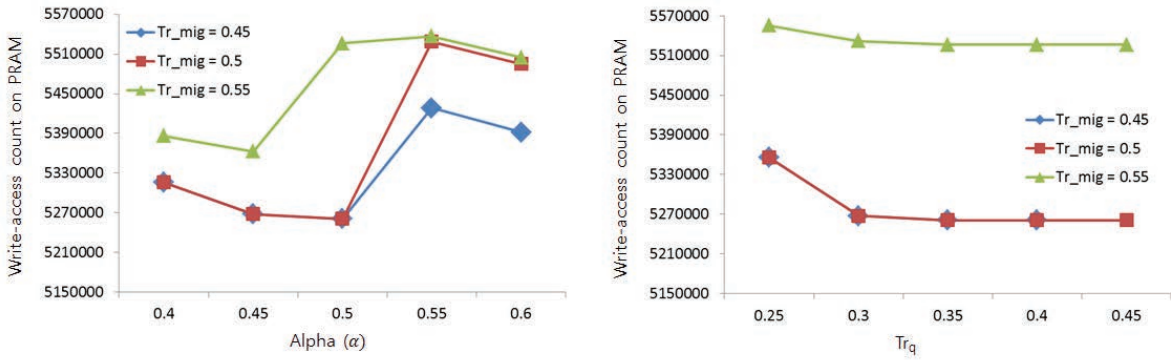
Figure 4. Composition of the hybrid main memory

4. EXPERIMENT

In order to evaluate the proposed page cache algorithm for the hybrid main memory, we used a trace-driven simulation. We implemented our algorithm and designed the hybrid main memory architecture. In this section, we present the performance results of our hybrid cache algorithm. First, we demonstrate the overall experimental environment. We then describe the evaluation results in terms of hit ratio, write access count on PRAM, and energy consumption.

4.1 Experiment Setup

In order to evaluate the performance of our page cache scheme, we have to define the hybrid main memory. We assume that the hybrid main memory consists of DRAM and PRAM, which are divided by a memory address. The memory which has the low memory address is DRAM and the high section is allocated to PRAM, as shown in Figure 4. PRAM density is generally expected to be higher than that of DRAM [17][23][20], so that we allocate larger amount of memory to PRAM. Because PRAM density is expected to be four times higher than that of DRAM [17][23], we mainly select that the PRAM-to-DRAM ratio is four. However, this density value can be varied in several researches [20] so that we additionally evaluated out page caching algorithm with various PRAM-to-DRAM ratios.



(a) Results when Tr_q is 0.35

(b) Results when Alpha (α) is 0.5

Figure 6. The experiment results with various parameters when the memory size is 2000 on Financial1 workload

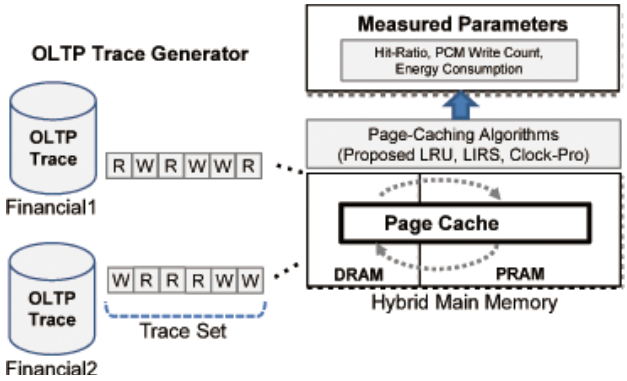


Figure 5. Trace-driven simulation for evaluation of page-caching algorithms

To evaluate the performance characteristics of the proposed page caching algorithm, we constructed a trace-driven simulation environment and coupled it to the OLTP traces. The objective of the simulator is to evaluate the performance of the page caching algorithms with regard to a real hybrid main memory and practical workload. Figure 5 shows the overall simulation environment.

As the workloads for the performance evaluation, we exploited an Online Transaction Processing (OLTP) application I/O, which has high throughput and is insert/update-intensive. We use two workload traces from the OLTP of two large financial institutions. These traces were made available courtesy of Ken Bates from HP, Bruce McNutt from IBM, and the Storage Performance Council [1]. These two traces are related to requests to a storage and suitable to test the page caching algorithm. The first financial workload has 9 million traces, and among them, about 80% of traces are write accesses. The second financial workload has 5 million traces and about 19% of traces are write accesses, as shown in Table 2. By using two financial workloads, we can test the performance with the cases of write-intensive and read-intensive workloads.

4.2 Evaluation Results

Table 2. A summary of the workloads used in this paper

Trace Name	Number of Requests	Ratio of Write-access pages
Financial1	9156833	80.59%
Financial2	5436256	18.95%

In this section, we show the trace-based simulation results in terms of hit ratio, write access count on PRAM and consumed energy in order to show the performance of our page caching algorithm. We compared the experimental results for our algorithm with those of the conventional page caching algorithms such as LRU, LIRS, and CLOCK-Pro.

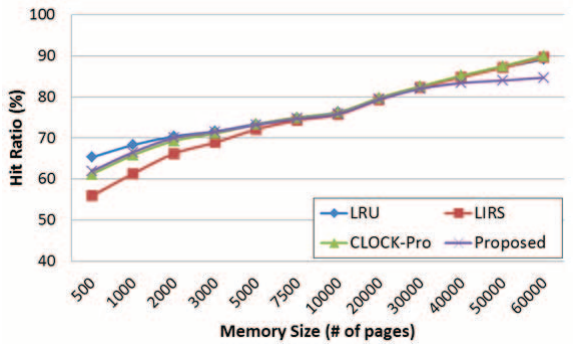
4.2.1 Parameters used in evaluation

In order to predict a page's access pattern, we use equation 1, which includes one parameter, α. In addition, we use two threshold values to determine the time when migration occurs and when a page cache changes its status between read-bound and write-bound pages. The two values are Tr_{mig} and Tr_q, explained in section 3.2. Before we test our page cache algorithm, we have to determine the values of these parameters in order to minimize the total access counts on PRAM.

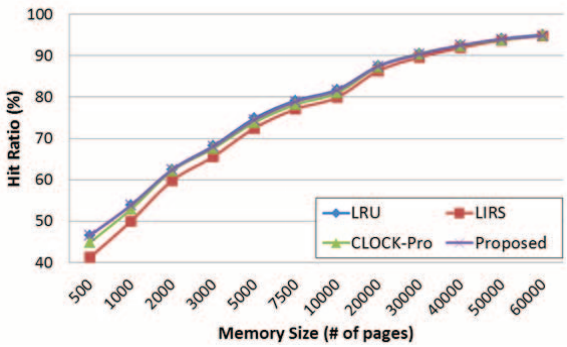
We chose the parameter values through experiments. Figure 6 shows the total number of write accesses on PRAM with the *Financial1* workload when we change α, Tr_{mig}, and Tr_q values. Figure 6(a) shows the total access count on PRAM with various α and Tr_{mig} when Tr_q is fixed at 0.35. From Figure 6(a), the evaluation results convince us that the write-access count is minimized when α is 0.5 and when Tr_{mig} is 0.45 or 0.5. Therefore, we select α as 0.5. Figure 6(b) shows the results when α is 0.5. From this graph, we can know that the results are the smallest when Tr_q is larger than 0.3. Finally, we select the values of Tr_{mig} and Tr_q as 0.5 and 0.35, respectively.

4.2.2 Hit ratio

The first experiment measures the hit ratio, which is important in determining the performance of the page caching algorithm. We compared the hit ratio of our page caching algorithm to the hit ratios of the conventional page caching algorithms. We tested it with many sizes of main memory. The results are shown in Figure 7.

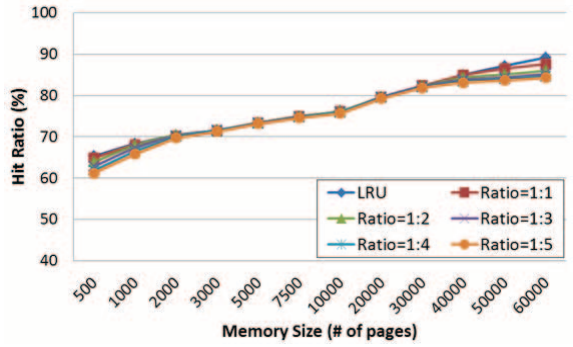


(a) *Financial1*

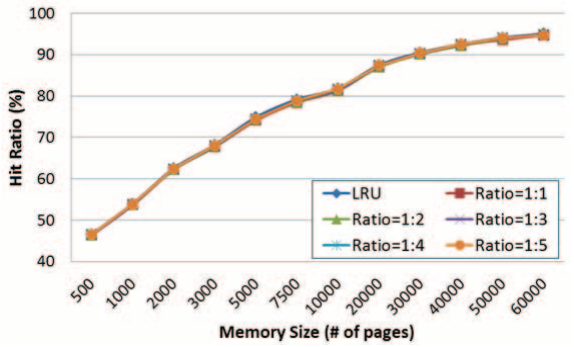


(b) *Financial2*

Figure 7. Hit ratio of proposed algorithm and conventional algorithms on financial workloads



(a) *Financial1*



(b) *Financial2*

Figure 8. Hit ratio of proposed algorithm with different PRAM-to-DRAM ratio on financial workloads

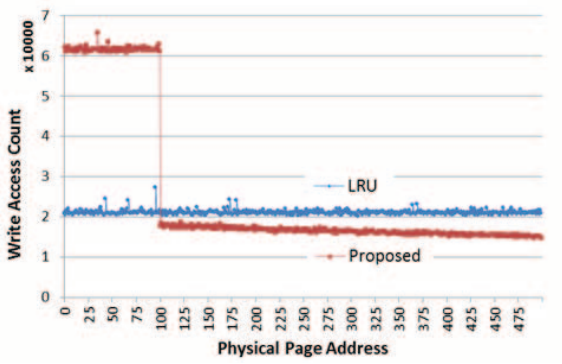
From the results, it can be seen that the LRU replacement algorithm shows the highest hit ratio through the whole range of memory sizes. While the hit ratio of our algorithm is lower than that of LRU, the results show that the hit ratio is similar to those of the LRU, LIRS, and CLOCK-Pro algorithms. Actually, we designed our algorithm on the basis of the LRU replacement algorithm and added the prediction and migration schemes. We expected the hit ratio of our algorithm to be similar to that of the LRU replacement algorithm. Because we designed the selected victim page for migration to simply be eliminated, as explained in section 3.2, it is possible that the page faults will occur more. Consequently, the hit ratio is lower than that of the LRU. Although the migration scheme causes a degradation of the hit ratio, the hit ratio is still larger than that of LIRS and CLOCK-Pro for small sizes of the main memory.

The next experiment measures the hit ratio with varying the ratio of the size of PRAM to the size of DRAM, as shown in Figure 8. We compare the hit ratio of the proposed scheme with the hit ratio of LRU replacement algorithm. In overall region, the hit ratio of the proposed scheme shows a similar hit ratio in comparison with LRU. In case of the results of *Financial1*, when the size of memory is set to 500 pages, the hit ratio of proposed scheme is decreased from 64.9% to 61.2% as the size of PRAM increases. This can be seen in Figure 8(a). It is due to the size reduction of DRAM, which leads to hit ratio degradation by the victim process when many migrations of write-bound pages occur. When the size of memory is set to 60,000 pages, the experiment result also shows a gap between hit ratio of proposed scheme and the hit ratio of LRU. The main reason is that even infrequently accessed pages can be remained to LRU list as the size of DRAM increases. On the other hand, the above pages in the proposed scheme can be frequently selected as victim so that the pages can be evicted from LRU list, which leads to more cache miss. In case of the results of *Financial2*, the hit ratio at 500-page size of a memory is barely changed. With read-intensive workload like *Financial2*, hit ratio degradation by the victim process of a read-bound page does not affect largely because the size of PRAM is larger than DRAM.

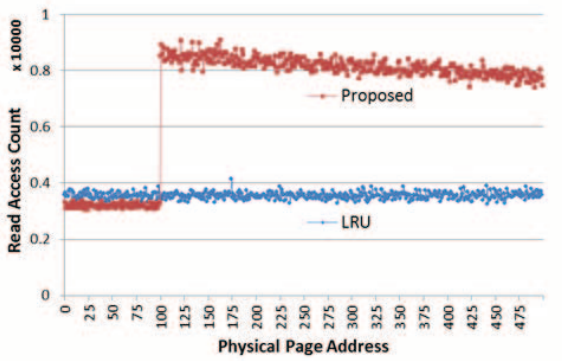
4.2.3 Write access count on PRAM

The write access count on PRAM is important because it is related to the total latency of the page cache and the lifetime of PRAM. PRAM has a very long latency compared to DRAM, so the performance of page caches degrades if many write pages hit on PRAM. In addition, PRAM wears out more quickly because of its

low endurance. Therefore, in order to use a page cache on the hybrid main memory of DRAM and PRAM, a page caching algorithm must be able to reduce the write access count on PRAM.



(a) Write access distribution



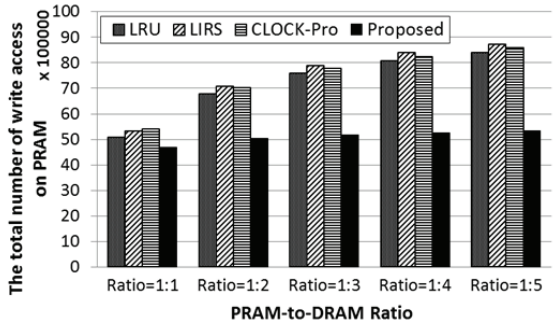
(b) Read access distribution

Figure 9. Write and read access distribution in the hybrid main memory

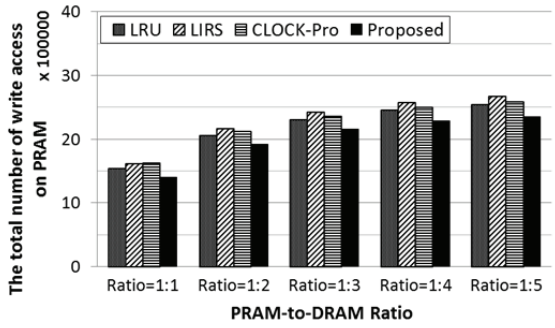
To show that the proposed scheme reduces the write access count on PRAM, and conversely increases the read access count on PRAM, an experiment was designed by profiling the value of the read/write access distribution when the start address was varied from 0 to 500. In this experiment, it was assumed that the first 100 blocks would be mapped to DRAM, and the other would be mapped to PRAM. As can be seen in Figure 9(a), the minimized write access distribution is suitable for PRAM characteristics. Conversely, as shown in Figure 9(b) the maximized read access distribution implies that the read-bound data is migrated to the PRAM region, and so, in comparison with the LRU scheme, the proposed page-caching algorithm is more suitable for a hybrid main memory system.

In this experiment, we evaluated the write access count on PRAM and compared it with the results of the conventional algorithms. During the simulation, we counted read and write accesses of DRAM and PRAM. Figure 10 shows the total number of write accesses on PRAM with workloads, denoted as *financial1* and *financial2*. We measured the count number with several memory sizes. The memory size is the total size of DRAM and PRAM. In Figure 10, the number of accesses for the four algorithms decreases as the memory size grows. Because the number of faults is decreased with the increase of the size of memory, the total number of write accesses on PRAM decreases when the memory size increases.

When using our algorithm, we can know that the total number of



(a) *Financial1*



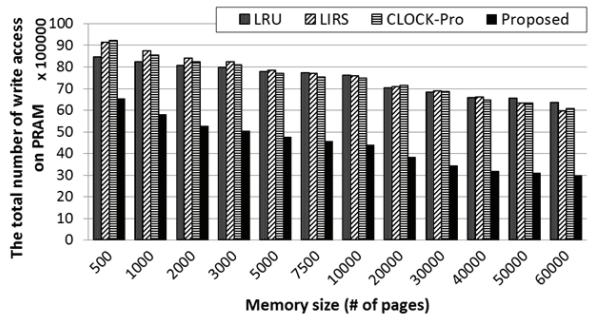
(b) *Financial2*

Figure 11. The total write access count with different PRAM-to-DRAM ratio when the memory size is 2000 on financial workloads

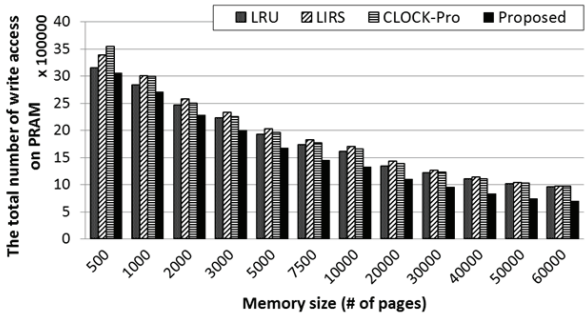
write accesses is reduced compared to that for the conventional page caching algorithm. We can reduce the total write access count on PRAM by 34.8% for *financial1* and 6.97% for *financial2* when we use the hybrid main memory with 2000-page sizes. We can reduce the total write access count by a maximum of 52.9% for *financial1* and 27.8% for *financial2*. The reason why the gain of *financial2* is smaller than that of *financial1* is that the workload of *financial2* has a small number of write accesses and we can obtain performance improvement by reducing the write access request on PRAM by using migration.

The next experiment shows the total write access count on PRAM when the ratio of the size of PRAM to the size of DRAM varies. Figure 11 shows the results when the total size of hybrid memory is 2000 pages. Because the size of PRAM is increased with the increasing the ratio of PRAM to DRAM, total write counts are increased with both two workloads. However, in case of *financial1*, the proposed algorithm can maintain the total number of write access in all cases while the results of other algorithms are increased, shown in Figure 11(a). Therefore, the proposed algorithm can efficiently decrease the total write access number with write-intensive workload like *financial1*. In case of *financial2*, although the total number of write access on PRAM is increase when using our algorithm, it can reduce the write counts

compared to other algorithms. We also evaluated the total number of write access with all cases of memory sizes and we find that the patterns of the total number of write access are the same through all cases of the PRAM-to-DRAM ratio.

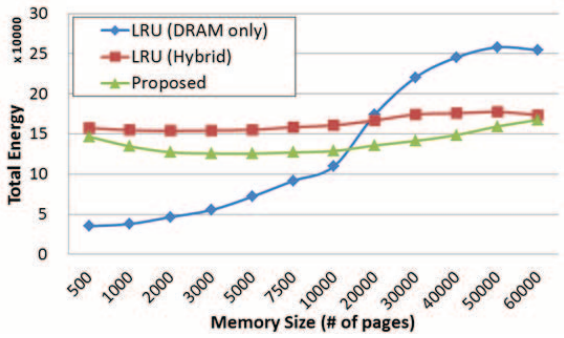


(a) *Financial1*

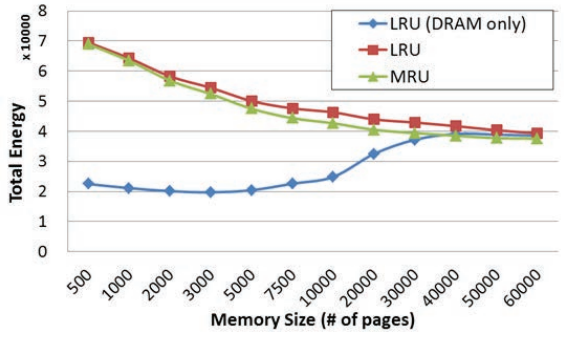


(b) *Financial2*

Figure 10. The total write access count of PRAM on financial workloads

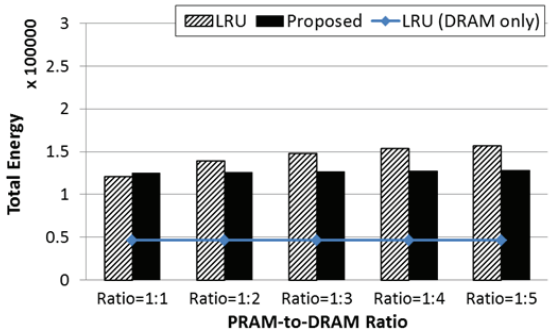


(a) *Financial1*

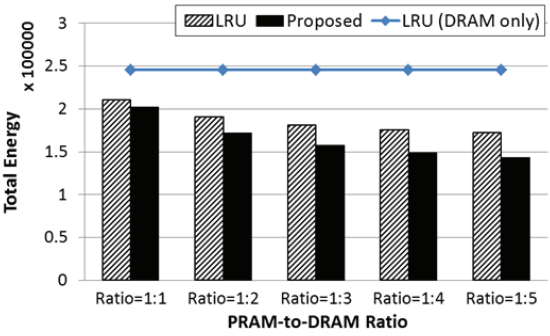


(b) *Financial2*

Figure 12. Consumed energy on memory of proposed algorithm and conventional algorithms on finalcial workloads



(a) 2000-page size of hybrid memory



(b) 40000-page size of hybrid memory

Figure 13. Consumed energy on the hybrid memory of proposed algorithm and LRU algorithm with different PRAM-to-DRAM ratios on finalcial1 workload

4.2.4 Total Energy

We measured the total energy, which consists of operation energy and idle energy. Operation energy is the energy when memory accesses occur. During the evaluation, we can obtain the total access count for the various memory sizes. By using measured count values, we can calculate the operation energy with the read and write energy of both DRAM and PRAM, as can be seen in table 1. The idle energy is the energy continuously consumed by memory devices even if they do not operate. Idle energy can be calculated according to the operation time and the idle power of each memory. We also measured the operation time, which is the time for running the workload. Figure 12 shows the consumed energy of a memory when using the proposed algorithm and the LRU algorithm. In the case of the LRU algorithm, we use hybrid memory and DRAM-only memory for comparison. DRAM shows

low read and write energy but has large idle power compared to PRAM. Therefore, by increasing the size of the memory, the power consumed by DRAM can be increased, as shown in Figure 12. From this result, we can use a page cache on the hybrid main memory in order to reduce the consumed energy if we use the memory with over 20000-page size. If we consider the results with the hybrid main memory, the proposed page caching algorithm uses lower energy than LRU does. Normally, PRAM write energy is large, but we can reduce the write counts on PRAM so that we can reduce the total energy. We can maximally reduce the total consumed energy by 19.9% for *financial1*.

Figure 13 shows the total energy which a memory consumes with various ratios of PRAM to DRAM, in which we selected two memory sizes. In case of 2000-page size which is lower than 20000-page size, the consumed energy by only DRAM with LRU is the lowest, as can be seen in Figure 13(a). As increasing the ratio of PRAM to DRAM, total energy consumed by memory with both LRU and proposed algorithms tend to increase. It is because the total number of write access on PRAM is increasing as increasing the ratio of PRAM to DRAM, as shown in Figure 11(a). However, the idle power of DRAM cannot affect the total energy because a size of DRAM is still small. In case of 40000-page size, DRAM size is large so that its consumed power occupies most of the total energy. Therefore, as increasing the ratio of PRAM to DRAM, total energy consumed by memory is decreasing even if the write-access counts on PRAM are increased, as shown in Figure 13(b). When comparing the consumed energies by the proposed and LRU algorithms, the proposed algorithm uses the lower power compared to the power by LRU through the all cases of PRAM-to-DRAM ratios because the proposed algorithm can reduce the total access counts on PRAM.

5. FUTURE WORK

As to future work, we need to evaluate our algorithm with more workloads. In our evaluation, we used only two workloads, which show the write-bound and read-bound features. Therefore, we have to explore our algorithm with various characteristics such as sequential read/write patterns or mixed write and read patterns.

We also used a trace-driven simulator to evaluate our algorithm with traces. For future work, we will apply our algorithm to the page cache algorithm of the Linux kernel code and evaluate it with benchmark programs like SPEC. In order to implement our algorithm, we will have to emulate PRAM because PRAM is not currently available. By analyzing the read/write time for PRAM, we can emulate PRAM with DRAM, albeit with software delay.

6. CONCLUSIONS

We propose a new page caching algorithm for a hybrid main memory. The hybrid main memory is organized by heterogeneous types of memories, which have different properties of the access latency, density, and endurance. In modern computer systems, a large amount of main memory is used as a page cache to hide disk access latency. Because the conventional page caching algorithms only deal with uniform access latency and endurance, a new page caching algorithm is needed. When using PRAM to make a hybrid main memory, the important things to consider are the long write latency and the low endurance of PRAM. Therefore, we proposed a page caching algorithm with page monitoring and migration schemes to keep read-bound access pages in PRAM and write-bound access pages in DRAM.

The experimental results show that out algorithm can reduce the total number of write accesses by a maximum of 52.9%. Our algorithm shows that a hit ratio is similar to the hit ratio of the conventional page caching algorithms, such as LRU, LIRS, and CLOCK-Pro. Our algorithm minimizes the write access of PRAM while maintaining the cache hit ratio. Therefore, we can enhance the average page cache performance and reduce the endurance problem in hybrid main memory. In addition, we can reduce the consumed energy by a maximum of 19.9%.

7. ACKNOWLEDGMENTS

The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.

8. REFERENCES

[1] UMass TraceRepository, *http://traces.cs.umass.edu/index.php/Storage/Storage*.

[2] Aven, E. G. Coffmann, and I. Kogan. *Stochastic Analysis of Computer Storage*. Amsterdam: Reidel, 1987.

[3] S. Bansal and D. S. Modha. CAR: Clock with Adaptive Replacement. *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004.

[4] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[5] R. W. Carr and J. L. Hennessy. WSClock - A Simple and Effective Algorithm for Virtual Memory Management. *Proceedings of the eighteh ACM symposium on Operating systems principles (SOSP'81)*, pages 87–95, 1981.

[6] E. G. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, 1973.

[7] F. J. Corbato. A Paging Experiment with the Multics System. *MIT Project MAC Report MAC-M-384*, May 1968.

[8] Dan and D. Towsley. An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 143–152, April 1990.

[9] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM*, 11(5):323–333, May 1968.

[10] G. Dhiman, R. Ayoub, and T. Rosing. Pdram:a hybrid pram and dram main memory system. *In Proceedings of the 46th Annual Design Automation Conference*, July 2009.

[11] E. Horowitz, D. D. Sleator, and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, February 1985.

[12] S. Jiang and X. Zhang. Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. *In Marina Del Rey*, pages 31.42. ACM Press, 2002.

[13] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. *Proceedings of the 20th VLDB Conference*, pages 297.306, 1994.

[14] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. *Proceedings of the 36th annual international symposium on Computer architecture*, June 2009.

[15] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies. *IEEE Transactions on Computers*, 50(12):1352.1361, December 2001.

[16] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. *Proceedings of the ACM SIGMOD international Conference on Management of data*, pages 297.306, May 1993.

[17] K. H. Park, Y. Park, W. Hwang, and K.-W. Park. Mn-mate: Resource management of manycores with dram and nonvolatile memories. *12th IEEE International Conference on HPCC*, September 2010.

[18] Y. Park, S. K. Park, and K. H. Park. Linux kernel support to exploit phase change memory. *Linux Symposium*, July 2010.

[19] S. J. Performance and S. Jiang. Clock-pro: An effective improvement of the clock replacement. In *Proceedings of USENIX Annual Technical Conference*, 2005.

[20] M. K. Qureshi, V. Srinivassan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.

[21] J. T. Robinson and M. V. Devarakonda. Data Cache Management Using Frequency-Based Replacement. *Proceedings of ACM SIGMETRICS conference*, pages 134.142, 1990.

[22] H. Seok, Y. Park, and K. H. Park. Migration Based Page Caching Algorithm for a Hybrid Main Memory of DRAM and PRAM. *Proceedings of the 2011 ACM Symposium on Applied Computing*, March 2011.

[23] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid Cache Architecture with Disparate Memory Technologies. *Proceedings of the 36th annual international symposium on Computer architecture*, June 2009.

[24] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. *Proceedings of the 36th annual international symposium on Computer architecture*, June 2009.

[25] Y. Zhou and J. F. Philbin. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. *Proceedings of the USENIX Annual Technical Conference*, pages 91–104, June 2001.

ABOUT THE AUTHORS:

Hyunchul Seok received the BS degree in Electrical Engineering from Pohang University of Science and Technology (POSTECH) in 2001 and the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2008. He is currently working toward the PhD degree in the division of electrical engineering at KAIST. His research interests include storage systems, memory systems, and embedded systems.

Youngwoo Park received the BS and MS degrees in the division of electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2004 and 2006, respectively. He is currently working toward the PhD degree in the division of electrical engineering at KAIST. His research interests include storage systems, flash file systems, and embedded systems. He is a member of the IEEE and the IEEE Computer Society.

Ki-Woong Park received the BS degree in computer science from Yonsei University in 2005 and the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007. He is currently working toward the PhD degree in the Division of Electrical Engineering at KAIST. His research interests include cloud computing systems, security protocol, and network security. . He received a 2009–2010 Microsoft Graduate Research Fellowship. He is a member of the ACM, IEEE and the IEEE Computer Society.

Kyu Ho Park received the BS degree in electrical engineering from Seoul National University, Korea in 1973, the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1975, and the DrIng degree in electrical engineering from the University de Paris XI, France in 1983. He has been a Professor of the Division of Electrical Engineering at KAIST since 1983. He was a president of the Korea Institute of Next Generation Computing for the period 2005-2006. His research interests include computer architectures, storage systems, cloud computing, and parallel processing. He is a member of ACM and IEEE.