Dynamic Data Migration in Hybrid Main Memories for In-Memory Big Data Storage

Hai Thanh Mai, Kyoung Hyun Park, Hun Soon Lee, Chang Soo Kim, Miyoung Lee, and Sung Jin Hur

For memory-based big data storage, using hybrid memories consisting of both dynamic random-access memory (DRAM) and non-volatile random-access memories (NVRAMs) is a promising approach. DRAM supports low access time but consumes much energy, whereas NVRAMs have high access time but do not need energy to retain data. In this paper, we propose a new data migration method that can dynamically move data pages into the most appropriate memories to exploit their strengths and alleviate their weaknesses. We predict the access frequency values of the data pages and then measure comprehensively the gains and costs of each placement choice based on these predicted values. Next, we compute the potential benefits of all choices for each candidate page to make page migration decisions. Extensive experiments show that our method improves over the existing ones the access response time by as much as a factor of four, with similar rates of energy consumption.

Keywords: Big data storage, hybrid main memory, in-memory data management.

Manuscript received Jan. 16, 2014; revised Oct. 15, 2014; accepted Oct. 24, 2014.

This work was supported by the IT R&D Program of MSIP/KEIT, Republic of Korea (10041709, Development of Key Technologies for Big Data Analysis and Management based on Next Generation Memory).

Hai Thanh Mai (corresponding author, mhthanh@etri.re.kr), Kyoung Hyun Park (hareton@etri.re.kr), Hun Soon Lee (hunsoon@etri.re.kr), Chang Soo Kim (cskim7@etri.re.kr), Miyoung Lee (mylee@etri.re.kr), and Sung Jin Hur (sjheo@etri.re.kr) are with the SW·Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

I. Introduction

Dynamic random-access memory (DRAM) has been the most important component of main memories in computer systems for decades. Nowadays, with the ever-increasing amounts of data that require real-time processing, there are even higher demands being placed on DRAM to scale-up performance rates and reduce the pressure on secondary storage devices. For example, besides keeping the indexes, storing and processing big (or entire) amounts of data in DRAM has become an attractive approach for commercial database management applications [1]–[3]. However, despite providing a very high access speed, DRAM has a huge disadvantage in terms of its energy consumption. For instance, in different data centers, about 10% to 30% of the total system energy is consumed by DRAM main memories [4]–[5]. The reason is that DRAM, as a volatile memory, always requires power to retain stored information. Meanwhile, energy efficiency is particularly important in large systems where energy costs can be remarkably high. Therefore, reducing the energy loss substantially while maintaining the high performance of the main memories for long-term in-memory big data storage and management is a challenging problem.

To solve such a problem, a recently promising approach is to combine non-volatile random-access memories (NVRAMs) with DRAM into hybrid main memory systems [6]–[11]. Examples of promising NVRAM technologies include phase-change memory (PRAM), ferroelectric RAM (FRAM), magnetoresistive RAM (MRAM), and flash [11]–[17]. The major advantage of NVRAMs is that they do not need energy for retaining stored data. Thus, in terms of energy efficiency, they are more appropriate than DRAM to store big amounts of data for a long time period. Researchers also believe that the aforementioned NVRAMs can provide more memory capacity and will become cheaper than DRAM in the future [8], [18]–[19]. Despite this, their crucial disadvantage is that their reading and writing speeds are often much worse than those of DRAM. The performance gap is not likely to be negligible soon enough, although some leading companies are working on improving the performance of NVRAMs [15]–[17]. NVRAMs also need higher access energy and have lower write endurance than DRAM. As a result, instead of replacing DRAM entirely by NVRAMs, designing hybrid main memories that consist of both DRAM and NVRAMs is a more favorable approach. In a hybrid memory of this kind, fast but energy-inefficient DRAM usually constitutes a relatively small part of the total memory, with slow but energy-efficient NVRAMs making up the remaining part.

To leverage the attractive attributes while mitigating the negative effects of both DRAM and NVRAMs, a number of data migration methods have been proposed [6]–[11]. Their basic idea is to move "hot" data to DRAM and "cold" data to NVRAMs. The underlying purpose is to exploit the fact that popular server workloads usually include accesses to only relatively small parts of the whole data set or accesses to the major part of the data set for only a short time [5]. Here, hot data is often defined as data that is accessed frequently, while cold data is data that is rarely accessed. The popularly used unit of data migration is an operating system page, typically 4 KB [6]–[10]. Hot data pages in NVRAMs are migrated to DRAM, and cold data pages in DRAM are migrated to NVRAMs. The migrations may happen periodically after a fixed amount of time or immediately when the access frequency values of some pages go beyond a predefined threshold. Different methods use different frequency thresholds to categorize data pages into hot or cold. For example, in an early and inspiring study about hybrid DRAM and PRAM main memory [7], the authors recommend setting the threshold value to 1,000. When the write frequency of a page becomes a multiple of this threshold, it is considered a hot page and will be moved to DRAM if it is residing in an NVRAM. In a later study [8], the authors suggested that an NVRAM page should be migrated to DRAM when its sum of read and write frequency values reaches 32. In another work [10], two consecutive accesses make an NVRAM page become hot and two consecutive non-accesses make a DRAM page become cold, triggering corresponding migrations.

The previous methods for data migration in hybrid main memories have some limitations as follows. First, the migration decision of a data page is made based on its most recently measured access frequency value, which is a value measured in the past. Their underlying assumption is that the access frequency of a page will not change in the future. In fact, the access frequency of a page may increase, decrease, or stay the same over time. Second, the gains and costs of migrations are not comprehensively considered. A data page will generally be migrated to another memory when its access frequency satisfies a certain threshold. Nonetheless, since no fixed threshold value can be good for all workloads all of the time, there can be many situations where the cost of migration is greater than the gain, yet the migration still happens. Finally, previous studies only consider one type of NVRAM in the hybrid memories. It is not clear how the data migration should be done when multiple types of NVRAMs are used in the same system. Different types of NVRAMs have different performance, energy, and investment cost characteristics. The combination of multiple different NVRAMs in one hybrid memory is potentially beneficial for its users.

In this paper, we propose a new data migration method for hybrid DRAM–NVRAMs memories that aims to solve the limitations of its predecessors. We target memory-based storage systems, such as in-memory databases, where a major part of the data (or, all of it in its entirety) is stored in the main memory. First, we propose a procedure to predict access frequency values of the pages in the near future and use the predicted frequency values for making page migration decisions. We use a statistical technique to compute the predicted frequency values, and then revise if necessary the prediction based on automatically learned experience. This procedure helps us to catch the patterns of the changing of the access frequency to make more accurate decisions. Second, we develop a decision-making procedure that measures comprehensively the gains and costs of each migration. We also use a frequency threshold as in the previous methods, but this threshold is no longer the only criteria for deciding the page migrations. Instead, we only use the threshold for finding candidate hot and cold pages. The migration decisions are then made based on the candidate pages' potential benefits, which are computed from their potential gains and costs. Finally, we develop our decision-making procedure in a general setting where there can be multiple NVRAMs in a hybrid main memory, making our method suitable for a broader range of applications. We conducted extensive experiments using seven real-life and standardized data access workloads. The results show that our method can reduce the average access response time by up to a factor of four, while maintaining a similar rate of energy consumption when compared with existing methods.

The remainder of this paper is organized as follows. In Section II, we introduce promising NVRAM technologies and hybrid main memory designs. In Section III, we present our data migration method in detail. Experimental results are presented in Section IV, followed by a review of related work in Section V. Finally, we conclude the paper in Section VI.

II. Background

1. NVRAM Technologies

As we mentioned in the previous section, PRAM, FRAM, MRAM, and flash are currently the most promising NVRAM technologies. Among them, PRAM and flash have received the greatest attention from the research community as well as industry. PRAM is a byte-addressable memory, and the performance of PRAM is worse than DRAM by about a factor of two for reads and a factor of five to six for writes [12]. Recently, a prototype of an 8 Gb PRAM has been announced [16]. Flash is another type of non-volatile memory. In this study, we consider a special kind of flash memory that is capable of being used as a main memory and that can bypass several layers of the traditional communication stack, as has been introduced in the market [17]. The performance of this special kind of new flash is about an order of magnitude worse than DRAM and an order of magnitude better than traditional flash memory such as solid-state drive (SSD). Both PRAM and flash do not consume energy to retain data due to their non-volatile characteristic. It is also believed that they are more scalable than DRAM with multi-level cell (MLC) capability and will have considerably lower prices than DRAM in the future [8], [18]–[19]. Therefore, they can be used to provide a much higher capacity for the memory system than DRAM can within the same budget. More detailed descriptions about the characteristics, architectures, and principles of operation of these memories can be found in many other studies (for example, [12], [17]).

2. Hybrid Main Memory Designs

There are two general designs for DRAM–NVRAMs hybrid memories. The first one, called hierarchical design, is to use DRAM as a cache for lower layer NVRAM real storage. The second one, called flat design, is to put all memories including DRAM and all NVRAMs at the same level. Due to page limitation, we omit the descriptions of these two designs here. Further information, for those interested, can be found in [6]–[10]. In general, the flat design is often preferable to the hierarchical one, although there is no consensus on which design is better [7]–[9]. We think that the two designs complement each other and are useful in different cases. However, we adopt the flat design in this paper.

III. Dynamic Data Migration Method

1. Overview

Figure 1 shows a sketch of our data migration method. Data

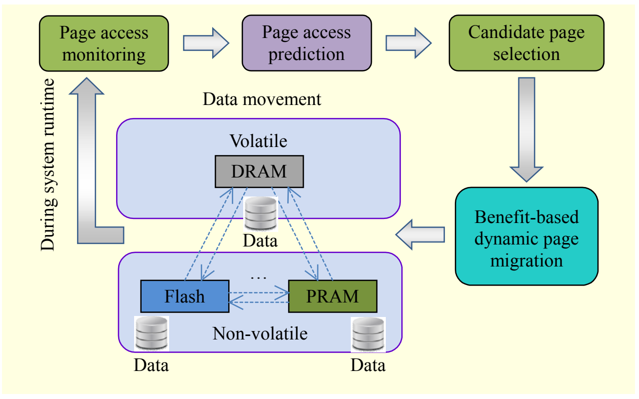

Fig. 1. Method overview.

is stored in both DRAM and NVRAMs and migrated among the memories dynamically during system runtime. We monitor the access frequency of the data pages. Then, based on the historical frequency values, we predict the near-future access frequency of the pages. Predicted frequency values are then used for selecting candidate pages for migrations. Next, potential benefits of migration choices are computed for each candidate page, resulting in a candidate page being migrated to the most beneficial memory type. We present the details of our method in the following subsections.

2. Page Access Monitoring

While the system runs, we monitor the read and write frequency values of all data pages for a number of time windows in the past. Let d denote the number of time windows for which we keep the measured frequency values. Let W_i denote the ith time window (i is an integer and $-d \leq i \leq 0$). We use negative values for i to indicate that the corresponding time windows are in the past. Greater values of i refer to more recent time windows where W_{-1} and W_{-d} are, respectively, the most and least recent ones. All time windows have the same length. Now, consider a certain data page p_j in a time window W_i. We use $N_r^{ms}(i, j)$, $N_w^{ms}(i, j)$, and $N^{ms}(i, j)$ to represent the measured read, write, and access (either read or write) frequency values of p_j in W_i, respectively. These values are stored in a small part of the DRAM. Assume that the page size is 4 KB, four bytes are used for each page ID, one byte for each frequency value, and $d = 5$, then the storage overhead for the historical frequency data is only 0.34% of the total memory.

3. Page Access Prediction

Given historical frequency values, we compute the predictive read and write frequency values for the data pages in a time window W_0 in the future. Note that W_0 is the nearest future time window, next to the current time point and the time window W_{-1} in the past. Here, a simple prediction strategy used

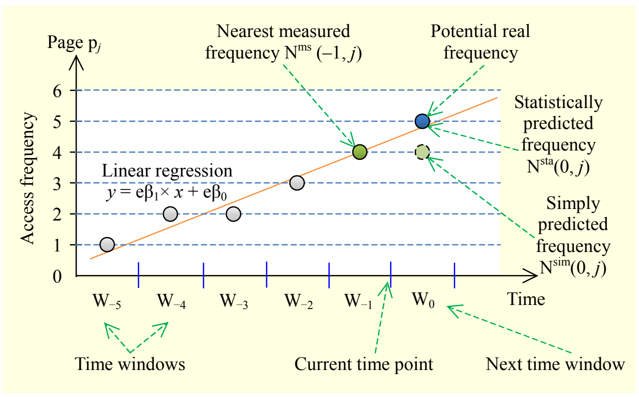

Fig. 2. Simple prediction strategy may fail and statistical prediction strategy may be better.

in previous data migration methods is to set the predicted value the same as the most recently measured one (that is, the value measured at W_{-1}). The underlying assumption is that the access frequency values of all pages will not change in the future. However, in practice, the access frequency of each page can vary dynamically over time, either increasing or decreasing or staying the same. For example, when a special event happens, there will be an increasing need for loading or writing to the pages containing the related data during a certain time interval. Figure 2 illustrates such a situation when the access frequency values (represented by small circles) of a page p_j are having an increasing trend during the last five time windows. Since the most recently measured value $N^{ms}(-1, j)$ is four, a simple prediction for the future value at W_0, denoted by $N^{sim}(0, j)$, is also four. However, because of the increasing trend, a more likely real frequency value, in this situation, is five.

Therefore, in this paper, we propose to use a statistical model named simple linear regression to predict the future access frequency values. This model has relatively high prediction accuracy and can be computed very quickly. Note that although there are more accurate prediction models in existence, they are not appropriate due to the increase in complexity that they cause, which then results in high processing times. As described in [20], the simple linear regression model for a list of data points $P = \{(x_1, y_1), \ldots, (x_v, y_v)\}$ has the form $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ where the observed value of the dependent variable y_i is composed of a linear function, $\beta_0 + \beta_1 x_i$, of the explanatory variable x_i and an error term ϵ_i. The parameters β_0 and β_1 are called the intercept parameter and slope parameter, respectively. The error term ϵ_i is commonly assumed to follow a Gaussian distribution $N(0, \sigma^2)$ for some error variance σ^2 [20]. This implies that the values y_i's ($1 \le i \le v$) are assumed to be observations from the random variables $Y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$, where N denotes a Gaussian distribution. Using "least squares" fitting techniques, we can obtain the estimates of the parameters β_0, β_1, and σ^2 denoted by parameters $e\beta_0$, $e\beta_1$, and $e\sigma^2$, respectively (please refer to [20] for more details).

Here, $e\beta_1$ and $e\beta_0$ represent the line $y = e\beta_1 \times x + e\beta_0$ that approximates the distribution of the points in P. In applying the linear regression model to predict the access frequency of a certain page p_j, we set up P to be the list of d points that correspond serially to d time windows from W_{-d} to W_{-1}. More specifically, for each point (x_i, y_i) that corresponds to a time window W_t we set $x_i = t$ and $y_i = N^{ms}(t, j)$, which is the access frequency value measured at W_t. Let us consider Fig. 2 again, here, the linear regression model is able to catch the increasing pattern of the frequency values more correctly. Let $N^{sta}(0, j)$ represent the statistically predicted frequency value by the linear regression model at W_0. In this example, $N^{sta}(0, j)$ is five and is a better prediction than the value of four that we would have obtained using the simpler aforementioned strategy.

Experience-based strategy switching. Although the statistical prediction strategy is often better than the simple prediction strategy, there is a case when the latter strategy is better than the former. That is, when the statistical strategy mispredicts the trend, from increasing to decreasing or from decreasing to increasing. To solve this problem, we propose a strategy selection procedure that switches between the two prediction strategies to choose the better one for getting the final predicted value. Our idea is to find the strategy that results in the lower prediction error in W_{-1} and then to use that strategy for the prediction of W_0. More specifically, after d time windows have passed and d corresponding frequency values have been measured for each page, our procedure works as follows:

- At each time point when we wish to predict a frequency value for a page p_j, we compute the two values $N^{sta}(0, j)$ and $N^{sim}(0, j)$ by using the two aforementioned prediction strategies. Now, imagine that we have proceeded one time window and that the previous W_0 has become W_{-1}. Similarly, $N^{sta}(0, j)$ and $N^{sim}(0, j)$ have become $N^{sta}(-1, j)$ and $N^{sim}(-1, j)$, respectively. We also have the real value measured at W_{-1}, that is $N^{ms}(-1, j)$.
- Let $e^{sta}(j)$ and $e^{sim}(j)$ be the *error of statistical prediction* and *error of simple prediction* strategies, respectively. We compute $e^{sta}(j)$ as the absolute difference between the statistically predicted value and the real measured value at W_{-1}.

$$e^{sta}(j) = |N^{sta}(-1, j) - N^{ms}(-1, j)|. \tag{1}$$

- Likewise, we compute $e^{sim}(j)$ as the absolute difference between the simply predicted value and the real measured value at W_{-1}.

$$e^{sim}(j) = |N^{sim}(-1, j) - N^{ms}(-1, j)|. \tag{2}$$

- Next, we compare the two prediction errors to choose the strategy that performed better in W_{-1} to use for W_0. Let $N^{pre}(0, j)$ be the finally predicted value. If $e^{sta}(j)$ is less than

$e^{sim}(j)$, then we set $N^{pre}(0, j) = N^{sta}(0, j)$; otherwise, we set $N^{pre}(0,j) = N^{sim}(0,j)$.

In summary, we jump back and forth between the two prediction strategies to find the most reasonable value.

4. Candidate Page Selection

After computing predicted read and write frequency values for each page, we use these values to choose candidate pages for migration. For each memory M_i, let $L_i = \{cp_0, \ldots, cp_{\eta_i - 1}\}$ be the list of candidate pages in M_i. Each element cp_k ($0 \le k \le \eta_i - 1$ and $\eta_i = |L_i|$ represents the number of elements in L_i) is a triple $\langle p_k, N_r^{pre}(0, k), N_w^{pre}(0, k)\rangle$ that represents a candidate page p_k and its predicted read and write frequency values.

If M_i is DRAM, then we add to L_i all "cold" pages in M_i. Otherwise (that is, if M_i is an NVRAM), we add to L_i all "hot" or "potentially hot" pages in M_i. We set a frequency threshold, denoted by t_f. A page is defined to be cold if the sum of its predicted read and write frequency values is less than t_f. In contrast, a page is defined to be hot if the sum of its predicted read and write frequency value is greater than or equal to t_f. A page is identified to be potentially hot if the time passed from the last access until the current time point is greater than the time distance between the two most recent accesses. More specifically, consider a page p_j. Let t_{-2} and t_{-1} (where $t_{-2} < t_{-1}$) denote the second-most recent and most recent time points that p_j was accessed, respectively. Moreover, let t_0 (where $t_{-1} < t_0$) be the current time point. Then, if $t_0 - t_{-1} > t_{-1} - t_{-2}$, then p_j is considered as a potentially hot page. We introduce the notion of potentially hot to use recency as a supplementary source of information for frequency. We treat both hot and potentially hot pages equally, and for brevity, will mention all of them as hot pages in our computations.

The candidate lists of all memories are updated after every time window. We sort all candidate pages in each list L_i based on their "integrated predicted access frequency" values, represented by if (cp_k) for each candidate page cp_k and defined as follows. We first compute $\theta_i = T_w(M_i)/T_r(M_i)$, which is a balancing constant to reflect the relative difference between the time cost to write a page to M_i (denoted by $T_w(M_i)$) and the time cost to read a page from M_i (denoted by $T_r(M_i)$). Then, if $\theta_i \ge 1$, we define

$$\mathrm{if}\left(cp_k\right) = N_r^{pre}\left(0, k\right) + \theta_i \cdot N_w^{pre}\left(0, k\right); \tag{3}$$

otherwise, we define

$$\mathrm{if}\left(cp_k\right) = \frac{1}{\theta_i} \cdot N_r^{pre}\left(0, k\right) + N_w^{pre}\left(0, k\right). \tag{4}$$

If two candidate pages have the same integrated frequency values, then the order among them is determined based on their predicted write frequency values. The sorting of the candidate pages is done so that cp_0 is always the coldest page if L_i is the candidate list of DRAM, or hottest page if L_i is the candidate list of an NVRAM. It means that, by frequency values, the candidate pages of DRAM are ordered ascendingly, while those of NVRAMs are ordered descendingly.

Table 1. Notations.

Notation	Descriptions
n	Number of memories
$M = \{M_0, \ldots, M_{n-1}\}$	Set of memories
M_i	i-th memory
m	Number of data pages
$P = \{p_0, \ldots, p_{m-1}\}$	Set of data pages
p_k	kth data pages
W_t	tth time window
$N_r^{pre}(t,k), N_w^{pre}(t,k)$	Predicted read, write frequencies of p_k at W_t
$T_r(M_i)$, $T_w(M_i)$	Time to read or write a data page from or to M_i
$E_r(M_i)$, $E_w(M_i)$	Energy to read or write a data page from or to M_i
$E_{id}(M_i)$	Idle energy to retain a data page in M_i
T_{id}	Idle time

5. Benefit-Based Dynamic Page Migration

In this section, we first present the migration of one candidate page in one candidate list. Then, we describe the migrations of all candidate pages in all candidate lists. Table 1 shows some notations that are frequently used.

A. Migration of One Page

Let n be the number of memories in the system and $M = \{M_0, \ldots, M_{n-1}\}$ be the set of memories. Without loss of generality, assume that M_0 is DRAM and that other memories are NVRAMs. Consider a data page p_k that is residing in a memory M_i and is being listed in M_i's candidate list. There are $(n - 1)$ possible choices to migrate p_k to a new memory M_j, plus one choice to *not* migrate. The choice of no migration can be seen as a special migration choice where i is equal to j. Among these n choices, the best one should be the one that maximally improves both the access response time and energy consumption during the next time window W_0 in the future. In the following, we propose a procedure to make the best migration decision. Our idea is to put p_k in the memory that can result in the highest potential benefit. The main steps of the procedure are described as follows:

- Let $T_i(k)$ and $E_i(k)$ denote the total access time and total energy consumption required by the system when p_k resides in a certain memory M_i during W_0, respectively. We first compute $T_i(k)$ and $E_i(k)$ based on the predicted read and write frequency values of p_k and the amounts of time and energy required for reading from and writing to M_i as

$$T_i(k) = N_r^{pre}(0, k) \times T_r(M_i) + N_w^{pre}(0, k) \times T_w(M_i), \tag{5}$$

$$\begin{aligned} E_i(k) = {} & N_r^{pre}(0, k) \times E_r(M_i) + N_w^{pre}(0, k) \times E_w(M_i) \\ & + T_{id} \times E_{id}(M_i) \end{aligned} \tag{6}$$

where the meanings of $N_r^{pre}(0, k)$, $N_w^{pre}(0, k)$, $T_r(M_i)$, $T_w(M_i)$, $E_r(M_i)$, $E_w(M_i)$, T_{id}, and $E_{id}(M_i)$ are presented in Table 1.

- Next, let $C^T(k, i \to j)$ and $C^E(k, i \to j)$ be the time and energy costs of migrating p_k from M_i to M_j, respectively. We compute $C^T(k, i \to j)$ as the sum of the time to read one page from M_i and the time to write one page to M_j. Similarly, we compute $C^E(k, i \to j)$ as the sum of the energy to read one page from M_i and the energy to write one page to M_j.

$$C^T(k, i \to j) = T_r(M_i) + T_w(M_j), \tag{7}$$

$$C^E(k, i \to j) = E_r(M_i) + E_w(M_j). \tag{8}$$

In the case of no migration, these costs are zeros.

- Then, let $B^T(k, i \to j)$ and $B^E(k, i \to j)$ denote the time and energy benefits of migrating p_k from M_i to M_j. We compute the values of $B^T(k, i \to j)$ and $B^E(k, i \to j)$ as a fraction between the time and energy requirements between the two memories, as follows:

$$B^T(k, i \to j) = T_i(k) / (T_j(k) + C^T(k, i \to j)), \tag{9}$$

$$B^E(k, i \to j) = E_i(k) / (E_j(k) + C^E(k, i \to j)). \tag{10}$$

In the case of no migration, we set $B^T(k, i \to i) = B^E(k, i \to i) = 1$.

- Next, we compute the *integrated time and energy benefit* of migrating p_k from M_i to M_j, denoted by $B^{TE}(k, i \to j)$, by multiplying the time and energy benefits.

$$B^{TE}(k, i \to j) = B^T(k, i \to j) \times B^E(k, i \to j). \tag{11}$$

- Now, among the n choices of migration or no migration, we choose the one that results in the highest value of $B^{TE}(k, i \to j)$. In other words, let j^* be the memory that corresponds to the highest benefit. If $j^* \neq i$, then we migrate p_k from M_i to M_{j^*}. Otherwise, we do nothing. Note that there are not always n migration choices. When a memory M_j is already full, we do not consider migrating p_k to M_j and set the corresponding $B^{TE}(k, i \to j)$ value to zero.

B. Migration of All Pages

The procedure MigrateAllCandidates, shown in the next page, illustrates the migrations of all candidate pages of all memories. The first input parameter is a set $L = \{L_0, \dots, L_{n-1}\}$ that contains the candidate lists of all memories. The second input parameter is the number n of memories. Without loss of generality, we assume that L_0 is the candidate list of DRAM and that L_i's ($i > 0$) are those of NVRAMs. Let $\langle \text{idx}(0), \dots, \text{idx}(n-1) \rangle$ be a list of n variables that corresponds to n candidate lists from L_0 to L_{n-1}. Each idx(i) stores the index of the candidate page $cp_{idx(i)}$ in the list L_i that is being considered for migration. In the procedure, we first initialize all idx(i)'s to be zeros (Line 1). Then, we consider page migration in a round-robin fashion between DRAM and NVRAMs by repeating the following two steps:

- *Consider migrating a DRAM cold page* (*Lines* 3 *to* 5). Since all candidate lists have been sorted as described in Section III-4, the candidate page $cp_{idx(0)}$ of the list L_0 represents the currently "coldest" page in DRAM that has not been considered for migration. We compute the index of the best memory to put this page by algorithm ComputeBestMem (described below) and store this index in a variable named bM (Line 3). Then, we check if the best choice is not DRAM (that is, bM $\neq$ 0), then we migrate this candidate page to the new memory (Lines 4); otherwise, we do nothing. The value of idx(0) is increased by one (Line 5), so that we will consider the second coldest page in DRAM in the next loop.
- *Consider migrating an NVRAM hot page* (*Lines* 6 *to* 13). First, we find the index of the NVRAM that contains the hottest candidate page in terms of integrated access frequency (Lines 6 to 10). Let hNV and hiF be the variables that store the hottest NVRAM's index and the highest candidate page's frequency value, respectively. We go through the candidate lists of all NVRAMs, check the integrated frequency values of their currently hottest pages, and keep track of the highest frequency value and the corresponding NVRAM's index. Next, given the index of the NVRAM containing the hottest candidate page, stored in hNV, we consider the candidate page $cp_{idx(hNV)}$ of the candidate list L_{hNV} for migration. To make a decision on this page, we compute the index of the best memory for placing it by algorithm ComputeBestMem (Line 11). If the best choice is not the current memory (bM $\neq$ hNV), then we migrate the candidate page to the new memory (Lines 12); otherwise, we do nothing. The value of idx(hNV) is then raised by one (Line 13).

The *repeat-until* loop stops when idx(i) is equal to the size of L_i for all values of i. In other words, the above two steps are repeated until all candidate pages have been considered.

```
**Procedure** MigrateAllCandidates(L, $n$)
**Input:** $L = \{L_0, \dots, L_{n-1}\}$ – set of candidate lists
           $n$ – the number of memories
1:  Initialize $\langle idx(0), ..., idx(n-1) \rangle$ to zeros
2:  **repeat**
        /* 1. Consider migrating a DRAM cold page */
3:      bM $\leftarrow$ ComputeBestMem($cp_{idx(0)}$, 0, $n$)
```

```
4:  If bM $\neq$ 0, migrate $cp_{idx(0)}$ from $M_0$ to $M_{bM}$
5:  idx(0) $\leftarrow$ idx(0) + 1
    /* 2. Consider migrating an NVRAM hot page */
6:  hNV $\leftarrow$ 1
7:  hiF $\leftarrow$ if($cp_{idx(1)}$)
8:  **for** $r$ = 2 to ($n$ – 1) **do**
9:      If if($cp_{idx(r)}$) > hiF,   then
        hiF $\leftarrow$ if($cp_{idx(r)}$) and hNV $\leftarrow$ $r$
10: **end for**
    /* $cp_{idx(hNV)} \in L_{hNV}$ is now the "hottest" NVRAM page */
11: bM $\leftarrow$ ComputeBestMem($cp_{idx(hNV)}$, hNV, $n$)
12: If bM $\neq$ hNV, migrate $cp_{idx(hNV)}$ from $M_{hNV}$ to $M_{bM}$
13: idx(hNV) $\leftarrow$ idx(hNV) + 1
14: **until** idx($i$) is equal to the size of $L_i$ for all $i$'s
```

```
**Algorithm** ComputeBestMem($p$, $i$, $n$)
**Input:**  $p$ – a candidate page
            $i$ – the index of the current memory
            $n$ – the number of memories
**Output:** bM – the index of the best memory for migration
1:  bM $\leftarrow$ 0; maxB $\leftarrow$ 0
2:  **for** $j$ = 0 to ($n$ – 1) **do**
3:      Compute $B^{TE}(p, i \rightarrow j)$
4:      If $B^{TE}(p, i \rightarrow j)$ > maxB,   then
        maxB $\leftarrow$ $B^{TE}(p, i \rightarrow j)$, bM $\leftarrow$ $j$
5:  **end for**
6:  Return bM
```

The algorithm ComputeBestMem computes the best memory for migration. The inputs include a candidate page p, the index of the current memory i, and the number of memories n. The output is the index of the best memory for migration bM. Let maxB represent the maximum benefit among all migration choices. First, we initialize both bM and maxB to be zeros (Line 1). Then, we examine all n choices from zero to (n – 1) (Lines 2 to 5). For each choice j, we compute the integrated time energy benefit of migrating p from M_i to M_j, denoted by $B^{TE}(p, i \rightarrow j)$ (Line 3). If $B^{TE}(p, i \rightarrow j)$ is greater than maxB, then we update the values of maxB to $B^{TE}(p, i \rightarrow j)$ and bM to j (Lines 4). When the for loop finishes, bM contains the index of the memory having the highest potential benefit. Finally, we return bM (Line 6).

The worst case time complexity of algorithm ComputeBestMem is O(n) and that of procedure MigrateAllCandidates is O(nm), where n is the number of memories and m is the number of data pages. We omit the detailed analysis here due to page limitation.

IV. Performance Evaluation

1. Experimental Setup

We compared the proposed data migration method with previous ones in a hybrid memory system that consists of three memory types; namely DRAM, PRAM, and flash. Note that the proposed method is also applicable to any hybrid memory system having more than three memory types. The timing, energy, and density characteristics of these memories are taken from [11]–[12] and [19]. The compared methods include PDRAM [7], RaPP [8], PaPA [10], LMRU (a comparative method that moves the *least recently used* pages from DRAM to NVRAMs and the *most recently used* pages from NVRAMs to DRAM), and PrBDR — the predicted benefit-based dynamic data migration method proposed in this paper.

We used seven different data access traces; namely Financial1 (Fi1), Financial2 (Fi2), TPC-C, TPC-H, CH(80-20), CH(50-50), and CH(20-80). Traces Fi1 and Fi2 are two real-life traces from online transaction processing (OLTP) applications running at two large financial institutions [21]. TPC-C is an OLTP trace from the standard TPC-C benchmark [22]. TPC-H is an online analytical processing (OLAP) trace from the standard TPC-H benchmark [23]. CH(80-20), CH(50-50), and CH(20-80) are three traces from a merged OLTP-OLAP benchmark [24]. The notation CH(N1-N2) represents a hybrid trace in which N1% of OLTP queries and N2% of OLAP queries are used (N1 + N2 = 100%). The read/write ratios of these workloads are 23.2/76.8, 82.3/17.7, 82/18, 100/0, 87.7/12.3, 90.5/9.5, and 93.3/6.7, respectively.

By default, the sizes of DRAM, PRAM, and flash are 1 GB, 2 GB, and 2 GB, respectively, when Fi's traces are used. The corresponding sizes are 2 GB, 4 GB, and 4 GB for other traces. We use the average response time (that is, average execution time) and average energy consumption as the performance metrics. The measurement units of these metrics are microseconds (us) and millijoules (mJ), respectively.

2. Result Analysis

Figures 3 and 4 show the average response time and average energy consumption of the methods with various workloads. With regard to response time, the proposed method PrBDR is about 1.5 to 4.5 times better than the other methods. For example, when the workload Fi1 is used, the improvement ratios of PrBDR over PDRAM, RaPP, PaPA, and LMRU are 1.58, 1.51, 1.69, and 1.44, respectively. Meanwhile, when Fi2 is used, the corresponding ratios are higher at 4.73, 2.31, 4.13, and 3.14. The reduced response times of PrBDR come from the two main factors that we have done to overcome the limitations of previous methods. First, we predict the access frequency values of the data pages in the near future, and then use the predicted values, instead of the historical values, for assessing the need of migrations. Second, we measure fully the gains and costs of all migration choices, and then put the data

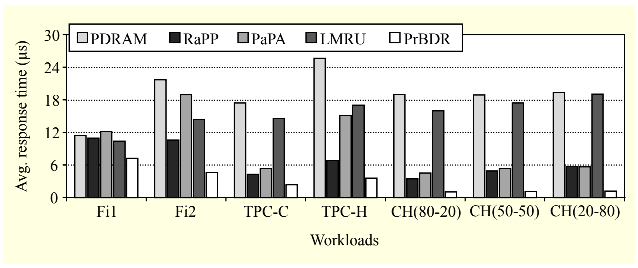

Fig. 3. Measuring average response time with various workloads.

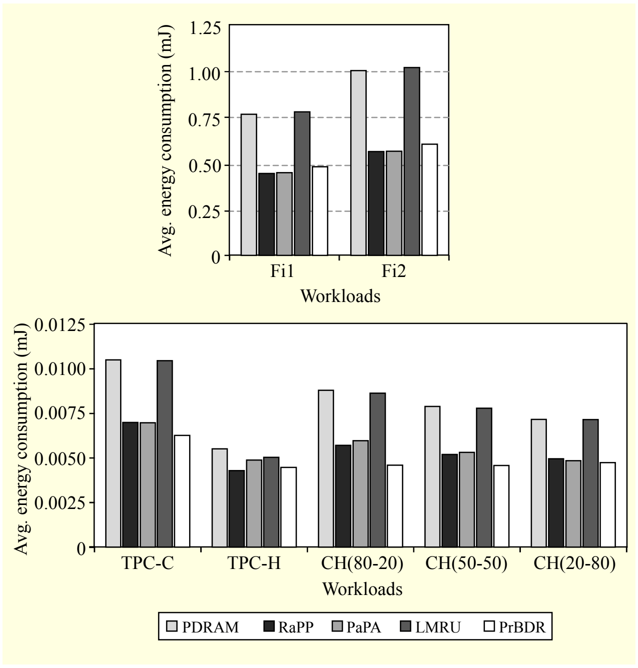

Fig. 4. Measuring average energy consumption with various workloads.

pages in the memories having the highest potential benefits, instead of simply migrating the pages every time their access frequencies go beyond a certain threshold. As for energy consumption, PrBDR is at the middle position compared to other methods. Specifically, the amounts of energy consumed by PrBDR are 15% to 44% lower than those of PDRAM and LMRU, while being often similar to, or sometimes 5% to 25% higher than, those of RaPP and PaPA. Since the proposed method does not always consume the lowest amount of energy, it is more suitable to be used in hybrid memory systems where response time reduction is preferable. Notice that even though some previous methods may consume a little less energy than PrBDR, adjusting those methods to use more energy to obtain lower response times is very difficult or even impossible since finding fixed threshold values that are good for all workloads all of the time is not easy.

Figure 5 presents the performance of the methods when we vary the ratio of the memories' sizes. First of all, Figs. 5(a) and

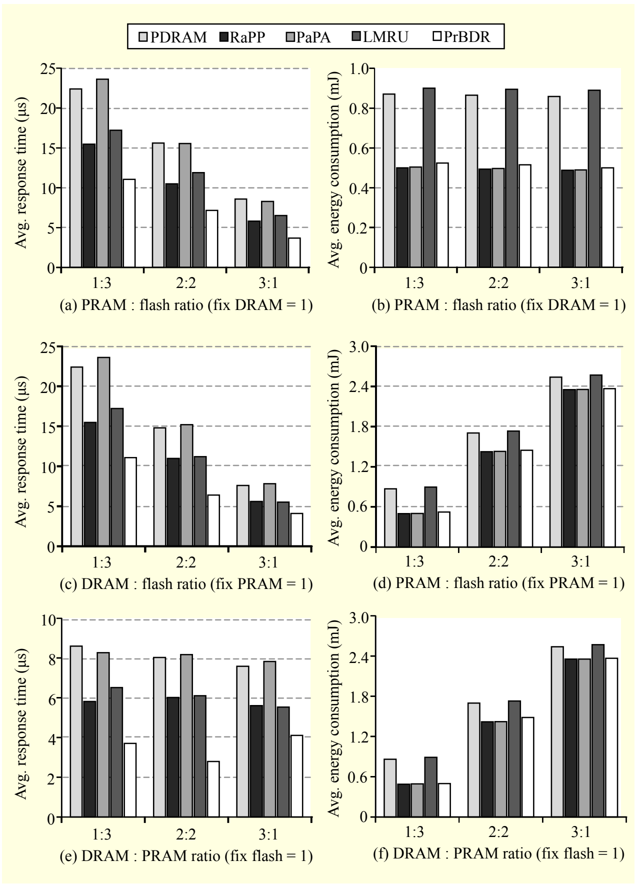

Fig. 5. Measuring average response time and energy consumption with various ratios of memory sizes.

5(b), respectively, demonstrate the response time and energy consumption of the methods when we fix the ratio of DRAM and change the ratios of PRAM and flash. From Fig. 5(a), we see that when the amount of PRAM is raised and the amount of flash is reduced, the average response times of all methods decrease quickly because both the read and write speeds of PRAM are much better than those of flash. However, Fig. 5(b) shows that the average energy consumptions of the methods decrease only a little for the same trend of changing the memory ratio. This is because the system's energy consumption is dominated by the energy consumed by accesses to a small number of pages in flash, which have not been moved to DRAM or PRAM yet, and the energy consumed by DRAM. Next, consider Figs. 5(c) and 5(d) where the ratio of PRAM is fixed at one and the ratios of DRAM and flash are changed. Due to DRAM's characteristics, the response times decrease while the amounts of energy consumption increase significantly for all methods. Note that from Figs. 5(a) and 5(c), it is clear that increasing the amount of either DRAM or PRAM will result in similarly high response time improvements. However, because DRAM is a volatile memory, while PRAM is a non-volatile one, using more DRAM results in a high negative impact on energy consumption, but using more PRAM does not, as we can observe in Figs. 5(b) and 5(d). Now, consider Figs. 5(e) and 5(f), where we do not alter the ratio of flash but adjust those of DRAM and PRAM. Interestingly, even though DRAM is over two-times faster than PRAM, the hybrid memory made mostly from DRAM has an average response time that is just a little lower than the hybrid memory made mostly from PRAM. The reason is that most hot pages have been moved to DRAM, and the penalty for accesses to hot pages still residing in PRAM is low. Nevertheless, there is a fast increase in the amount of energy consumption when DRAM occupies larger proportions. Overall, the proposed method obtains significantly lower response time than the others with various ratios of memory sizes. Besides this, the energy consumption rates of our method are also similar to or sometimes lower than those of other methods.

V. Related Work

Qureshi and others present in [6] one of the first studies about hybrid main memories where DRAM is combined with PRAM. The authors use DRAM as a cache for PRAM and manage the data movement at page level. On a page fault, they fetch the required page from the hard drive to only DRAM and allocate a space for that page in PRAM, but do not write to PRAM until the page is evicted from DRAM later. When a dirty page is evicted from DRAM, only the dirty data of the page is written to PRAM. In an inspiring work [7], Dhiman and others locate DRAM and PRAM at the same level (flat layout) instead of using DRAM as a cache. An access map is managed in PRAM to store the write frequency values of pages in PRAM. When the frequency value of a PRAM page becomes a multiple of a given threshold, this write-intensive page will be swapped with a victim page in DRAM. In [8], Ramos and colleagues also organize DRAM and PRAM in a flat layout and perform page-based data migration. Both read and write frequency values are taken into account when making page migration decisions. Pages in DRAM and PRAM are arranged into 15 queues based on their access frequency values. Among the queues, a page is promoted to a higher queue or demoted to a lower queue when its access frequency is updated. Whenever the frequency of a PRAM page reaches a migration threshold, it is promoted to one of the top queues and migrated to DRAM. In [10], the authors suggest to use an on-chip DRAM at the upper layer and an off-chip DRAM together with a PRAM in a flat layout at the lower layer. For page migration between DRAM and PRAM, they record the access frequency of the pages and place hot pages in DRAM and cold pages in PRAM. Here, they categorize a page as hot when it has two consecutive accesses or cold when it has two consecutive non-accesses. In [9], other authors propose to use four unused bits in the page table entry to store write information. More recent bits are assigned higher weights to exploit temporal locality. Then, instead of looking at individual pages, they consider groups of physically continuous pages for migration. Average group write frequency values are computed, and groups that have such values greater than a hot threshold or lower than a cold threshold will be migrated to DRAM or PRAM, respectively. Choi and others focus on the flat layout of the memories and assume that all page accesses are known in advance [14]. Then, they propose an evaluation framework to compute the theoretically maximum performance of a hybrid main memory. Instead of PRAM, the authors in [25] propose a hybrid DRAM and flash SSD memory system. The flash memory is treated as a transparent extension of DRAM. Applications can access the flash memory via the page-based virtual memory interface, but internally, the system works at arbitrary-sized objects level. In the virtual memory, each object is allocated a whole page even if it needs much less than a page, and the object is always placed at the start of the page. DRAM is separated into two parts; namely, a page buffer and an object cache. An extension of this work is presented in [11], where a new kind of flash memory connected through PCIe ports is used.

VI. Conclusion

We proposed a new dynamic data migration method for hybrid DRAM–NVRAMs main memories. Like previous studies, we attempt to identify and migrate hot and cold data pages among DRAM and NVRAMs to take advantage of DRAM's high speed and NVRAMs' low power consumption. However, to make the process much more effective, our method goes further by predicting carefully the data access frequency in the near future and computing comprehensively the potential benefits of all migration choices before making decisions. Our method is the first that overcomes the use of fixed migration thresholds, considers fully the gains and costs of migrations, and is applicable to hybrid memories that have multiple NVRAMs. Experimental results with real-life and standardized data show that our method improves over the existing ones the average response time by a factor of up to four, while keeping a similar rate of energy consumption.

References

[1] *SAP HANA In-Memory Database*, SAP. Accessed Jan. 6, 2014. http://www.sap.com/pc/tech/in-memory-computing-hana.html
[2] *Oracle TimesTen In-Memory Database*, Oracle. Accessed Jan. 6,

2014. http://www.oracle.com/us/products/database/timesten/

[3] *UNICOM SolidDB In-Memory Database*, UNICOM Systems. Accessed Sept. 12, 2014. http://unicomsi.com/soliddb

[4] L.A. Barroso and U. Hoelzle, "*The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*," San Rafael, CA, USA: Morgan and Claypool, 2009.

[5] C. Lefurgy et al., "Energy Management for Commercial Servers," *IEEE Comput*., vol. 36, no. 12, Dec. 2003, pp. 39–48.

[6] M.K. Qureshi, V. Srinivasan, and J.A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," *Int. Symp. Comput. Archit*., Austin, TX, USA, June 20–24, 2009, pp. 24–33.

[7] G. Dhiman, R.Z. Ayoub, and T. Rosing, "PDRAM: A Hybrid PRAM and DRAM Main Memory System," *Des. Autom. Conf*., San Francisco, CA, USA, July 26–31, 2009, pp. 664–669.

[8] L.E. Ramos, E. Gorbatov, and R. Bianchini, "Page Placement in Hybrid Memory Systems," *Int. Conf. Supercomput*., Tucson, AZ, USA, May 31–June 4, 2011, pp. 85–95.

[9] D.-J. Shin et al., "Adaptive Page Grouping for Energy Efficiency in Hybrid PRAM-DRAM Main Memory," *Res. Appl. Computat. Symp*., San Antonio, TX, USA, Oct. 23–26, 2012, pp. 395–402.

[10] K.H. Park et al., "Resource Management of Manycores with a Hierarchical and a Hybrid Main Memory for MN-MATE Cloud Node," *IEEE World Congress Services*, Honolulu, HI, USA, June 24–29, 2012, pp. 301–308.

[11] K. Sudan, A. Badam, and D. Nellans, "NAND-Flash: Fast Storage or Slow Memory?" *Non-Volatile Memory Workshop*, San Diego, CA, USA, Mar. 4–6, 2012.

[12] B.C. Lee et al., "Architecting Phase Change Memory as a Scalable DRAM Alternative," *Int. Symp. Comput. Archit*., Austin, TX, USA, June 20–24, 2009, pp. 2–13.

[13] P. Zhou et al., "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," *Int. Symp. Comput. Archit*., Austin, TX, USA, June 20–24, 2009, pp. 14–23.

[14] J.H. Choi et al., "OPAMP: Evaluation Framework for Optimal Page Allocation of Hybrid Main Memory Architecture," *IEEE Int. Conf. Parallel Distrib. Syst*., Singapore, Dec. 17–19, 2012, pp. 620–627.

[15] *Intel*, *STMicroelectronics Deliver Industry's First Phase Change Memory Prototypes*, Intel Corporation, 2008. Accessed Jan. 6, 2014. http://www.intel.com/pressroom/archive/releases/2008/20080206 corp.htm

[16] Y. Choi et al., "A 20 nm 1.8 V 8 Gb PRAM with 40 MB/s Program Bandwidth," *IEEE Int. Solid-State Circuits Conf*., San Francisco, CA, USA, Feb. 19–23, 2012, pp. 46–48.

[17] *Fusion-Io Iodrive2 Duo Data Sheet*, Fusion-io. Accessed Jan. 6, 2014. http://www.fusionio.com/data-sheets/iodrive2-duo/

[18] F. Bedeschi et al., "A Multi-level-Cell Bipolar-Selected Phase-Change Memory," *IEEE Int. Solid-State Circuits Conf*., San Francisco, CA, USA, Feb. 3–7, 2008, pp. 428–625.

[19] S. Chen, P.B. Gibbons, and S. Nath, "Rethinking Database Algorithms for Phase Change Memory," *Biennial Conf. Innovative Data Syst. Res*., Asilomar, CA, USA, Jan. 9–12, 2011, pp. 21–31.

[20] A.J. Hayter, "*Probability and Statistics for Engineers and Scientists*," Belmont, CA, USA: Cengage Learning, 2007, pp. 533–545.

[21] *UMass Trace Repository*, University of Massachusetts Amherst, 2007. Accessed Jan. 6, 2014. http://traces.cs.umass.edu/index.php/Storage/Storage

[22] *TPC-C Benchmark*, Transaction Processing Performance Council, 2014. Accessed Jan. 6, 2014. http://www.tpc.org/tpcc/

[23] *TPC-H Benchmark, Transaction Processing Performance Council, 2014*. Accessed Jan. 6, 2014. http://www.tpc.org/tpch/

[24] R. Cole et al., "The Mixed Workload CH-benCHmark," *Int. Workshop Testing Database Syst*., Athens, Greece, June 13, 2011, pp. 1–6.

[25] A. Badam and V.S. Pai, "SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy," *USENIX Symp. Netw. Syst. Des. Implementation*, Boston, MA, USA, Mar. 30–Apr. 1, 2011, pp. 211–224.

Hai Thanh Mai is a researcher of the Big Data Software Platform Research Department, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. He received his BS degree in computer science from the University of Sciences, Ho Chi Minh, Vietnam, in 2005. He then went on to receive his MS and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea, in 2009 and 2012, respectively. He was a BK21 postdoctoral fellow at KAIST's Department of Computer Science in 2012. His research interests include database systems, big data management, and data mining.

Kyoung Hyun Park is a senior researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. He received his MS degree in computer science from Chungbuk National University, Cheongju, Rep. of Korea, in 2001. He has been involved in many projects related to continuous speech recognition systems and database systems. His current research interests include big data management systems and cloud computing platforms.

Hun Soon Lee received his BS and MS degrees in computer science from Chungnam National University, Daejeon, Rep. of Korea, in 1997 and 1999, respectively. Since 1999, he has been with the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His main research interests include data management systems, data processing systems, and storage systems.

Chang Soo Kim received his MS degree in computer science from Sogang University, Seoul, Rep. of Korea, in 1995 and his PhD degree in information and communication engineering from Chungbuk National University, Cheongju, Rep. of Korea, in 2006. He is a principal researcher working at the Software Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include big data management and processing systems; database systems; cloud computing; and storage systems.

Miyoung Lee is a principal research engineer of the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. She received her MS degree in computer science from Seoul National University, Rep. of Korea, in 1983 and her PhD degree in computer engineering from Chungnam National University, Daejeon, Rep. of Korea, in 2006. From 1988, she has conducted research on database technology and has been involved in many projects related to this and the field of distributed data processing technology. Her current research interests include database systems, stream processing systems, and big data platforms.

Sung Jin Hur is a director of the Data Management Research Section, Electronic and Telecommunications Research Institute, Daejeon, Rep. of Korea, which makes contributions to the nation's economic and social development through research. He has more than fifteen years of experience in real-time data processing and cloud computing systems. He is currently a project manager developing big data SW platforms based on accelerators such as GPGPU, FPGA, and next-generation memory technology to provide the environment for a real-time analysis of big data.