Cluster Comput (2017) 20:1155–1166
DOI 10.1007/s10586-017-0789-4

High-performance data mining with intelligent SSD

Yong-Yeon Jo[1] · Sang-Wook Kim[1] · Sung-Woo Cho[1] · Duck-Ho Bae[1] · Hyunok Oh[2]

Received: 2 November 2016 / Revised: 19 January 2017 / Accepted: 10 February 2017 / Published online: 28 February 2017

Abstract An intuitive way to process the big data efficiently is to reduce the volume of data transferred over the storage interface to a host system. This is the reason that the notion of intelligent SSD (iSSD) was proposed to give processing power to SSD. There is rich literature on iSSD, however, its real implementation has not been provided to the public yet. Most prior work aims to quantify the benefits of iSSD with analytical modeling. In this paper, we first develop on iSSD simulator and present the potential of iSSD in data mining through the iSSD simulator. Our iSSD simulator performs on top of the gem 5 simulator and fully simulates all the processes of data mining algorithms running in iSSD with cycle-level accuracy. Then, we further addresse how to exploit all the computing resources for efficient processing of data mining algorithms. These days, CPU, GPU, and SSD are recently equipped together in most computing environment. If SSD is replaced with iSSD later on, we have a new computing environment where the three computing resources collaborate one another to process big data quite effectively. For this, scheduling is required to decide which computing resource is going to run for which function at which time. In our heterogeneous scheduling, types of computing resources, memory sizes in computing resources, and inter-processor communication times including IO time in SSD are considered. Our scheduling results show that processing in the collaborative environment outperforms that in the traditional one by up to about 10 times.

Keywords Intelligent SSD · Simulator-based evaluation · Collaborative processing · Heterogeneous scheduling

✉ Sang-Wook Kim
wook@hanyang.ac.kr

Yong-Yeon Jo
jyy0430@hanyang.ac.kr

Sung-Woo Cho
nera@hanyang.ac.kr

Duck-Ho Bae
dhbae@agape.hanyang.ac.kr

Hyunok Oh
hoh@hanyang.ac.kr

[1] Department of Computer and Software, Hanyang University, Seoul, Korea

[2] Department of Information Systems, Hanyang University, Seoul, Korea

1 Introduction

The era of big data has already begun. Increasingly more applications such as Internet business, on-line shopping, e-mails, and social networking services handle big datasets. Researchers have long recognized that a good way to process the big data more efficiently is to reduce the volume of data transferred over storage to a host system by taking computations in storage rather than in host CPUs. This is the reason that the notion of intelligent SSD (iSSD) was proposed to give computing power to SSD [1–4]. The iSSD is equipped with a core in each channel, thereby making it possible to process data inside iSSD without transferring data to host.

Most prior works verified the efficiency of iSSD on the basis of the analytical models [1,2]. We feel that there is a strong need in the academic community for a virtual computing environment that is able to process user applications. This paper presents the potential of iSSD in data mining algorithms through the iSSD simulator we develop. Our simulator performs on the gem 5 simulator [5] and fully simulates all the processes of data mining algorithms running in iSSD with cycle-level accuracy.

These days, CPU, GPU, and SSD are equipped together in most computing environment. If SSD is replaced with iSSD later on, we face a new effective collaborative computing environment. Researchers already have tried to deal with big data efficiently by exploiting heterogeneous computing resources such as graphic processing units (GPUs), digital signal processors (DSPs), and field-programmable gate arrays (FPGAs) to move over traditional host CPU. Typical example is OpenCL [6]. However, iSSD was not a member of such computing resources yet. In this paper, we consider the collaboration of computing resources, especially iSSD, graphic processing unit (GPU), and CPU.

GPU is a computing resource to process graphics-related applications [7]. There are some methods proposed recently to exploit GPU for processing general applications rather than graphics-related ones. GPU is composed of a larger number of cores organized in multiple sets, each of which processes the same group of instructions in parallel with cores in it [8]. Some existing studies have addressed applications by using a single computing resource [1,9] while some other methods exploit collaborations of two different resources (e.g., CPU & GPU or CPU & iSSD) for more efficient processing [4,10]. We should address how to exploit all three computing resources for efficient processing of data mining algorithms since computing resources have their own characteristics.

The collaborative environment is a heterogeneous one because it has different types of computing resources that provide different performances and also have their own memory independently, thereby requiring inter-processor communication (IPC) time [11]. In this environment, scheduling is required to decide which core is going to run for which function at which time, aiming at processing target algorithms in an optimal way [4]. We need heterogeneous scheduling [11,12] for our collaborative environment because it has computing resources of different types. A schedule can be obtained from scheduling on a target algorithm normally modeled as a graph [13,14]. This modeling needs to consider the types of computing resources, memory sizes in computing resources, and IPC times between computing resources. In this paper, we are going to focus on how to model a given algorithm as a graph for effective collaborations of computing resources.

Our graph modeling for an algorithm in our collaborative environment is performed as follows: First, we classify functions into two categories, parallelizable functions and merging functions. GPU and iSSD have architecture appropriate for parallel processing of data, and thus are beneficial when assigned to parallelizable functions. On the other hand, CPU is beneficial when assigned to such functions that require merging or have high time complexity. Second, we determine the amount of data to be processed at a time by a function. For example, GPU processes a large amount of data simultaneously by using a large number of cores in parallel. It is more effective to set the amount of data to be matched with the number of cores in GPU. This paper proposes an effective way to process data mining algorithms by using the three kinds of computing resources based on the strategies above.

The main contributions of this paper are summarized as follows.

- We developed an iSSD simulator based on the gem 5 simulator and verified the potential of iSSD in efficient processing of data mining algorithms via the iSSD simulator.
- Assuming this collaborative environment feasible in the (very) near future, we proposed a new paradigm of the data mining algorithms based on collaborations of heterogeneous computing resources of CPU, GPU, and iSSD.
- We designed heterogeneous scheduling to make the best use of heterogeneous computing resources of CPU, GPU, and iSSD. For effective processing, our scheduling considers the types of computing resources, the memory sizes in computing resources, and IPC times including IO time in SSD.
- We implemented the data mining algorithms[1] on top of our iSSD simulator.[2] The source codes of the data mining algorithms and the iSSD simulator are all accessible from the github server.
- We performed extensive experiments to compare the performances of algorithms in all possible combinations of collaborative processing resources. We not only showed the results obtained from different scheduling strategies as well as different numbers of computing resources but also evaluated resource usages of our heterogeneous scheduling. All the results reveal that processing in collaborative environment outperforms that in the traditional one by up to 10 times.

The organization of the paper is as follows. Section 2 introduces the characteristics of computing resources and the synchronous dataflow. Section 3 briefly introduces the modeling of data mining algorithms used in our paper. Section 4 explains our iSSD simulator and evaluation results for the potential of iSSD on our iSSD simulator. Section 5 presents the proposed collaborative environment, and examines our heterogeneous scheduling strategies for collaborative processing. Section 6 shows the effectiveness of our collaborative processing via a variety of experiments. Finally, Sect. 7 summarizes and concludes the paper.

[1] https://github.com/Yongyeon-Jo/Datamining-algorithms-for-S4Sim.

[2] https://github.com/hyunokoh/s4sim.

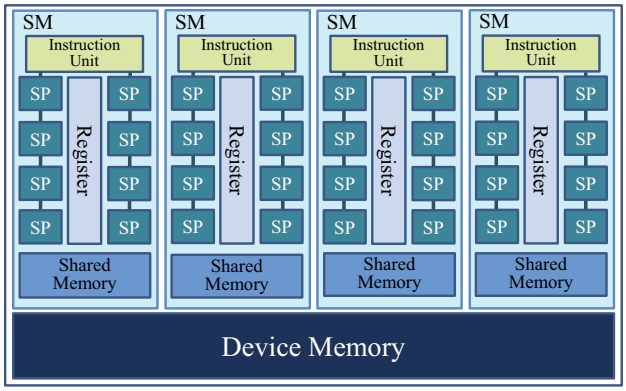


Fig. 1 Architecture of GPU

2 Related work

2.1 Graphic processing unit (GPU)

Figure 1 shows the architecture of GPU. GPU consists of a set of streaming multiprocessors (SM), each of which is again composed of a set of streaming processors (SP). There are memories with different sizes and latency on GPU such as device memory, shared memory, texture memory, constant memory, and register [15]. GPU follows the single instruction multiple threads (SIMT) model on which SPs in an SM execute the same instruction at one time [8]. GPU is suitable for processing algorithms that execute simple operations (e.g., addition and multiplication) iteratively [15]. Many previous researches have been conducted on the data mining algorithms by utilizing the parallelism of GPU based on such characteristics. For example, there are representative techniques such as clustering [16], frequent pattern mining [17,18], and classifications [19].

2.2 Intelligent SSD (iSSD)

Figure 2 shows the internal structure of iSSD [1]. iSSD is SSD that has a flash memory controller (FMC) equipped with general-purpose cores (called FMC cores) and DRAM for data processing in each channel and employs more powerful SSD cores in existing SSD. However, the performance of those cores in iSSD is relatively lower than that of the host CPU. It also has SRAM and DRAM to store the data to be read/written from/to the cells as normal SSD. Data are transferred between DRAM in iSSD and the main memory in host through a host interface, which is controlled by the host interface controller [1].

The in-storage processing (ISP) with iSSD has the following advantages over in-host processing (IHP) [1]: (1) the internal bandwidth of SSD is relatively higher than that of host interface; (2) the time of transferring the data to in-SSD cores from flash memory cells is reduced since the data is processed within iSSD by FMC cores or SSD cores instead of host CPU cores; (3) host CPU, which is quite expensive, is in charge of much less processing workloads because it only receives smaller results processed by iSSD. In order to fully utilize the internal I/O bandwidth of the SSD, some studies to adopt the similar concept of iSSD have been done. There are attempts to attach special-purpose hardware for embedding scan operations in DBMSs to each channel in the SSD [2,20], porting some query processing components within real SSD devices [3], and embedding map operations in the MapReduce framework in real SSD devices [21].

3 Modeling data mining algorithms as synchronous dataflow

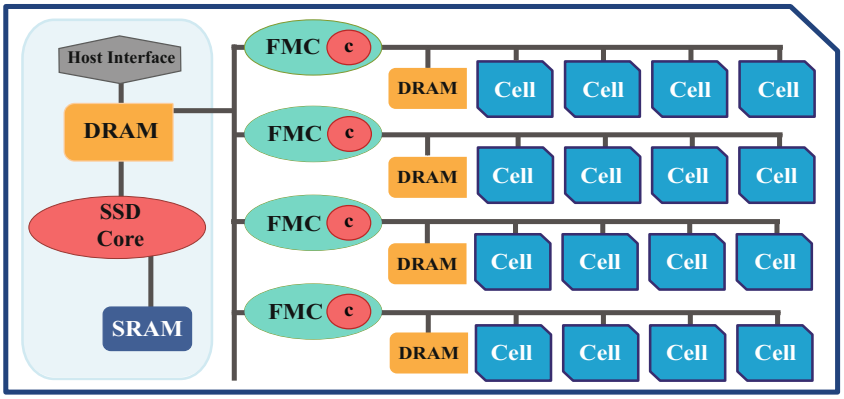
To describe the workflow of data mining algorithms, we use synchronous dataflow (SDF), a well-known graph modeling technique [22].[3] SDF simply represents a graph with relationships between functions in an algorithm. In an SDF graph, functions and calls among them are represented as nodes and edges, respectively. Both ends of an edge have numbers that represent the number of IO tokens (called sample rate) that the two nodes connected to the edge exchange. Figure 3 shows an example of an SDF graph. There are three nodes. Node *A* only generates an IO token during its execution. Node *B* needs two IO tokens (obtained from Node *A*) for its execution and generates an IO token during its execution. Finally, Node *C* only needs an IO token for its execution.

It is difficult to select representative ones among data mining algorithms because they are so diverse. However, most of data mining algorithms have common characteristics of data-intensiveness of requiring repeated accesses to data in storage device. They are the killer applications targeted in our collaborative environment. In this paper, we selected k-means [23], PageRank [24], and SimRank [25] because they are very popular in many application domains and also are data-intensive. SDF graphs for these algorithms are already constructed by our previous work [4] and we adopt them for our evaluation. Now, we present their SDF graphs.

k-means is one of the most-widely used clustering algorithms in data mining area. It performs as follows. First, it reads every object from a storage device (Read) and computes the distances between the object and centroids of current *k* clusters (CalcDist). Based on the distances, it assigns the object into the cluster nearest to the object (SelClust). After assigning all the objects, it re-computes new centroids of the *k* clusters newly made (NewClust). It repeats this process by the pre-defined number of times. When the process is completed, it stores the result into storage device (Write).

[3] The SDF graphs of data mining algorithms are also used in our heterogeneous scheduling scheme in Sect. 5.

Fig. 2 Architecture of iSSD



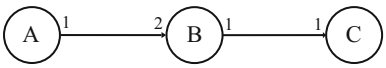


Fig. 3 An example of an SDF graph

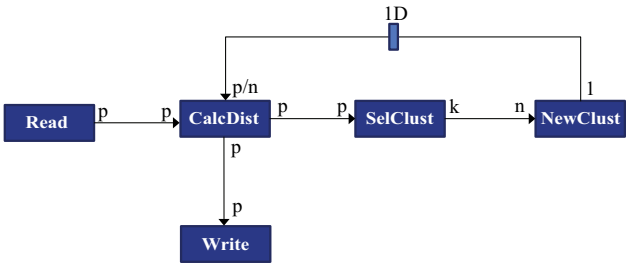


Fig. 4 SDF graph for k-means

Figure 4 is an SDF graph of k-means that follows the above-mentioned steps, where n is the number of objects in data, p indicates the data granularity of each node, k does the number of central clusters, m does the number of iterations for k-means, and $1D$ does delayed buffer which holds the initial centroids of k clusters for CalcDist in the first iteration.

PageRank is a well-known ranking algorithm computed by Eq. 1, where r_i is a ranking vector, W is an adjacency matrix, and a is a damping factor.

$$r_i = (1-a)W * r_{i-1} + ar_0 \tag{1}$$

PageRank performs as follows. An adjacency matrix is read from a storage device (Read). Then, an element value is obtained by multiplying a row of matrix W by Vector r_{i-1} (Multiply). A vector is made by combining all the elements computed by Multiply (MakeVec). Vector r_i is generated by adding this vector and ar_0 in MakeVec (AddDamp). This process is repeated by the pre-defined number of times. When the process is completed, vector r_i is stored to a storage device (Write). Figure 5 shows an SDF graph of PageRank having m iterations, where n represents the number of elements in a ranking vector, $1D$ does delayed buffer that holds the initial vector r_0 for Multiply.

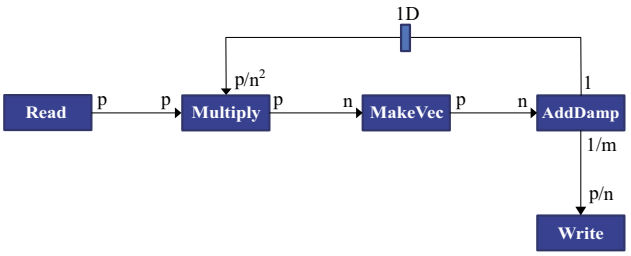


Fig. 5 SDF graph for PageRank

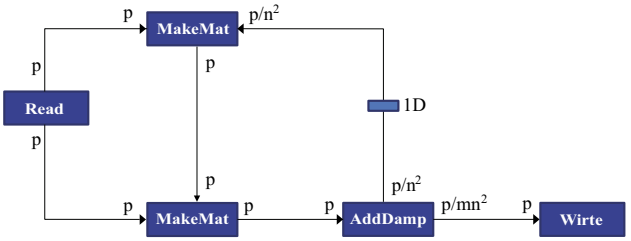


Fig. 6 SDF graph for SimRank

SimRank is an algorithm to compute link-based similarities by employing matrix multiplications as in Equation 2 where S represents a similarity matrix, W does an adjacency matrix, and c does a damping factor.

$$S_k = cW^t S_{k-1} W + (1-c)S_0 \tag{2}$$

SimRank performs as follows. An adjacency matrix is read from a storage device (Read). Matrix W^t is multiplied with S_{k-1}, and then, matrix W is multiplied with a resulting matrix of multiplying W^t and S_{k-1} (MakeMat). S_k is obtained by multiplying the result of MakeMat and c and adding $(1-c)S_0$ to it (AddDamp). This process is repeated by the pre-defined number of times. Matrix S_k is stored into a storage device (Write). Figure 6 shows an SDF graph of SimRank with m iterations, where n is the number of elements in matrix S_{k-1} and $1D$ is delayed buffer that holds the initial matrix S_0.

4 iSSD simulator-based evaluation

To measure the performance of three data mining algorithms mentioned in Sect. 3, we first develop an iSSD simulator that fully simulates all the process of data mining algorithms running in iSSD with cycle-level accuracy. Then, we implement three data mining algorithms on the top of the iSSD simulator.

4.1 Implementation of iSSD simulator

We implemented the iSSD simulator on top of the gem 5 simulator. The gem 5 simulator [5] provides a virtual computing environment that is able to process applications on someone's target system. It allows varying types and performances of processors, memory, and cache of the target system. In our iSSD simulator, we employ processors, memory, and cache as our SSD cores, DRAM, and SRAM, respectively. The gem 5 simulator also provides a variety of capabilities and components which give it a lot of flexibility. These cover a wide range of speed-accuracy trade-offs [26]. In order to improve the accuracy of the simulator, we set the CPU model as *AtomicSimple* which is a minimal single IPC CPU model, and the system mode as *System-call Emulation mode* which avoids the need to model devices or an operating system by emulating most system-level services [26]. Table 1 presents the parameters for the iSSD simulator.

Due to the limitations in the gem 5 simulator, there are two design considerations. First, it is difficult to simulate a channel in such a way that it behaves identically to the iSSD channel [26]. In iSSD, each channel has an FMC core and DRAM and can process data stored in its own cells independently of those in other channels. FMC cores are thus primarily computing resources to perform ISP [1,4]. However, the gem 5 simulator only enforces their cores to share the memory. Therefore, we decide that FMC cores are substituted with SSD cores and the number and rates of SSD cores are set following those of FMC cores. Second, IO simulation is not allowed in the gem 5 simulator [26]. In order to compute the IO time, we add the following steps: (1) count the number of IOs ($\#IO$) by dividing the total data size by the page size (defined in the operating system); (2) find the IO time (T_IO) for reading/writing a single page from/to SSD by referring to a catalog provided by the SSD manufacturer [27]; (3) compute the IO time for given data by multiplying $\#IO$ and T_IO; (4) divide the total IO time by the number of SSD cores ($\#SSDcores$), which is because SSD cores perform IOs in parallel within iSSD. As a result, the IO time in iSSD is computed as in Eq. 3.

$$IOtime = (\#IO \times T\_IO) \;/\; \#SSDcores \tag{3}$$

An algorithm is processed in the iSSD simulator as follows: (1) The binary code of the algorithm is transferred with the data from the host to iSSD; (2) each SSD core executes the binary code as a single thread; (3) the results obtained from SSD cores are delivered to the host. We measure the total processing time by computing the times spent in these three steps.

Table 1 Parameter settings for the iSSD simulator

Components	Parameters
SSD core	Type: ARM
	Number of cores: 4
	Core Rates: 1,200 MHz
DRAM	Type: ddr3_1600_x64
	Size: 8 GB
SRAM	Type: L3
	Size: 16 MB

4.2 Implementation of data mining algorithms

We implemented data mining algorithms on top of the iSSD simulator. One of the major criteria in the iSSD simulator is the cost of creating a new thread. Thus, we carefully design a function granularity of the data mining algorithms to decide which function executes in parallel with multiple threads or executes with a single thread.

- k-means: SelClust is implemented as a single thread because it is simple and requires computations less than other functions. CalcDist and NewClust can be parallelized to make data processed more efficiently. Thus, they are implemented as multiple threads, each of which is assigned with the same amount of objects to be processed.
- PageRank: MakeVec and AddDamp are implemented as a single thread because it is simple and requires much less computations than Multiply. Because Multiply needs to be parallelized for efficient processing, it is implemented as multiple threads, each of which is assigned with the same amount of nodes.
- SimRank: MakeMat and AddDamp are composed of muliple threads and a single thread, respectively. The reason is same as Multiply and AddDamp, respectively, in our PageRank implementation.

4.3 Evaluation on our iSSD simulator

The iSSD simulator based on the gem 5 simulator requires a huge amount of time to measure the execution time. It normally requires 12–18 h for simulating real executions taking 0.01 s. We only perform two algorithms such as k-means and

Table 2 Execution times with varying the number of SSD cores (seconds)

#SSD cores	k-means	PageRank
4	0.766	0.136
8	0.330	0.071
16	0.159	0.039
32	0.078	0.025
64	0.043	0.017
128	0.026	0.013
256	0.019	0.012

Table 3 Execution times of IHP and ISP on iSSD

Algorithm	Execution time (s)	
	IHP	iSSD
k-means	0.056	0.019
PageRank	0.016	0.012

PageRank except for SimRank which is difficult to acquire the result within a day on even small data.

For evaluating ISP on iSSD in terms of data mining algorithms, we conducted a series of experiments on a Linux server equipped with Intel i5 3.2 GHz, 8 GB main memory, and Intel SSD 530. The parameters for iSSD are shown in Table 1. We used the small dataset in our experiments [28]. For k-means and PageRank, there are about 10,000 objects and 7000 nodes, respectively.

4.3.1 ISP with different iSSD settings

We present the experimental results of evaluating the performance of data mining algorithms performed with ISP in iSSD. We measured their execution times while changing the number of SSD cores within iSSD. Table 2 shows the results: both k-means and PageRank show that the execution times decrease with increasing the number of SSD cores with ISP on SSD.

4.3.2 Comparison of ISP with IHP

We compared the performance of ISP on iSSD with that of IHP. To evaluate the performance of IHP, we measured the time of executing the same data mining algorithms on the host CPU with the same datasets used in ISP. To evaluate the performance of ISP, the number and rate of SSD cores are set as 256 and 1200 MHz, respectively, in our iSSD simulator. Table 3 shows the results of the experiments. ISP on iSSD shows performance improvement over IHP up to about 300 and 130% in the k-means and pageRank algorithms, respectively. In these algorithms, the functions performed in parallel occupy a big portion in its total execution. Thus, iSSD equipped with a more number of SSD cores is likely to show a smaller execution time.

5 Data processing in collaborative environment

In this section, we discuss how to process data mining algorithms in collaborative environment having CPU, GPU, and iSSD to achieve the maximum efficiency.

5.1 Scheduling

Scheduling algorithms are classified into static and dynamic ones. The static scheduling generates a schedule of all cores before the execution of a data mining algorithm while the dynamic scheduling determines a schedule of cores just before every function of a data mining algorithm begins to run. In our environment, we have a large number of cores. When dynamic scheduling algorithms are employed, the total execution time increases drastically due to the scheduling overhead. Thus, we first considered two algorithms that allow heterogeneous static scheduling of a general dynamic level (GDL) and the best imaginary level (BIL) [11] in our paper. After comparing them by experiments [29], we finally chose BIL scheduling because it provides better performance than GDL scheduling.

BIL scheduling produces a schedule by using the acyclic precedence expanded graph (APEG), which contains all the nodes generated according to the sample rate in the SDF graph. Figure 7 shows an example of an APEG generated from the SDF graph in Fig. 3. Node *A* in the SDF graph presents two separate nodes in the APEG because Node *B* needs two IO tokens obtained from Node *A*.

BIL scheduling tries to reduce the execution time of a node that needs more time at first [11]. It greedily chooses a node having the longest execution time in the APEG and assigns a processor that can execute the chosen node faster than any other processors. BIL scheduling needs the following inputs: (1) An APEG of a target algorithm. Nodes and edges in the APEG represent functions and calling relationships, respectively. (2) A time table including per-processor execution time of each function and the IPC time among processors [11].

5.2 Graph modelling strategy

The followings are our two strategies for modeling data mining algorithms as an APEG. First, we need to decide the function granularity in a graph, which is a granular-

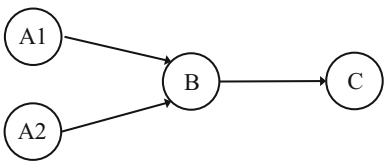


Fig. 7 An example of an APEG

ity of a node corresponding to a function. We can model the whole algorithm as one big node or can model each instruction as each node. To decide function granularity, we should consider the characteristics of computing resources. In our collaborative environment, GPU has a large number of cores in spite of low processing power, thus providing a high degree of parallelism. iSSD also provides a certain degree of parallelism with fast IO speed. Meanwhile, CPU has high processing power, thus beneficial to execute such functions having high lock overhead like merging functions, rather than parallel processing. We thus classify functions into two categories as follows: (1) Functions for merging data from other functions (called merging nodes in the APEG) and (2) functions that are able to process data in parallel (called parallelizable nodes in the APEG).

Second, we need to decide the data granularity, which represents the numbers of IO tokens between two functions. The number of IO tokens represents the number of nodes in the APEG. Thus, a fine data granularity function, which is represented by a large number of nodes in the APEG, can be processed in parallel. GPU is quite effective in processing a large amount of data simultaneously since it employs SIMT architecture, and thus enables to process efficiently a function having a high data granularity. It is not allowed to assign a specific node directly to GPU because GPU has its own scheduler. In order to induce parallelizable nodes assigned to GPU by scheduling, we should make functions having a high data granularity. If a node needs to be assigned to GPU, its number of IO tokens should be determined by Eq. 4. With this number of IO tokens, all the cores in GPU could be fully utilized. Here, the number of IO tokens, number of cores in GPU, memory size in GPU, and average size of IO tokens are denoted as n_{IO_tokens}, n_{cores}, S_{mem}, and S_{IO_tokens}, respectively. In other words, the summation of numbers of all IO tokens should be smaller than or equal to the memory size in GPU. Also, the number of IO tokens should be multiples, m, of the number of cores in GPU.

$$\begin{aligned} n_{IO\_tokens} &= m \times n_{cores} \\ n_{IO\_tokens} &\leqslant S_{mem}/S_{IO\_tokens} \end{aligned} \tag{4}$$

In order to determine the best data and function granularities for finding the most efficient schedule for a given algorithm, we perform the following process. We first model each function by using multiple nodes as fine-grained as possible, and classify each node into one of the two categories above. We then select a node to be assigned to GPU, and then determine its data granularity according to Eq. 4. Now, we have an APEG in the most fine-grained level. We also build a coarse-grained APEG in the similar way, after merging those nodes belonging to the same function. Then, we create different schedules for APEGs thus obtained. We run the algorithm multiple times with those schedules, and select the best one as our final schedule of the algorithm for the future use.

Because BIL scheduling is a static scheduling, the function and data granularity is determined according to the size of input data. The function and data granularity needs to be changed if an input data of a different size arrives. Since different granularity leads to a different schedule, a new schedule is normally required for the new input data. In order to solve this problem, we use the strategy proposed by [4] which recycles a pre-generated schedule by slightly modifying the data granularity of nodes in APEG. We first generate a schedule for the first input data (hereafter, we refer to this as a *base schedule*). When a new input data of a different size arrives, we adjust the data granularity to process the new data. Also, the execution time of nodes and IPC time are recomputed according to the adjusted granularity. For example, if the size of input data is twice of the original one, then we simply increase data granularity in APEG of the base schedule by double. The execution time and IPC time in the time table become doubled as well.

Of course, the modified version from the base schedule is unlikely to be optimal for the new input data. However, we note that scheduling is time-consuming and thus it may need more than (or at least comparable to) that of execution time, in processing a large volume of data. So, it is not recommended to start over re-scheduling for new data to get the best schedule. The scheduling time of the modified version schedule only requires less than 1%, further the error of the performance based on it is just under 10% compared with the base schedule [4].

Some algorithms generate intermediate results of quite different sizes even when the datasets are quite similar in their sizes. In such a case, a static scheduling algorithm like BIL is unable to employ the same schedule under a schedule recycling strategy, and needs to use different schedules generated for processing those datasets. This leads to generation of a new schedule every time a new dataset arrives. These algorithms are not matched with BIL scheduling. Apriori [30] for a frequent pattern mining is a typical example belonging to this category of algorithms.

6 Performance evaluation

6.1 Experimental setup

We classify the nodes in SDF graphs of the three algorithms into merging nodes and parallelizable nodes as in Table 4. The nodes for Read and Write are not included because they are the functions related to data IO and thus need to be definitely executed in FMC cores.

For our performance evaluation, we employ Vtune [31] to measure the time for executing a node and the time for

Table 4 Node categories

Algorithm	Node	
	Merging	Parallelizable
k-means	NewClust	CalcDist, SelClust
PageRank	MakeVec, AddDamp	Multiply
SimRank	AddDamp	MakeMat

Table 5 Hardware related parameters

	Parameter	Value
Host	# of host CPU cores	4
	Host CPU rate (GHz)	3
	Interface bandwidth (Gbps)	3
iSSD	# of SSD cores	4
	SSD core rate (Mhz)	800
	# of FMC cores	32
	FMC core rate (MHz)	400
	Cell read time (μs)	50
	Cell Write time (μs)	1200
GPU	# of GPU cores	3000
	GPU core rate (MHz)	900

IPC, both of which are needed as inputs in scheduling. The parameters related to hardware for simulation are given in Table 5. We synthetically generate 100 GB data in a random way for our experiments. The dataset for k-means consists of a list of objects, each of which has eight dimensions. A value of each dimension ranges from 1 to 32,000. The datasets for PageRank and SimRank consist of a list of edges connecting source and destination nodes. The number of nodes is about 800,000. A value of each dimension in an object and the index of a node are randomly selected. We assume that the data are uniformly stored over all the cells in iSSD, thereby making all FMC cores process similar IO workloads.

In order to gauge the effect of our collaborative environments, we repeatedly ran the BIL scheduling with the best performance under different computing environments. We call a environment having CPU only and iSSD only to IHP and ISP, respectively. Also, we denote the computing environment embedding CPU and GPU as CPU+GPU, that with CPU and iSSD as CPU+iSSD, and that with CPU, GPU, and iSSD all together as CPU+GPU+iSSD. We note that BIL scheduling is not applicable to IHP because it is a heterogeneous scheduling algorithm. The execution times of IHP in our experiments are obtained from a real machine.

6.2 Determining granularities

In this section, we perform experiments to find the function and data granularities appropriate for the best schedule. First, we determine the APEG in a most fine-grained level.

Table 6 Execution times with different function granularities (s)

Algorithm	Graph	
	Fine-grained	Coarse-grained
k-means	**402.4**	536.9
PageRank	2791.5	**1902.9**
SimRank	**3791.5**	4239.9

The bold values indicate the best case in each algorithm

For this purpose, we use the graphs introduced in Sect. 3. Then, for the parallelizable nodes, we determine the data granularity according to Eq. 4. Next, we build an APEG in a coarse-grained level. In k-means, we merge CalcDist and SelClust into SetClust, a coarse-grained node. In PageRank, we build an SDF graph by combining MakeVec and AddDamp (i.e., those requiring merging), to ResultVec, a coarse-grained node. In SimRank, to build an SDF graph in a coarse-grained level, we merge MakeMat (parallelizable node) and AddDamp (merging node) into ResultMat, a coarse-grained node even though they belong to different categories.

Table 6 shows the execution times of three algorithms with the schedules obtained from APEG in different granularity levels. Here, we set m in Eq. 4 as one for the data granularity. In k-means, we try to reduce the IPC time by combining two parallelizable nodes into a single node. However, the APEG in a fine-grained level shows better performance than that in a coarse-grained level. We decide the data granularity of CalcDist as large as possible in such a way that it is processed in GPU. SetClust, which contains CalcDist and SelClust in a single node, has a large data granularity in order that it is processed in GPU as well. We note, however, SelClust (parallelizable node) can be processed in iSSD or CPU without being transferred to GPU. This strategy of employing coarser SetClust adversely affects the performance. In PageRank, the performance gets better by combining MakeVec and AddDamp into ResultVec, which reduces the IPC time in its execution. In SimRank, it is more effective to use separate MakeMat and AddDamp because they belong to different categories. In the following experiments, we thus use the APEG in a fine-grained level for k-means and SimRank, but the APEG in a coarse-grained level for PageRank.

Table 7 shows the execution times of the three algorithms according to the schedules obtained with different data granularities. The result shows that the execution time is reduced in the cases when m is set as the multiples of cores in GPU. In other cases, the execution time becomes longer because the cores in GPU are not fully utilized. In the following experiments, we use the data granularity with m set as 1 in Eq. 4.

Table 7 Execution times with different data granularities (s)

Algorithm	m			
	0.5	1	1.5	2
k-means	454.7	**402.4**	421.3	402.4
PageRank	2975.8	**1902.9**	2290.6	1944.8
SimRank	7044.3	**3791.5**	5687.5	3802.9

The bold values indicate the best case in each algorithm

6.3 Performance comparisons

In this section, we perform experiments to compare the performances of algorithms in five different computing environments. Table 8 shows the execution times and rankings (in parentheses) for each environment. The result indicates that the execution time in a collaborative environment is much smaller than that in a non-collaborative environment. Also, the performance gap increases as the time complexity and parallelizability of nodes get higher. The reason for this is that the parallelizable nodes can be processed in iSSD or GPU in parallel, thereby reducing the execution time. Thus, CPU+GPU and CPU+iSSD that use CPU and iSSD, respectively, and ISP perform better than IHP. CPU+GPU+iSSD that use GPU and iSSD together show the best performance.

6.4 Resource utilization

We examine computing resources in CPU+GPU+iSSD are well collaborated with one another when performing algorithms. Figure 8 shows the utilization of resources for each function. The x-axis represents nodes in algorithms and the y-axis does the utilization of resources. The result shows that collaborations among resources are effectively done in processing all the algorithms. In k-means, NewDist (merging node) and SelClust (parallelizable node) are primarily processed in CPU. On the other hand, CalcDist (parallelizable node), is assigned mainly in GPU by the BIL scheduler. Thus, iSSD seems to play a role of supplementing other computing resources. In PageRank, ResultVec (merging node) is processed in CPU, *Multiply* (parallelizable node) is mostly processed in GPU and iSSD. In SimRank, however, AddDamp is processed in all computing resources except for GPU even though it requires merging. We conjecture it exploits all the resources due to the high time complexity of SimRank.

6.5 Changing parameters

We measured the execution times with changing the parameter values related to hardware components within iSSD and GPU. In Fig. 9, the left side shows the results with changing the number of FMC cores; the center does the results with changing the number of SSD cores; the right side does results with changing the number of GPU cores. The x-axis indicates the number of cores and the y-axis does the execution time. Each line denotes the execution time on a computing environment.

As a result, the performance of ISP improves as the number of FMCs increases. Likewise, the performance of our collaborative environment can be improved as well if the number of FMCs increases. The performance improvement by the number of SSD cores is lower than that by FMC cores because FMC cores primarily process the algorithms in iSSD. The execution time of environments equipped with GPU also decreases while the number of GPU cores increases. The results reveal that CPU+GPU+iSSD outperforms IHP up to about 10 times.

7 Conclusions

In this paper, we developed an iSSD simulator based on the gem 5 simulator and then evaluated how much data mining algorithms perform better inside iSSD on top of the simulator with the cycle-level accuracy. Via this simulator-based evaluation, we successfully verified the potential of iSSD.

Also, we claimed that a near future computing environment may be embedded with different computing resources of CPU, GPU, and iSSD, and addressed how to effectively perform data mining algorithms in a collaborative way in this environment. We observed that the three computing resources have different characteristics, and thus should have different roles for efficient processing of data mining algorithms. CPU is the most powerful among them and is suitable for processing the functions with high computational com-

Table 8 Execution time for each environment (seconds)

Algorithm	Environment				
	CPU+GPU+iSSD	CPU+iSSD	CPU+GPU	ISP	IHP
k-means	**402.46 (1)**	650.77 (3)	419.92 (2)	977.16 (4)	1264.71 (5)
PageRank	**1902.93 (1)**	2482.63 (3)	2328.28 (2)	3807.54 (4)	3796.69 (5)
SimRank	**3791.59 (1)**	5884.29 (2)	5980.02 (3)	10488.23 (4)	13612.28 (5)

The bold values indicate the best case in each algorithm

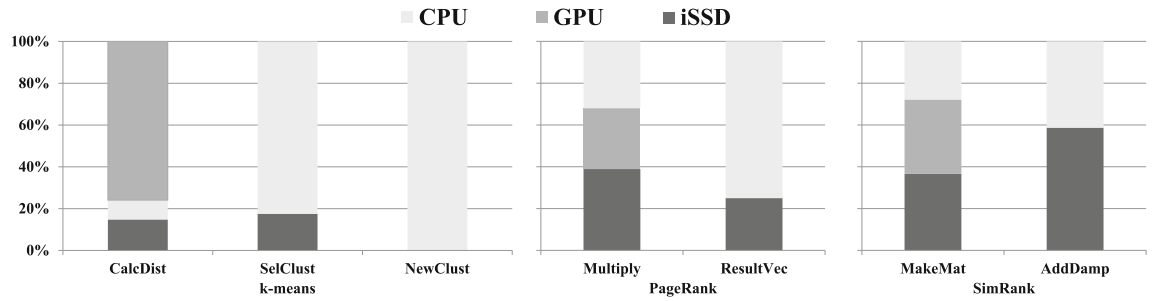


Fig. 8 Utilization of resources

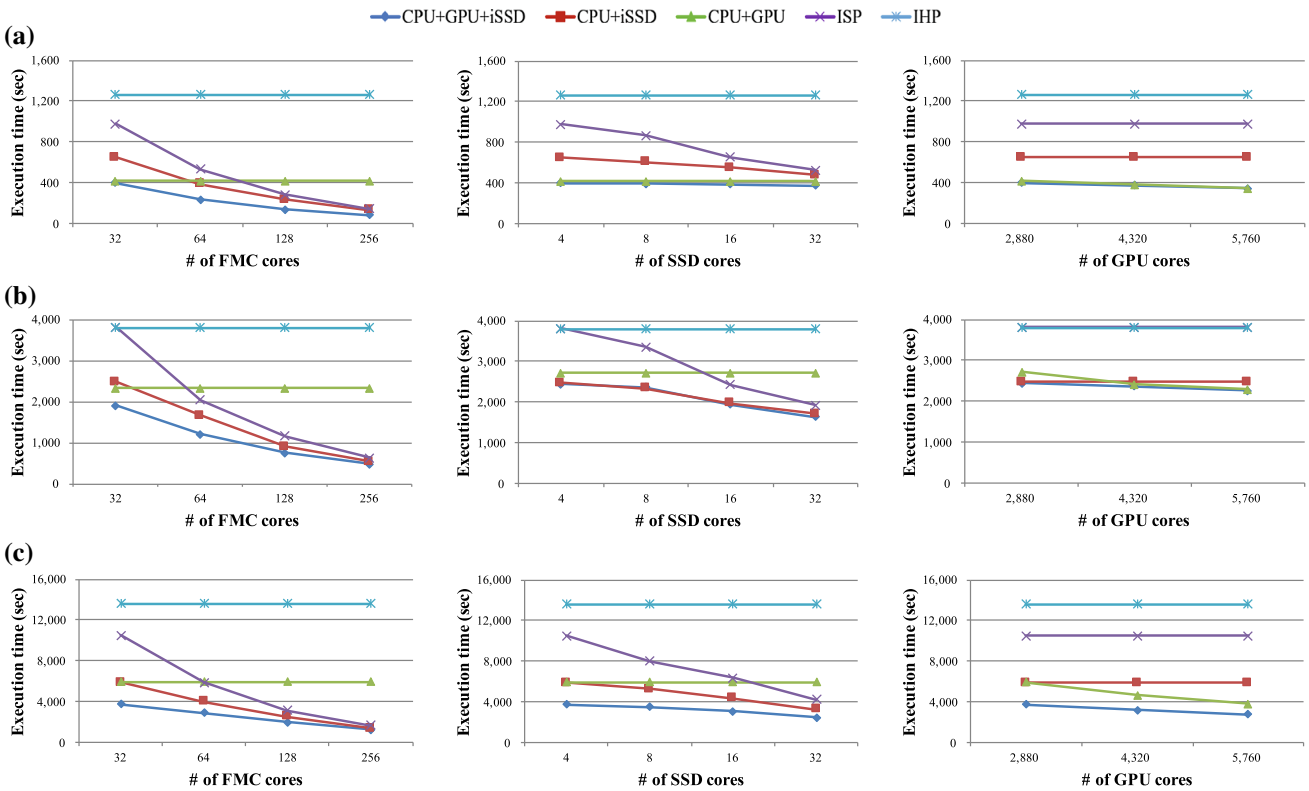


Fig. 9 Execution time with changing the number of cores on resources

plexity. In the case of GPU, its SIMT architecture enables simultaneous processing of the same job for a large amount of data. iSSD, a future SSD having cores within it, is easier to access data than the other two and is suitable for processing the functions with low computational complexity in parallel.

In order to process efficiently data mining algorithms in a collaborative way, it is important to orchestrate the three computing resources effectively. Towards this end, we need to determine how to assign the functions of a data mining algorithm to appropriate computing resources in such a way to make the three resources utilized as much as possible. Thus, we designed heterogeneous scheduling to utilize our collaborative environment for running data mining algorithms effectively. We addressed the issues of determining data and function granularities that affect the performance in collaborative environments. We proposed methods of how to set the two granularities to get better performance.

Finally, we showed the superiority of data processing in a collaborative environment by a series of experiments. The results revealed that data processing in our collaborative environment outperforms that in non-collaborative ones by up to 10 times.

This paper discusses how to effectively perform data mining algorithms using the collaborative environment on a *single* machine having a *single* CPU, a *single* GPU, and a *single* iSSD. As a further study, we plan to study how to process data mining algorithms in the environment having *multiple* CPUs, *multiple* GPUs and *multiple* iSSDs on a *single* machine, and also how to perform data mining algorithms in distributed environment composed of a number of nodes,

each of which is equipped the three resources. This could be a good research direction for high performance data mining in future computing environment.

Acknowledgements Moonjun Chung helped the implementation of iSSD simulator based on the gem 5 simulator for our experiments. This work was supported by (1) a Semiconductor Industry Collaborative Project between Hanyang University and Samsung Electronics Co. Ltd. and (2) the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2014R1A2A1A10054151).

References

1. Bae, D., Kim, J., Jo, Y., Kim, S., Oh, H., Park, C.: Intelligent SSD: a turbo for big data mining. Compu. Sci. Inf. Syst. **13**(2), 375–394 (2016)
2. Kim, S., Oh, H., Park, C., Cho, S., Lee, S.: Fast, energy efficient scan inside flash memory SSDs. In: Proceedings of International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (2011)
3. Do, J., Kee, Y., Patel, J., Park, C., Park, K., DeWitt, D.: Query processing on smart SSDs: opportunities and challenges. In: Proceedings of the ACM International Conference on Management of Data, pp. 1221–1230 (2013)
4. Jo, Y., Cho, S., Kim, S., Bae, D., Oh, H.: On running data-intensive algorithms with intelligent SSD and host CPU: a collaborative approach. In: Proceedings of the International Conference on ACM/SIGAPP Symposium on Applied Computing, pp. 2060–2065 (2015)
5. Website for the gem 5 simulator: http://www.gem5.org/main (2015)
6. OpenCL: https://www.khronos.org/opencl/
7. Fung, J., Mann, S.: Using graphics devices in reverse: GPU-based image processing and computer vision. In: Proceedings of the 2008 IEEE International Conference on Multimedia and Expo, pp. 9–12 (2008)
8. Ryoo, S., Rodrigues, C., Baghsorkhi, S., Stone, S., Kirk, D., Hwu, W.: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 73–82 (2008)
9. Govindaraju, N., Gray, J., Kumar, R., Manocha, D.: GPUTeraSort: high performance graphics coprocessor sorting for large database management. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 325–336 (2006)
10. Pabst, S., Koch, A., Straber, W.: Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. Comput. Graph. Forum **29**(5), 1605–1612 (2010)
11. Oh, H., Ha, S.: A static scheduling heuristic for heterogeneous processors. In: Proceedings of International Conference on Europar Parallel Processing, pp. 573–577 (1996)
12. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **3**(3), 260–274 (2002)
13. Sih, G., Lee, E.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. **4**(2), 175–187 (1993)
14. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
15. Bell, N., Garland, M.: Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report (2008)
16. Farivar, R., Rebolledo, D., Chan, E., Campbell, R.: A parallel implementation of K-means clustering on GPUs. PDPTA **13**(2), 212–312 (2008)
17. Adil, S., Qamar, S.: Implementation of association rule mining using CUDA. In: Proceedings of International Conference on Emerging Technologies, pp. 332–336 (2009)
18. Zhou, J., Yu, K., Wu, B.-C.: Parallel frequent patterns mining algorithm on GPU. In: Proceedings of International Conference on Systems Man and Cybernetics, pp. 435–440 (2010)
19. Catanzaro, B., Sundaram, N., Keutzer, K.: Fast support vector machine training and classification on graphics processors. In: Proceedings of the 25th International Conference on Machine Learning, pp. 104–111 (2008)
20. Cho, S., Park, C., Oh, H., Kim, S., Y,i Y., Ganger, G.: Active disk meets flash: a case for intelligent SSDs. In: Proceedings of the 27th International ACM Conference on Supercomputing, pp. 91–102 (2013)
21. Kang, Y., Kee, Y., Miller, E., Park, C.: Enabling cost-effective data processing with smart SSD. In: Proceedings of the 29th IEEE Symposium on Massive Storage Systems and Technologies, pp. 1–12 (2013)
22. Lee, E., Messerschmitt, D.: Synchronous data flow. Proc. IEEE **75**(9), 1235–1245 (1987)
23. MacQueen, J., et al.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1(14), pp. 281–297 (1967)
24. Page L. et al.: The PageRank citation ranking: bringing order to the web. Technical report, Stanford University (1999)
25. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, pp. 538–543 (2002)
26. Binkert, N., et al.: The Gem5 simulator. ACM SIGARCH Comput. Archit. News **39**(2), 1–7 (2011)
27. IO performance of SSD: http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-530-sata-specification.pdf
28. Jo, Y., Kim, S., Chung, M., Oh, H.: Data mining in intelligent SSD: simulator-based evaluation. In: Proceedings of Big Data and Smart Computing, pp. 123–128 (2016)
29. Jin, D., Cho, S., Jo, Y., Kim, S.: Performance analysis of collaborative processing by scheduling algorithm. In: Proceeding of The 2014 Fall Conference of the KIPS, pp. 105–107 (2014)
30. Agraval, R., Srikant, R.: Fast algorithms for mining association rules in large data bases. In: Proceedings of 20th International Conference on Very Large Data Bases. pp. 487–499 (1994)
31. Intel VTune Amplifier: https://software.intel.com/en-us/node/529213 (2014)

Yong-Yeon Jo received the M.S. degree in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2013. He is currently a Ph.D. candidate in Department of Computer and Software, Hanyang University. His research interests include graph processing, data mining, social network analysis, and high-performance computing with SSD and GPGPU.

Sang-Wook Kim received the B.S degree in Computer Engineering from Seoul National University, Seoul, Korea in 1989 and earned the M.S. and Ph.D. degrees from Korea Advanced Science and Technology (KAIST) at 1991 and 1994, respectively. He is Professor at Department of Computer Science and Engineering, Hanyang University. Professor Kim worked with Carnegie Mellon University and IBM Watson Research Center as a visiting scholar. He is an associate editor of Information Sciences. His research interests include data mining and databases.

Sung-Woo Cho received the M.S. degree in Computer and Software from Hanyang University, Seoul, Korea, in 2014. He is currently a researcher at WIPS, an intellectual property service corporation. His research interests include data mining, social network analysis, and high-performance computing with SSD and GPGPU.

Duck-Ho Bae received the B.S., M.S., and Ph.D. degrees in Electronics and Computer Engineering from Hanyang University, Seoul, Korea at 2006, 2008, and 2013, respectively. Currently, he is a senior engineer at Samsung Electronics. His research interests include flash-based DBMSs, embedded DBMSs, data mining, and social network analysis.

Hyunok Oh received the M.S. and B.S. degree in Computer Engineering from Seoul National University, Korea, in 1998 and 1996, respectively. He received the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Korea, 2003. He is currently an associate professor in Hanyang University. His research interests include security, non-volatile memory, SSD, dataflow model, and real-time analysis.