Computer Science and Information Systems 13(2):375–394 DOI: 10.2298/CSIS150820008D

Intelligent SSD: A turbo for big data mining

Duck-Ho Bae[1], Jin-Hyung Kim[1], Yong-Yeon Jo[1], Sang-Wook Kim[2*], Hyunok Oh[3], and Chanik Park[4]

[1] Dept. of Electronics and Computer Engineering, Hanyang University,
222, Wangsimni-ro, Seongdong-gu, Seoul, Republic of Korea
{dhbae,kjhfreedom,jyy0430}@agape.hanyang.ac.kr
[2] Dept. of Computer and Software, Hanyang University,
222, Wangsimni-ro, Seongdong-gu, Seoul, Republic of Korea
wook@hanyang.ac.kr
∗Corresponding author
[3] Dept. of Information Systems, Hanyang University
222, Wangsimni-ro, Seongdong-gu, Seoul, Republic of Korea
hoh@hanyang.ac.kr
[4] S/W Development Team, Samsung Electronics Co., Ltd.,
129, Samsung-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do, Republic of Korea
ci.park@samsung.com

Abstract. This paper introduces a new notion of the intelligent SSD and presents its potential benefits in terms of data mining applications. With intelligent SSDs, a large volume of data can be directly processed by CPU and DRAM inside intelligent SSDs, and the final result of a very small size needs to be transferred to the host CPU instead of all the data stored in intelligent SSDs. We first discuss design considerations of intelligent SSDs compared with the current SSD architecture. We then analyze the execution costs of data mining applications running on intelligent SSDs by formulating their cost models. Finally, we show the efficiency of performing data mining on intelligent SSDs by comparing it with those of traditional ones through a series of simulations. Through the experimental results, we show that the intelligent SSDs provide significant improvement in performance over the host CPUs in processing data-intensive data mining applications.

Keywords: Intelligent SSD, big data mining, in-storage processing, active SSD.

1. Introduction

Recently, due to the growing of Internet-related business, the avalanche of e-mails, the spread of smart devices, and the emergence of various social network services, the amount of data generated and accumulated is growing at a significantly increasing rate. Owing to the high I/O bandwidth and low access latency, *solid state drives (SSDs)* have been rapidly replacing hard disk drives (HDDs) for data storage [1,2]. SSD manufacturers such as Micron, Samsung, SanDisk, and Toshiba try to overcome the physical limitations of flash memory and to provide higher I/O bandwidth for faster data accesses.

However, such efforts made us face a new performance bottleneck in data-intensive applications [3]. In HDD environment, the *I/O bandwidth of HDDs is a main performance bottleneck in data-intensive applications* because HDDs read data by rotating a magnetic disk physically. However, SSDs employ no mechanical devices and provide much higher

internal I/O bandwidth thanks to multi-way and multi-channel interleaving [4]. Therefore, in SSD environment, overall I/O performance is bound to the *I/O bandwidth of a host interface*, *which subsequently becomes a new performance bottleneck* in data-intensive applications such as big data analysis [3]. As a result, although the internal I/O bandwidth in SSDs can easily scale up by increasing the numbers of ways and channels for more effective multi-way and multi-channel interleaving, enlarging the internal bandwidth of SSDs as such is currently meaningless due to the problem of the host interface bottleneck.

In this paper, as a fundamental solution to the problem of the I/O performance bottleneck in the host interface, we propose an approach called *intelligent SSD* (iSSD) in which a large volume of data are directly processed by computing resources (cores) inside the iSSD and then only the final result of a very small size is transferred from the iSSD to the host instead of all the entire data stored in the iSSD. With this approach, the host interface becomes free from the I/O performance bottleneck when processing data-intensive applications even with the enlarged internal I/O bandwidth of the SSD. As a result, the iSSD enables the significant performance improvement in processing data-intensive applications. Moreover, since the data is processed by the cores inside the iSSD that are so close to the location where it is stored, we can save both the resources and energy for data transfer between the host and the storage [4].

There have been several researches on processing data within storage devices, mainly focusing on HDDs [5,6]. Since HDDs have a limitation to the expansion of the internal I/O bandwidth due to their physical characteristics (i.e., very long times for seek and rotational latency), their approaches have not been practically adopted [4]. However, the SSD employs no mechanical mechanism and also provides high scalability of internal I/O bandwidth. Moreover, *by adding a general-purpose core and memory to each channel* inside the SSD that has a multi-channel structure, we can process the data *in parallel*. This enables the iSSD to have inside computing power large enough to process data-intensive applications efficiently.

In this paper, we focus on *data mining* as our primary applications of the iSSD. Data mining is a typical example of data-intensive applications and has the following characteristics: (1) It frequently accesses the stored data during its execution; (2) It conducts relatively simple operations repeatedly applied to the accessed data; and (3) the scalability of data parallelism in data mining fairly is high[5] [7,8]. Thus, the iSSD could be a good solution to the problem of dealing with such big data mining applications.

In this paper, we first discuss design considerations of the iSSD compared to a current SSD architecture for processing data within the iSSD efficiently (in Section 3). With the iSSD, the bandwidth of the host interface is no longer the performance bottleneck of data-intensive applications. Therefore, we can provide the iSSD internal I/O bandwidth significantly higher than that of a current SSD by increasing the numbers of channels and ways inside the iSSD. Moreover, in order to fully exploit the internal I/O bandwidth and to avoid becoming a new performance bottleneck in in-storage processing, the iSSD should have sufficient inside processing power by adding a general-purpose processor to each channel.

We then propose strategies for executing data mining efficiently inside the iSSD (in Section 4.1). Our strategies are (1) to process all the data in parallel over all the channels

[5] Data parallelism is achieved when each processor performs the same task on different datasets [10].

and (2) to use the flash memory cell as virtual memory for channel memory when an application needs more memory during its execution. This allows us to make up for the weakness of the iSSD (i.e, low computation power compared with the host side) and to reinforce the strength of the iSSD (i.e., high I/O internal bandwidth owing to the multi-channel structure).

We also formulate cost models of data mining applications (in Section 4.2) and then validate them (in Section 5.1). We show our cost models are quite accurate in terms of the correlation between the execution time in the real machine and the estimated execution time obtained from our cost models. We finally verify the potential of the iSSD by comparing its performance with those of traditional ones through a series of simulations (in Sections 5.2 and 5.3). The results show that the iSSD achieves significant speed up by up to 83 times compared with the host CPU in data mining applications.

This paper is an extension of its preliminary version that has been presented as a short paper (4 pages) in ACM CIKM 2013 [9][6]. New materials in this extended version are as follows. We present basic strategies to process the data efficiently in intelligent SSD (Section 4); in the experimental section, we verify the validity of our cost model for processing of data mining applications in intelligent SSD (Section 5.1); we show the performance of data mining applications according to different sizes of channel memory in intelligent SSD (Section 5.3).

The rest of this paper is organized as follows. Section 2 reviews related work for intelligent SSD. Section 3 reviews the architecture of current SSD and introduces the notion of intelligent SSD. Section 4 proposes our approach to data mining in intelligent SSD. Section 5 demonstrates and analyzes the results of experimental evaluation. Finally, Section 6 summarizes and concludes the paper.

2. Related Work

The attempts to process data inside storage devices have continued from the end of 1970. A database machine, appeared in 1970s and 1980s, is special-purpose hardware for simple but core database operations such as sort and join [8,10]. However, database machines have low price-efficiency due to the high cost of extra hardware and failed to be commercially successful [11].

In late 1990s, as the prices of processor and memory decrease, the low-cost general-purpose CPU and DRAM are equipped inside HDDs. Some researchers explored the concept of active (or intelligent) disks that execute database operators such as scan, selection, sorting, and join using CPU and DRAM inside HDDs [12,13,14]. Reference [12] focused on scan-based algorithms for nearest neighbor searches, frequent sets, finding and image edge detection. Reference [13] also focused on similar applications but assumed more

[6] This manuscript is marginally related to reference [15]. This is a paper written in Korean and has been presented in a Korean conference targeted only for Korean audiences. It only contains a very small portion of our paper, addressing the k-means algorithm in the iSSD. Our paper contains the following more contents, which are not contained in reference [15]: (1) Building and verifying our cost model for data processing and data accessing, (2) comparisons of in-storage-processing on the iSSD with in-host-processing in performing additional data mining algorithms such as PageRank and Apriori, (3) Sensitivity analysis on parameters such as the size of channel memory, the speed of channels, and the number of channels.

powerful in-storage processing and also higher memory capacity. Reference [14] also handled more general applications.

However, their approaches have not been practically adopted due to insignificant performance gain, the lack of data-intensive killer applications, and the limitations of storage interface. In particular, due to the physical limitations of HDDs, the low internal I/O bandwidth of HDDs became the main performance bottleneck in processing data inside the active disk. There is another research effort that re-explores the concept of active disk for unstructured data processing applications [16]. However, it also has not been practically adopted due to the same reasons.

Recently, in order to fully utilize the internal I/O bandwidth of the SSD, some studies to adopt the concept of active disk to the SSD have been done. References [4] and [17] tried to attach *special-purpose hardware* for scan operations in DBMSs to each channel in the SSD. Reference [3] ported some query processing components within real SSD devices. Reference [18] also ported map operations in the MapReduce framework in real SSD devices.

While these studies and our approach all focus on the SSD, their target applications and inside architecture assumed for the SSD are different. The operational mechanisms of efficient data mining applications are quite different from one another [19,20]. Thus, it is infeasible to employ special-purpose hardware for every data mining application inside the SSD as done in References [4] and [17]. Moreover, in order to perform data mining applications inside our iSSD, unlike the simple scan operations and map operations, the iSSD requires more processing power and memory capacity.

Nevertheless, the iSSD could be a very good solution for large-scaled data mining applications because data mining applications have the following data-intensive characteristics: (1) frequent accesses of data and repeated executions of simple operations and (2) high scalability of data parallelism. Furthermore, we note that current internal I/O bandwidth of the SSD is bound to that of the host interface even if the internal I/O bandwidth in SSDs can easily scale up. Therefore, we expect that exploiting the iSSD for data mining applications should be successful.

3. Intelligent SSD

In this section, we review the architecture of the current SSD and then present design considerations and the potential benefits of the iSSD in terms of data mining.

3.1. Background: Architecture of an SSD

Figure 1 shows the overall architecture of an existing SSD with multi-way and multi-channel interleaving [2]. Typically, the SSD contains 8∼32 channels. In each channel, there are (1) 8∼16 *NAND flash memory cells* for storing data, (2) a *flash memory controller* (FMC) for managing all the flash memory cells in each channel, and (3) *DRAM* for reading/writing data from/to flash memory cells [2]. All the channels are managed and controlled by firmware called *flash translation layer* (FTL) [4]. Embedded CPUs (SSD cores) execute the FTL and DRAM saves FTL metadata (core memory) in the SSD.

Table 1 shows the I/O bandwidth of each part [4]. The internal I/O bandwidth of the SSD with 8-way and 16-channel is 6.4GB/s [4]. This indicates that the internal bandwidth of the SSD is bound to that of the host interface (PCIe x16: 6GB/s). Therefore,

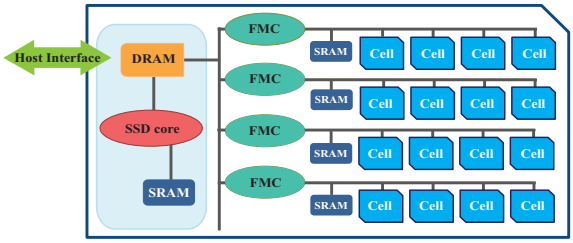

Fig. 1. Architecture of an SSD.

currently enlarging the internal bandwidth of SSDs is meaningless even though SSDs provide higher scalability of internal I/O bandwidth.

Table 1. I/O bandwidth of each part in SSDs

	I/O bandwidth
Host interface	SATA 3G: 250MB/s SAS 6G: 1GB/s PCIe x16: 6GB/s
NAND flash memory cell	SLC: 40MB/s MLC: 400MB/s
Main memory	DDR3: 6.4GB/s
Internal bandwidth of an SSD	8-way, 16-channel: 6.4GB/s

3.2. Intelligent SSD: A Proposal

In this section, we first discuss our design considerations of the intelligent SSD (iSSD) for efficient in-storage data processing[7]. Based on this, we then analyze the suitability of the iSSD in terms of data mining applications.

The iSSD should have the following structural characteristics. First, the iSSD should have internal I/O bandwidth higher than that of a current SSD. With the iSSD having in-storage processing power, the bandwidth of the host interface is not the performance bottleneck of data-intensive applications anymore. Therefore, by expanding the internal I/O bandwidth of the SSD, the iSSD will be able to access and to process a large volume of data more efficiently. As mentioned earlier, the internal I/O bandwidth in iSSDs can easily scale up by increasing the number of channels and the number of ways inside the iSSD [4].

Second, in order to fully exploit the internal I/O bandwidth, reasonable processing power needs to be equipped inside the iSSD; otherwise, it could become a new performance bottleneck of in-storage processing. We could add a cheap (while having relatively-

[7] It is noteworthy that a special-purpose host interface such as object storage devices [22,23] is needed for processing data inside the iSSD. However, such an interface is orthogonal to our contributions, and will not be mentioned further.

low performance) processor called *channel core* and a small memory called *channel memory* to each channel inside the iSSD. The advantages with this approach are as follows: (1) Data can be processed over all channel cores *in parallel*. The size of the data processed in a channel core is in inverse-proportion to the number of channels in the iSSD. Therefore, the cheap channel core and small channel memory sufficiently take advantage of the channel I/O bandwidth if we increase the number of channels inside the iSSD; (2) Adding a channel into the iSSD provides not only the expansion of the internal bandwidth but also the improvement of the inside processing power, thereby not burdening both the host interface and the host CPU; (3) A number of cheap channel cores are better than a single or a few high performance SSD cores inside an iSSD in the *thermal aspect* [21].

The processing power and memory size of each part in the iSSD assumed in this paper are as follows.

- Processing power: host CPU $>>$ SSD core $>$ channel core
- Memory size: host CPU $>>$ SSD core $>$ channel core

Data processing in the iSSD (1) requires a data access cost lower than data processing based on the host CPU and (2) can process data over all channel cores in parallel. Therefore, we expect that data mining applications having data-intensive characteristics can be performed efficiently in the iSSD. In Section 4, we discuss the strategies for data processing in the iSSD, and analyze the execution costs of data mining applications for predicting their performance in the iSSD.

4. Data Mining in iSSD

In this section, we first propose three strategies for running data mining functions efficiently in the iSSD. Then, we formulate the cost models for estimating the performance of data processing in the iSSD (In-Storage Processing, ISP) with our strategies and that of traditional data processing based on the host CPU (In-Host Processing, IHP).

4.1. Strategies for running data mining functions in the iSSD

In this paper, we use the following three strategies for efficient in-storage processing of data mining functions in the iSSD.

- *Strategy 1: Process data over all channel cores in parallel.* A parallel paradigm is general and has also been employed in other frameworks such as MapReduce [17] using multiple nodes and GPGPU [25] using multiple cores. Our first strategy is to have the parallel paradigm realized inside the iSSD by giving an FMC core to each channel. Each channel core in the iSSD performs the same task on its own dataset. While the strategy itself is not novel, its application to SSD could be regarded novel and also makes ISP more efficient inside the iSSD via the parallelism. It is fairly beneficial to data-intensive applications including data mining and databases.
- *Strategy 2: Using the internal memory.* It is to make each channel possible to process its data inside the iSSD without soliciting the usage of main memory in host. We get rid of transferring time between main memory in host and a flash memory cell in the iSSD and use only the internal memory in the iSSD. As a result, we can reduce the total execution time by removing the data transfer cost between main memory in host and flash memory cell.

– *Strategy 3: Conduct the global merge in SSD cores.* Some data mining applications require global merge that aggregates multiple local results obtained independently from channel cores. The global merge adversely affects the performance in the ISP because it cannot be performed over channel cores in parallel. We assume to have global merge conducted in SSD cores since they directly communicate with every channel core and have computing power higher than channel cores.

4.2. Cost models of data mining applications

In this section, we analyze the execution costs of data mining applications. Table 2 summarizes the notations used throughout this paper. We note that there are no real-world iSSDs (no hardware and no firmware) provided yet by SSD manufacturers. This makes us unable to evaluate the effectiveness of the in-storage processing based on our iSSD in comparison with that of the traditional in-host processing. Thus, we instead formulate two cost models for the IHP and the ISP by estimating the performance of data mining applications in the host and iSSD, respectively. With the cost models, we can easily investigate the performance change affected by parameter settings under different host and iSSD architectures.

Figure 2 shows the steps of processing data stored in cells. The IHP requires three steps for data accesses: (1) from a cell to SRAM of a channel core, (2) from SRAM of a channel core to DRAM of a SSD core, and (3) from DRAM of a SSD core to main memory in the host. The steps of transferring data processed in the host to cells are performed in the reverse order.

In contrast, the ISP requires only steps (1) and (2) during processing data stored in cells because it does not need to transfer data to the host. Also, we note step (2) is unlikely to occur frequently because SSD cores are mainly used for merging data obtained from channel cores that are in charge of processing data stored in their cells.

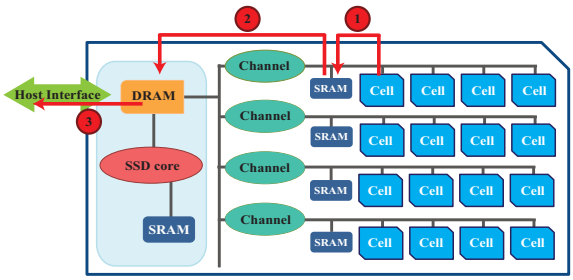

Fig. 2. Execution steps of processing data in cells.

It is noteworthy that we formulate the execution cost of a target data mining application in a *unit of a function*. This is because, even though functions belong to the same data mining application, their execution times could be quite different depending on their characteristics.

Data access cost We begin by defining some preliminaries to describe the cost to transfer d bytes in each path, similar to the derivation in [4].

Table 2. Notations

Notation	Description
N_{ch}	Number of channels in the iSSD
N_{way}	Number of ways in a channel
S_{page}	Size of a single NAND flash memory page
M_{host}	Size of host memory
M_{core}	Size of core memory
M_{ch}	Size of each channel memory
$t_{cell-read}$	Time to read a page from flash memory cell
$t_{bus-load}$	Time to load a page from a flash memory bus after busy phase for a page read
$t_{core-read}$	Time to read d bytes from core memory
$t_{core-write}$	Time to write d bytes to core memory
$t_{host-write}$	Time to write d bytes to host memory through the host interface
$\|F\|$	Number of functions in the target data mining application
$t_{IHP(f_k)}$	Time for performing function f_k in the host CPU
$t_{ISP(f_k)}$	Time for performing function f_k in the channel core
$N(f_k)$	Size of input data for function f_k
$M(f_k)$	Size of memory required for storing an immediate result for function f_k
$t_{merge(f_k)}$	Time for performing a global merge for function f_k in the SSD core
$M_{merge(f_k)}$	Size of memory required for performing global memory for function f_k
N	Size of total data
$R(f_k)$	Size of a local result from the channel core for function f_k

• **Cell → channel:** Equation 1 represents the cost to read d bytes from flash memory cell to channel memory.

$$t_{cell \to ch} = \lceil \frac{d}{S_{page} \times N_{way}} \rceil \times t_{cell-read} + \lceil \frac{d}{S_{page}} \rceil \times t_{bus\_load} \quad (1)$$

• **Channel → core:** For efficiency, data transfer between channel memory and core memory employs *the direct memory access (DMA) operation* and *the pipeline strategy* [4]. Equation 2 represents the cost to read d bytes from channel memory to core memory.

$$t_{ch \to core} = max(t_{cell \to ch}, t_{core-write}) \quad (2)$$

• **Core → host:** Similar to $t_{ch \to core}$, the data transfer between core memory and host memory also employs the DMA operation and the pipeline strategy. Equation 3 represents the cost to read d bytes from core memory to host memory.

$$t_{core \to host} = max(t_{host-write}, t_{core-read}) \quad (3)$$

The data access costs to transfer d bytes in each path, derived from above preliminaries, with the IHP and the ISP are shown in Equations 4 and 5, respectively.

$$\textit{data\_access(IHP)} = \sum_{k=1}^{|F|}((t_{ch\rightarrow core}/N_{ch} + t_{core\rightarrow host}) \times N(f_k)/d) \quad (4)$$

$$\textit{data\_access(ISP)} = \sum_{k=1}^{|F|}(t_{cell\rightarrow ch}/N_{ch} \times N(f_k)/d) \quad (5)$$

As shown in Figure 2, the data transfer path of the ISP is shorter than that of the IHP. Moreover, in the ISP, data do not move through the host interface that is a new bottleneck in the iSSD. Consequently, the data access cost in the ISP is much smaller than that in the IHP.

Data processing cost The data processing costs of the target data mining application in the IHP and in the ISP are formulated as shown in Equations 6 and 7. Note that $t_{IHP-swap(f_k)}$ ($t_{ISP-swap(f_k)}$) is the swap cost for function f_k in the IHP (in the ISP) as shown in Equations 8 and 9. The data processing cost depends mainly on a *CPU rate* of the place where the function is performed. The processing power of each channel core is much lower than that of host CPU. However, the iSSD performs data mining applications on a number of channel cores in parallel. Therefore, the aggregated processing power from all the channel cores in the iSSD is enough to execute data-intensive applications efficiently.

$$\textit{data\_processing(IHP)} = \sum_{k=1}^{|F|}(t_{IHP(f_k)} + t_{IHP-swap(f_k)}) \quad (6)$$

$$\textit{data\_processing(ISP)} = \sum_{k=1}^{|F|}(t_{ISP(f_k)}/N_{ch} + t_{ISP-swap(f_k)}) \quad (7)$$

The swap cost is influenced by (1) the memory size required for storing an immediate result for function f_k, ($M(f_k)$), (2) the memory size of the place where the function f_k is performed, and (3) the input data size for function f_k, ($N(f_k)$). The swap cost is also influenced by the swapping strategy such as round robin and LRU. In this paper, we assume that LRU is used as a swapping strategy. However, since most data mining applications access the whole data sequentially during their execution [5,6,24], the *buffering effect* due to the LRU strategy would be insignificant.

$$t_{IHP-swap(f_k)} = \lceil M(f_k)/M_{host} \rceil \times (t_{ch\rightarrow core}/N_{ch} + t_{core\rightarrow host}) \times N(f_k)/d \quad (8)$$

$$t_{ISP-swap(f_k)} = \lceil M(f_k)/M_{ch} \rceil \times t_{cell\rightarrow ch}/N_{ch} \times N(f_k)/d \quad (9)$$

As mentioned above, in the ISP, some data mining applications require the global merge that aggregates all the local results produced by channel cores. The global merge

adversely affects the performance in the ISP because it cannot be performed over multiple channel cores in parallel. The global merge cost in the ISP is shown in Equation 10.

$$\textit{merge(ISP)} = \sum_{k=1}^{|F|}(t_{merge(f_k)} + (t_{ch \rightarrow core}/d \times R(f_k) \times N_{ch})) \tag{10}$$

The global merge cost is influenced by (1) the size of a local result from each channel core, (2) the number of channels in the iSSD, and (3) the complexity of the global merge: i.e., the global merge cost increases as the size of a local result, the number of channels, or the complexity of the global merge increases.

Total execution cost Equations 11 and 12 show the total execution costs of a data mining application in the IHP and in the ISP, respectively.

$$\textit{total\_cost(IHP)} = \textit{data\_access(IHP)} + \textit{data\_processing(IHP)} \tag{11}$$

$$\textit{total\_cost(ISP)} = \textit{data\_access(ISP)} + \textit{data\_processing(ISP)} + \textit{merge(ISP)} \tag{12}$$

With these equations, we can compare the performance of the ISP with that of the IHP. In Section 5, we first validate our cost models and then quantify the degree of the performance improvement in the ISP over the IHP by using the cost models.

5. Performance Evaluation

5.1. Cost model validation

We first validate the proposed cost models by using three well-known data mining applications of k-means (for data clustering) [26], PageRank (for web page ranking) [27], and Apriori (for frequent pattern mining) [28]. Note that, Apriori requires the *global merge* to aggregate all the local results, each of which is the local frequency of a candidate itemset obtained by each channel core, in order to get its global frequency [28].

Table 3 shows the execution characteristics of the primitive functions in those data mining applications. To extract *the number of cycles per instruction* (CPI) and *the number of instructions* (#INS) in each function of the target data mining application, we executed the application with V-Tune[8]. The data and parameter settings to obtain the results are shown in Table 4. As a result, we have a total of 27 cases obtained by varying the sizes of data and the parameter values of data mining applications.

We can estimate the total execution time of each data mining application in the IHP (ISP) to substitute the execution characteristics shown in Table 3 into Equation 11 (12). For the data access cost, we used the Samsung SSD datasheet[9]. To obtain the time for performing each function in the data processing cost, we used Equation 13.

$$\textit{Function execution time} = \textit{(CPI} \times \textit{\#INS) / CPU rate} \tag{13}$$

[8] http://software.intel.com/en-us/intel-vtune-amplifier-xe/

[9] http://www.samsung.com/global/business/semiconductor/product /flash-ssd/catalogue

Table 3. Descriptions of data mining applications

k-means - A method for data clustering				
	CPI	# of INS	Data access	Global merge required
create_initial_center	0.60	15939.00	-	X
calculate_distance	0.75	3553.81	Read N bytes	X
clustering	2.98	3.61	-	X
insert_entry	0.99	212.63	Write N bytes	X
update_MSE	0.62	86.22	-	X
calculate_new_center	0.75	1268.35	-	X
PageRank - A method for web page ranking				
initlalize_ranking_vector	0.56	199.41	Read N bytes	X
matrix_multiplication	0.60	18.12	-	X
add_restart_vector	2.22	3.80	Write N bytes	X
Apriori - A method for frequent pattern mining				
count_frequency	0.50	10.02	Read N bytes	X
check_MinSupport	0.56	7.32	-	O
create_candidate_itemsets	0.51	42.89	-	O
create_frequent_itemsets	1.57	49.58	Write N bytes	O

Table 4. Data and parameter settings for our data mining applications

Algorithm	Description for experiments
k-means	The number of objects: 100,000 $\sim$ 700,000 (in step of 100,000)
	The number of dimensions per object: 8
	The number of iterations for convergence: 30
PageRank	The number of nodes: 200,000 $\sim$ 1,000,000 (in step of 200,000)
	The number of edges: the number of nodes $\times$ the number of nodes
	The number of iterations for convergence: 10
Apriori	The number of transactions: 4,000 $\sim$ 12,000 (in step of 4,000)
	The threshold: 4 $\sim$ 16 (in step of 4)
	The average number of items in each transaction: 8

To validate the accuracy of our cost models, we computed the Pearson Correlation Coefficient (PCC) between (1) the execution time of k-means, PageRank, and Apriori estimated by our IHP model and (2) the execution time obtained by executing them separately in a real machine. We measured the total execution times in the same way as in Table 4.

Figure 3 shows the results. The x-axis indicates the execution time in a real machine and the y-axis does the execution time estimated by our model. A single point indicates the execution time results for each case. We observe they are correlated significantly as all the points are located closely on the graph of $y = x$. The PCC between two values for 27 cases is shown to be 0.997, indicating that the proposed cost model is very accurate.

5.2. Experimental Setup

For experiments, we measured the performance of the IHP and the ISP under different iSSD architectures as shown in Table 5. A **boldface letter** represents a pivot value of

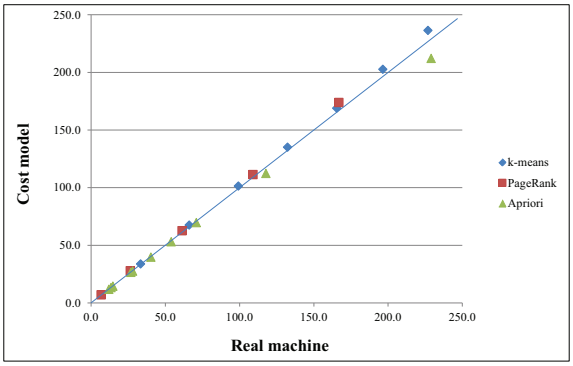

Fig. 3. PCC between the real and estimated execution time.

each parameter. While changing the values of one parameter, the other parameters are fixed to their pivot value. We used 1GB data obtained from MineBench [18].

Both of data mining applications and their data are stored in the iSSD. For processing, a data mining application is first loaded from the iSSD to memory in host and then is offloaded to memory on the iSSD. Inside the iSSD, the application performs its task by processing the data stored in channels. The time for offloading the application is a new one additionally required for ISP. However, a data mining application is normally quite small in size, compared with data. Specifically, the size of a data mining application is only several tens of kilobytes while that of data is typically several gigabytes as in the experimental section. Moreover, the data need to be loaded multiple times during data analysis while the application is offloaded only once. Therefore, we did not include the offload time, which is negligible in the total execution time.

Table 5. Parameter values for experiments

	Parameter	Values(s)
iSSD	#channels	**32**, 64, 128, 256
	#ways	8
	SSD core rates (MHz)	400
	channel core rates (MHz)	**200**, 400, 600, 800
	Core memory size (MB)	400
	Channel memory size (KB)	2, 4, **8**, 16
	Cell read time (μs)	50
	Cell write time (μs)	1,200
	Page size (B)	8k
Host interface	Bandwidth (Gbps)	3
Host	Host CPU rates (GHz)	2.5
	Host memory size (GB)	4

5.3. Results and analyses

ISP with different iSSD settings In this experiment, we first changed the number of channels in the iSSD as 32, 64, 128, and 256, respectively. Other parameters were set to pivot values shown in Table 5. As shown in Figure 4, the performance of the ISP in all cases dramatically increases as the number of channels increases. In particular, it increases almost linearly with the number of channels for k-means and PageRank which do not have the stage of the global merge in their execution. However, in the case of Apriori, the performance is improved much less, which is due to its global merge.

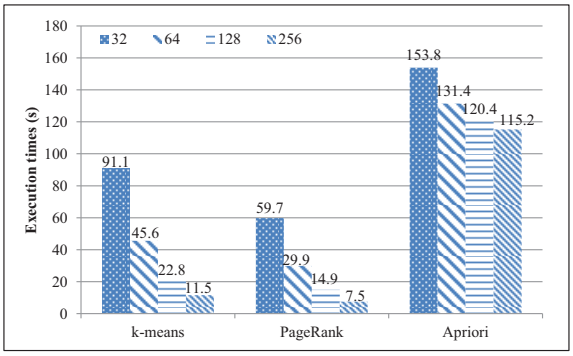

Fig. 4. Performance of the ISP with different numbers of channels.

We also changed the channel core rates as 200MHz to 800MHz in step of 200MHz. Other parameters were also set to pivot values shown in Table 5. In Figure 5, we observe linear performance improvement with channel core rates in k-means and PageRank. This indicates that the internal processing power may become a new performance bottleneck of data mining applications which do not have the stage of global merge leading to (high parallelism) in the iSSD.

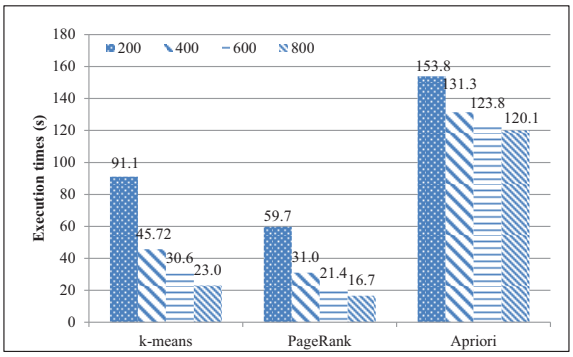

Fig. 5. Performance of the ISP with different channel core rates.

Then, we changed the channel memory size as 2KB, 4KB, 8KB, and 16KB, respectively. Table 6 indicates the performance result. The performance of the ISP is almost the same and is independent of the channel memory size. This result indicates that k-means, PageRank, and Apriori require just a small memory to store an intermediate result during their execution, and the small channel memory is enough to hold all the intermediate result. Therefore, we can perform those data mining applications in the iSSD efficiently.

Table 6. Performance of the ISP with different channel memory sizes

	2	4	8	16
k-means	91.1	91.1	91.1	91.1
PageRank	59.7	59.7	59.7	59.7
Apriori	154.0	153.9	153.8	153.8

Comparison of the ISP with the IHP For showing the potential benefits of the iSSD quantitatively, we compared the performance of the IHP and the ISP with k-means, PageRank, and Apriori. For the IHP, we deployed not only SSD but also HDD for examining performance change with different storage media. For the ISP, we considered two iSSD architectures: iSSD_C (typical settings in the *current* SSD) with 32 channels and their CPUs of 200MHz and iSSD_F (settings in the *future* iSSD) with 256 channels and their CPUs of 800MHz.

Figure 6 shows the results. The ISP in both iSSD architectures outperforms the IHP for k-means and PageRank that do not require a global merge stage. Even with the current settings (iSSD_C), the ISP outperforms the IHP about $2 \sim 3$ times. With the future settings (iSSD_F), the ISP dramatically improves the performance of the IHP up to 83 times.

On the other hand, Apriori with a global merge shows opposite results. In Apriori, ISP(iSSD_C) shows lower performance than the IHP(SSD), and ISP(iSSD_F) also shows insignificant performance improvement over the IHP(SSD). This is because the SSD core that performs the global merge becomes a new performance bottleneck in the execution of data mining applications. Such findings are vividly illustrated in Figure 7.

Figure 7 shows the breakdown result of the ISP(iSSD_C) and ISP(iSSD_F). In this figure, transfer and process indicate the data transfer time between a channel and an SSD core and the data processing time on channels, respectively. In the ISP, the figure shows that the internal communication time is not significant. In particular, as the number of channels and channel core rates increase, the execution time on each channel core decreases.

The global merge costs of Apriori in both ISP(iSSD_C) and ISP(iSSD_F) are almost same. This is because, as the number of channels increases, the number of local results to be merged in the SSD core also increases. As a result, in Apriori, the data processing in the SSD core becomes a new performance bottleneck. As a possible solution to solve the performance bottleneck in the SSD core, we can consider the method to conduct the

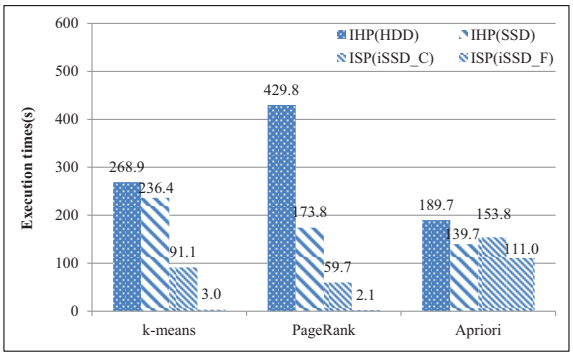

Fig. 6. Performance comparison of the ISP with the IHP.

global merge with the help from the host CPU which has much higher processing power than the SSD core.

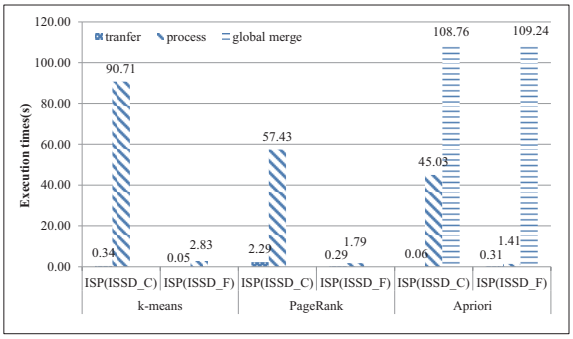

Fig. 7. Breakdown of the ISP cost.

We indeed measured the performance of Apriori to conduct the global merge in the host CPU (denoted as IHP+ISP). Figure 8 shows the results. The figure shows that the IHP+ISP outperforms the others about 3.3 $\sim$ 8 times. We think that this result gives us a future research direction to collaborate the iSSD with the host CPU in order to maximize the overall performance.

Correlation between the execution times obtained by the iSSD simulator and the cost model In this section, we performed an experiment to show the correlation between the execution times obtained by the iSSD simulator [29] and the cost model. We developed our iSSD simulator to verify our cost model on top of the gem 5 simulator. The gem 5 simulator [30] provides a virtual computing environment that makes it possible to run applications as in someones target system.

Simulation-based evaluation normally consumes a huge amount of time even with a small size of data because it requires to run algorithms in the simulation environment.

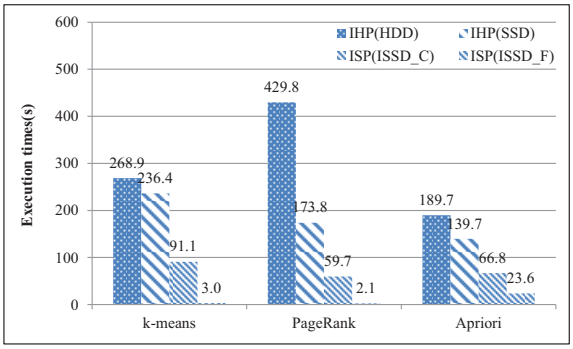

Fig. 8. Performance of the IHP + ISP.

Thus, it is infeasible to evaluate the performance of ISP on the iSSD by using our iSSD simulator in the case of a large size of data. In contrast, a cost model predicts the execution time very quickly based on a simple formula, but could have a problem of producing inaccurate results.

Therefore, we verify our cost model to evaluate the performance of the iSSD by comparing its results with those of our iSSD simulator. If the cost model shows reasonable coincidence with the simulator, we can use it to estimate quickly the execution time of data mining algorithms even with the large size of data. To compare the execution times, we used results shown in Table III of [29]. We computed the Pearson correlation coefficient between the time by simulation and that by the cost model to show their coincidence.

Table 7 shows the results, which indicate that the Pearson correlation appears to be very close to one in all the three data mining algorithms.

Table 7. Execution times with the iSSD simulator and the cost model

Algorithms	PCC
Apriori	0.9989
k-means	0.9997
PageRank	0.9998

6. Conclusions

In this paper, we introduced the iSSD as a turbo for big data mining. The detailed contributions of this paper are summarized as follows.

– First, we presented our design considerations for the iSSD. The iSSD should have (1) enlarged internal I/O bandwidth and (2) sufficient inside processing power. Our ISP

(1) incurs a low data access cost compared with the conventional IHP and (2) handles data over all channel cores in parallel.
- Second, we proposed the strategies for the ISP. Our strategy is (1) to process data over all channel cores in parallel, (2) to use flash memory cell as virtual memory of channel memory, and (3) conduct the global merge in SSD cores since they efficiently communicate with every channel core and have performance higher than channel cores.
- Third, we formulated the execution cost models of data mining applications for predicting their performance in the iSSD. In the ISP, the execution cost is composed of (1) a data access cost, (2) a data processing cost for accessed data, and (3) a global merge cost for aggregating all the local results in SSD cores.
- Finally, we validated our cost models and showed the potential of the ISP with data mining applications through a series of simulations. The results show that the ISP achieves dramatical speed up by up to 83 times compared with the current the IHP.

As future work, we plan to do research on a novel job scheduling algorithm where the iSSD and the host CPU collaborate with each other in order to maximize the overall performance. To the end, the job scheduling algorithm should be intelligent enough to know the characteristics of applications and hardware specifications such as the local result size, the global merge overhead, and the I/O bandwidth of a host interface. Also, we plan to do research on efficient ways to exploit a large number of iSSDs in super-computer environment. For this, we are devising a data placement strategy that minimizes the dependency among data to be analyzed.

Acknowledgement. This research was supported by (1) Semiconductor Industry Collaborative Project between Hanyang University and Samsung Electronics Co. Ltd., (2) the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2014R1A2A1A10054151), (3) the ICT R&D program of MSIP/IITP (B0101-15-0266, Development of High Performance Visual Big-Data Discovery Platform for Large-Scale Realtime Data Analysis), and (4) the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) (No. 2015R1A5A7037751).

References

1. Bae, D., Chang, J., Kim, S.: An efficient method for record management in flash memory environment. Journal of Systems Architecture 58(6), 221-232 (2012)
2. Lee, S., Moon, B., Park, C.: Advances in flash memory SSD technology for enterprise database applications. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. pp.863-870. ACM, New York, USA (2009)
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 2013 ACM SIGMOD International Conference on Management of Data. pp. 1221-1230. ACM, New York, USA (2013)
4. Kim, S., Oh, H., Park, C., Cho, S., Lee, S.: Fast, energy efficient scan inside flash memory SSDs. In: Proceedings of the International Workshop on Accelerating Data Management Systems. pp.36-43. (2011)
5. Choudhary, A., Honbo, D., Kumar, P., Ozisikyilmaz, B., Misra, S., Memik, G.: Accelerating data mining workloads: current approaches and future challenges in system architecture design. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(1), 41-54 (2011)

6. Zambreno, J., Ikylmaz, B., Memik, G., Choudhary, A.: Performance characterization of data mining applications using MineBench. In: Proceedings of 9th Workshop on Computer Architecture Evaluation using Commercial Workloads. Citeseer (2006).
7. Schuster, S., Nguyen, H., Ozkarahan, E., Smith, K.: RAP. 2-An associative processor for databases and its applications. IEEE Transactions Computers 100(6), 446-458 (1979)
8. Lewis, T.: Data parallel computing: An alternative for the 1990s. Computer 24(9), 110-111 (1991)
9. Bae, D., Kim, J., Kim, S., Oh, H., Park, C.: Intelligent SSD: a turbo for big data mining. In: Proceedings of the 2013 ACM International Conference on Information and Knowledge Management. pp. 1573-1576. ACM, New York, USA (2013)
10. Su, S., Lipovski, G.: CASSM: A cellular system for very large data bases. In: Proceedings of the 1st International Conference on Very Large Data Bases. pp. 456-472. ACM, New York, USA (1975)
11. Boral, H., DeWitt, D.: Database machines: An idea whose time has passed?. In: Proceedings of International Workshop on Database Machines. (1983)
12. Riedel, E., Gibson, G., Faloutsos, C.: Active storage for large-scale data mining and multimedia. In: Proceedings of the 24rd International Conference on Very Large Data Bases. pp. 62-73. Morgan Kaufmann, San Francisco, USA (1998)
13. Acharya, A., Uysal, M., Saltz, J.: Active disks: Programming model, algorithms and evaluation. In: Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 81-91. ACM, New York, USA (1998)
14. Keeton, K., Patterson, D. A., Hellerstein, J. M.: A case for intelligent disks (IDISKs). ACM SIGMOD Record 27(3), 42-52 (1998)
15. Kim, J., Bae, D., Kim, S., Oh, H., and Park, C.: An efficient data mining algorithm for intelligent SSD. In: Journal of KIISE 39(1), 178-179 (2012)
16. Smullen, C., Tarapore, S., Gurumurthi, S., Ranganathan, P., Uysal, M.: Active storage revisited: the case for power and performance benefits for unstructured data processing applications. In: Proceedings of the 5th conference on Computing Frontiers. pp. 293-304. ACM, New York, USA (2008)
17. 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 International Conference on Supercomputing. pp. 91-102. ACM, New York, USA (2013)
18. 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. IEEE (2013)
19. Choudhary, A., Narayanan, R., Ikylmaz, B., Memik, G., Zambreno, J., Pisharath, J.: Optimizing data mining workloads using hardware accelerators. In: Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads. Citeseer (2007)
20. Narayanan, R., Ozisikyilmaz, B., Zambreno, J., Memik, G., Choudhary, A.: Minebench: A benchmark suite for data mining workloads. In: Proceedings of 2006 IEEE International Symposium on Workload Characterization. pp. 182-188. IEEE (2006)
21. Reddi, V., Lee, B., Chilimbi, T., Vaid, K.: Web search using mobile cores: Quantifying and mitigating the price of efficiency. ACM SIGARCH Computer Architecture News 38(3), 314-325 (2010)
22. Du, D., He, D., Hong, C., Jeong, J., Kher, V., Kim, Y., Lu, Y., Raghuveer, A., Sharafkandi, S.: Experiences in building an object-based storage system based on the OSD T-10 standard. In: Proceedings of 23rd IEEE Conference on Mass Storage Systems and Technologies. IEEE (2006)
23. Mesnier, M., Ganger, G., Riedel, E.: Object-based storage. Communications Magazine 41(8), 84-90 (2003)
24. Ozisikyilmaz, B., Narayanan, R., Zambreno, J., Memik, G., Choudhary, A.: An architectural characterization study of data mining and bioinformatics workloads. In: Proceedings of 2006 IEEE International Symposium on Workload Characterization. pp. 61-70. IEEE (2006)

25. Jo, Y., Kim, S., and Bae, D.: Efficient Sparse matrix multiplication on GPU for large social network analysis. In Proceedings of International conference on Information and Knowledge Management. pp 1261-1270. ACM (2015)
26. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability. pp. 281-297. University of California Press (1967)
27. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Technical Report, Stanford InfoLab (1999)
28. 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. Morgan Kaufmann, Santiago, Chile (1994)
29. Jo, Y., Chung, M., Kim, S., and Oh, Hyunok.: Data mining in intelligent SSD: simulation-based evaluation. In: Proceeding of Big Data and Smart Computing (BigComp). pp. 123-128. KIISE (2016)
30. Binkert, N., Beckmann, B., Black, G., Reinhardt, S., Saidi, A., Basu, A., Hestness, J., Hower, D., Krishna, T., and Sardashti, S. The gem5 simulator. In: ACM SIGARCH Computer Architecture News 39(2), 1-7 (2011)

Duck-Ho Bae received the BS, MS, and PhD degrees in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2006, 2008, and 2013, respectively. Currently, he is a senior engineer at Samsung Electronics. His research interests include data mining, databases, and distributed systems.

Jin-Hyung Kim received the MS degree in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2013. He is currently working as a system engineer in Hyundae Autoever.

Yong-Yeon Jo received the MS degree in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2013. He is currently a PhD 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 BS degree in Computer Engineering from Seoul National University, Seoul, Korea in 1989 and earned the MS and PhD 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.

Hyunok Oh received the BS, MS, and PhD degrees in Computer Engineering from Seoul National University, Seoul, Korea, in 1996, 1998, and 2003, respectively. He is currently an associate professor in Department of Information Systems, Hanyang University, Seoul, Korea. His research interests include cryptography, embedded system design automation, non-volatile memory optimization, parallel processing, multimedia, and real-time analysis.

Chanik Park received the B.S. and M.S. degrees in Computer Engineering and the Ph.D. degree in Electrical and Computer Engineering from Seoul National University, Seoul, Korea, in 1995, 1997, and 2002, respectively. He is currently a Vice President with Memory Business at Samsung Electronics, Hwaseong, Korea. His research interests include storage architecture and high-performance and reliable solid-state drive-based NAND flash memories with the assistance of hardware/software codesign.

Received: August 20, 2015; Accepted: April 15, 2016.