Data Mining in Intelligent SSD: Simulation-based Evaluation

Yong-Yeon Jo
Department of Computer and Software
Hanyang University, Korea
Email: jyy0430@hanyang.ac.kr

Moonjun Chung
Division of Computer Science and Engineering
Hanyang University, Korea
Email: chung1246@hanmail.net

Sang-Wook Kim
Department of Computer and Software
Hanyang University, Korea
Email: wook@hanyang.ac.kr

Hyunok Oh
Department of Information Systems
Hanyang University, Korea
Email: hoh@hanyang.ac.kr

***Abstract*—Due to an explosive growth of Internet applications, the amount of data has increased enormously. In order to store and process this big data more efficiently, a solid-state device (SSD) has replaced a hard disk drive (HDD) as a primary storage media. In spite of high internal bandwidth, SSD has its performance bottleneck on the host interface whose bandwidth is relatively low. To overcome the problem of performance bottleneck in big data processing, the notion of intelligent SSD (iSSD) was proposed to give computing power to SSD. However, its real implementation has not been provided to the public yet. In this paper, we are going to verify the potential of iSSD in handling data-intensive algorithms. To the end, we first develop an iSSD simulator and then evaluate the performance of data mining algorithms inside iSSD on the top of it in comparison with that by the host CPU. The results reveal that data mining with iSSD outperforms that with host CPUs up to around 300%.**

I. Introduction

Due to an explosive growth of Internet applications such as Internet business, on-line shopping, e-mails, and social networking services, the amount of data has increased greatly. In order to store and process this big data more efficiently, a solid-state device (SSD) has been used as a primary storage media thanks to its characteristics such as high IO bandwidth and low power consumptions [1][2].

In the case of traditional In-host processing (IHP), the data stored in SSD is loaded into main memory (DRAM) in the host, and then is processed by CPUs in the host. In this case, the performance bottleneck occurs on the host interface that provides relatively low bandwidth (in spite of the high internal bandwidth of SSD) [3][4]. In particular, the problem could get serious when we perform data-intensive algorithms for big data [5].

To overcome such performance bottleneck, in-storage processing (ISP), an approach of processing data by using the computing resources equipped within storage devices, has been proposed [6][7]. In this context, the notion of intelligent SSD (iSSD) has also been appeared [8][9].

iSSD is a special type of SSD equipped with computing resources (cores) in its inside to process data within SSD. iSSD consists of multiple channels (as normal SSD does), each of which has a series of flash memory cells in which data is stored. For processing the data inside iSSD, there are cores (called FMC cores) and memories such as DRAM and SRAM with each channel [10].

ISP with iSSD has the advantages over IHP [10] as follows: (1) the degree of parallelism increases because the number of FMC cores could be much larger than that of CPU cores; (2) the time of transferring the data stored in flash memory cells to processing cores is reduced since the data is processed within iSSD by FMC cores or SSD cores instead of CPU cores; (3) CPU, which is a quite expensive resource, is in charge of much less amount of workloads because it receives a much smaller result processed by iSSD [4][10].

In this paper, we are going to examine the potential of iSSD in handling data-intensive applications. Currently, there is no real implementation of iSSD announced by manufacturers. We thus develop an iSSD simulator first and then evaluate how data mining algorithms perform well inside iSSD to verify the potential of iSSD in real situations.

The contributions of this paper are summarized as follows.

- We developed the iSSD simulator based on the gem 5 simulator [11] that provides a virtual computing environment. The source code for iSSD simulator is released as open source at *https://github.com/hyunokoh/s4sim*.
- We implemented the data mining algorithms on top of the iSSD simulator. The source code of these algorithms is also released as open source at *https://github.com/Yongyeon-Jo/Datamining-algorithms-for-S4Sim*.
- Through extensive experiments, we evaluated the performance of those data mining algorithms with iSSD compared to that with IHP. The experimental results show that data mining with ISP on iSSD outperforms significantly that with IHP up to around 300%.

The rest of this paper is organized as follows. Section 2 briefly describes the implementation of our iSSD simulator based on the gem 5 simulator. Section 3 presents how to

implement data mining algorithms on the iSSD simulator. Section 4 demonstrates and analyzes the experimental results of evaluating data mining algorithms with IHP and ISP on iSSD. Finally, Section 5 summarizes and concludes the paper.

II. Inteligent SSD simulator

In this section, we introduce iSSD (inteligent SSD), and present how to implement the iSSD simulator.

A. iSSD

Figure 1 shows the internal structure of iSSD [10]. iSSD is a special type of SSD that equip a general processing core (called FMC core) with a flash memory controller (FMC) in each channel and also more powerful SSD cores in SSD. It also has SRAM and DRAM to load the data to be read/written from/to cells. 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 [4].

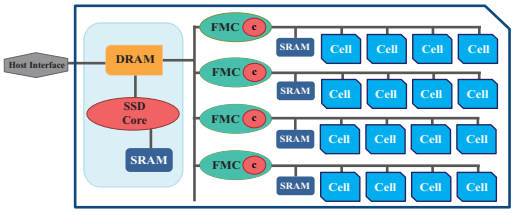


Fig. 1: Internal of iSSD.

B. Implementation of the iSSD simulator

We developed the iSSD simulator to evaluate the performance of iSSD, on top of the gem 5 simulator. The gem 5 simulator [11] provides a virtual computing environment that is able to process applications on someone's target system. It allows configuring a target system by determining types and performance of processors, memory, and cache.

We thus are able to implement our iSSD simulator by employing processors, memory, and cache as our SSD cores, DRAM, and SRAM, respectively. In order to improve the accuracy of the simulator, we set the CPU model as *Atomic-Simple* and the system mode as *System-call Emulation mode* [12].

Table I represents the parameter settings for the iSSD simulator.

TABLE I: Parameter settings for the iSSD simulator

Components	Parameters
SSD core	Type: ARM
	Number of cores: 4
	Core Rates: 400 MHz
DRAM	Type: ddr3_1600_x64
	Size: 8GB
SRAM	Type: L3
	Size: 16MB

In the case of using the gem 5 simulator, there are two difficulties in reality.

First, it is difficult to simulate a channel in such a way that it behaves identically to the iSSD channel [12]. A channel in iSSD has an FMC core and DRAM, which process data stored in its own cells independently of those in other channels. In the original iSSD, FMC cores are thus primarily computing resources to perform ISP [4][10]. However, the gem 5 simulator enforces their cores to share the memory. Therefore, we decided to make the role of FMC cores substituted by SSD cores. The number and rate of SSD cores were set following those of FMC cores.

Second, IO simulation is not allowed in the gem 5 simulator [12]. In order to compute the IO time, we added the following steps: (1) we 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 [13]; (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 IO in parallel within iSSD. As a result, the IO time in iSSD is computed as in Equation 1.

$$IOtime = (\#IO \times T\_IO) \ / \ \#SSDcores \qquad (1)$$

An algorithm is processed in the iSSD simulator as follows: (1) the algorithm implemented in binary code is transferred with the data from the host to iSSD; (2) each SSD core makes a thread of the algorithm executed to process the given data; (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.

III. Implementation of data mining algorithms

In this section, we present the implementation details of the data mining algorithms (specifically, apriori [14], k-means [15], pageRank [16], and decisionTree [17]) on top of the iSSD simulator.

A. apriori

Apriori is one of the most popular algorithms to find frequent itemsets in a database containing a large number of itemsets [14]. Figure 2 shows a flow of apriori. First, it reads itemsets one by one from a database (*Read*). It finds those itemsets frequently occurring in a database by pruning the itemtsets of length *1* (having a single item in it) whose frequency does not meet a given threshold (*PruneItems*). It generates candidate itemsets of length (k+1) from frequent itemsets of length k (*GenCandidates*). It repeats *PruneItems* and *GenCandidates* until no more frequent itemsets are generated. Then, it generates the final frequent itemsets as a result (*GenFreqItemsets*). Finally, it stores the result into the storage device (*Write*).

In our apriori implementation, all the functions except for IO functions such as *PruneItems*, *GenCandidates* and *GenFreqItemsets* are realized as multiple threads in order that we can run them in a parallel way.

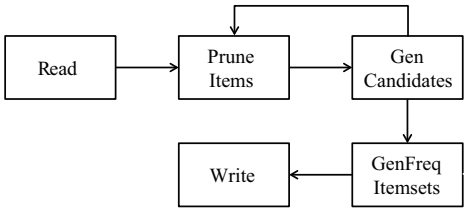


Fig. 2: Flow of apriori.

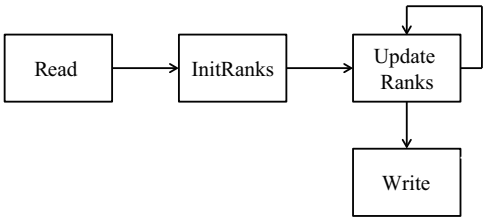


Fig. 4: Flow of pageRank.

B. k-means

k-means is a classical and the most-widely used clustering algorithm in data mining applications. It divides all the objects into k clusters in such a way that objects in a cluster are similar and objects in different clusters are dissimilar [15]. Figure 3 shows the flow of k-means. First, it reads objects from a storage device (*Read*) and randomly selects *k* initial centroid objects for *k* clusters (*SelectCentroids*). For every object, it computes its *k* distances to *k* centroids and makes the object assigned into the cluster whose centroid is nearest to the object (*MakeClusters*). For every cluster, it selects a new centroid in a way that the new one has the minimum average distance to every other object in the clusters (*SelectNewCentroids*). After repeating this process until the centroids are converged, it stores the result into the storage device (*Write*).

In our k-means implementation, *SelectCentroids* is implemented as a single thread because it is simple and requires computations less than other functions. On the other hands, functions such as *MakeClusters* and *SelectNewCentroids* can be parallelized to make data processed more efficiently. Thus, they are realized as multiple threads, each of which is assigned with the same amount of objects to be processed.

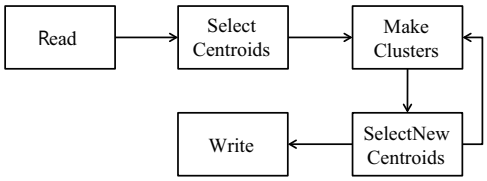


Fig. 3: Flow of k-means.

C. pageRank

pageRank is one of the well-known algorithms of ranking web pages connected as a graph [16]. Figure 4 represents the flow of pageRank. It reads a matrix representing a graph from a storage device (*Read*). Then, it sets initial ranks for all the nodes (*InitRanks*). It renews the rank of every node by considering the scores coming from its incoming edges (*UpdateRanks*). Function *UpdateRanks* is repeated until the ranks of all the nodes are converged. It stores the final ranks for all the nodes into the storage devince (*Write*).

In our pageRank implementation, *InitRank* is realized as a single thread because it is simple and requires much less computations than *UpdateRank*. Because *UpdateRank* needs to be parallelized for efficient processing, it is implemented as multiple threads, each of which is assigned with the same amount of nodes.

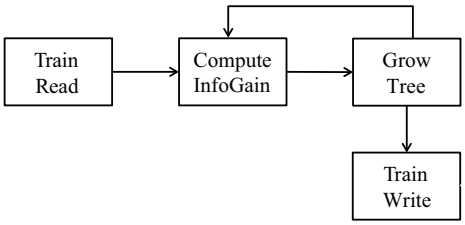
D. decisionTree

decisionTree is one of the most popular algorithms for classification [17]. It performs with two stages: the training and test stages. Figure 5 shows the flow of decisionTree.

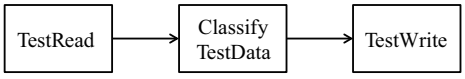
In the training stage, it builds a tree as a model from the training data to be used for classifying new input data. First, it reads the training data from a storage device (*TrainRead*). Then, it computes the information gain (or other measure such as gain ratio) for each attribute in the training data (*ComputeInfoGain*). It creates a new node and its next level edges in a tree by selecting an attribute based on the computed result (*GrowTree*). It repeats the two functions *ComputeInfoGain* and *GrowTree* until there are no more nodes created. It stores the tree in a storage device (*TrainWrite*).

In the test stage, it classifies new input data by referring to the given model represented as a tree. The model and new input data are loaded from the storage device (*TestRead*). It classifies the test data by traversing the tree (*ClassifyTestData*) and stores the result into the storage device (*TestWrite*).

Since *GrowTree* in the training stage and *ClassifyTestData* in the test stage are relatively simple ones, each one is implemented as a single thread. The computations of information gain of attributes are complicated and time-consuming. Thus, they are implemented as multiple threads to be processed in a parallel way.



(a) Training stage.



(b) Test stage.

Fig. 5: Flow of decisionTree.

TABLE II: Descriptions on datasets

Dataset	Type	Details
k-means datset	Synthetic	Number of objects: 10,000
		Domain of an attribute: 0 $\sim$ 1,000
		Dimensionality: 3
apriori dataset	Synthetic	Number of itemsets: 10,000
		Number of item types: 20
		Maximum size of itemset: 10
pageRank dataset	Real-world	Number of nodes: 7,115
		Number of edges: 103,689
decisionTree dataset	Real-world	Number of training data: 700
		Number of test data: 150
		Number of attributes: 19

IV. Evaluation

A. Experimental environments

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 3.2GHz, 8GB main memory, and Intel SSD 530. Table II shows the dataset used in our experiments, some of which (for k-means and apriori) are synthetic ones while the others (for pageRank [18] and decisionTree [19]) are real-world ones. Our iSSD simulator requires a huge amount of time to measure the execution time, which leads us to use only a small dataset for experiments.

B. Results and analyses

In this section, we present the experiment results of evaluating the performance of data mining algorithms performed with ISP in iSSD. The experiments were performed on top of our iSSD simulator. We measured their execution times while changing the parameter values related to hardware components within iSSD. The left side of Figure 6 shows the results according to the number and rate of SSD cores. The *x*-axis indicates the number of cores and the *y*-axis does the execution time. Different lines represent the results with different rates of SSD cores.

The right side of Figure 6 shows the results of details of execution time for each algorithm. Bold face one of functions means that it occupies most of the execution time for its corresponding algorithm. The *x*-axis and *y*-axis are the same as the right side in Figure 6.

1) apriori: Figure 6(a) shows the execution times with changing the number of SSD cores for apriori with ISP on iSSD. The execution time of all the lines decreases, as the number of SSD cores gets larger.

However, a limitation to improve the performance exists in apriori due to a step that sequentially performs in *GenCandidates* which occupies most execution time of apriori as shown in Figure 6(b). We implement all the functions in our apriori as multiple threads. In *GenCandidates*, there is a step to remove overlapping candidate items, which must be sequentially performed. Therefore, this step undermines to enhance the performance using multiple SSD cores on iSSD. We also observe that the execution time decreases, as the rate of SSD cores gets more powerful.

2) k-means: Figure 6(c) shows the execution times with changing the number of SSD cores for k-means with ISP on iSSD. The execution times for all the SSD rates get reduced as the number of SSD cores increases.

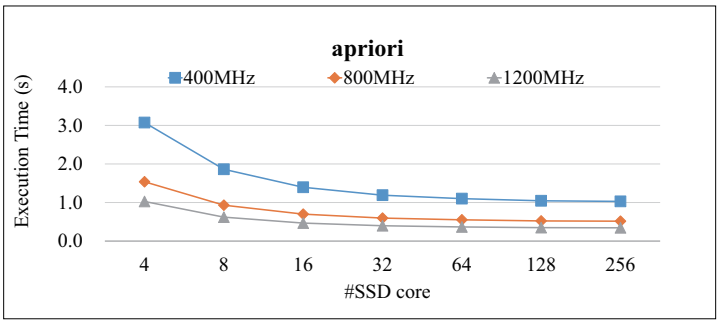
In our k-means implementation on the iSSD simulator, we have *SelectCentroids* performed in a single thread. Fortunately, this function appears not to be a bottleneck that decreases the performance on iSSD because the execution of *SelectCentroids* occupies only a small proportion of a total execution time as shown in Figure 6(d). As in the case of apriori, the execution time decreases with the increasing rate of SSD cores.

3) pageRank: Figure 6(e) demonstrates the execution times with changing the number of SSD cores for pageRank with ISP on iSSD. pageRank runs faster with all the SSD rates as the number of SSD cores grows. This is because *UpdateRank*, parallelized with multiple threads, occupies most of the total execution time in running pageRank as shown in Figure 6(f). The execution time decreases considerably as SSD cores employed are more powerful.

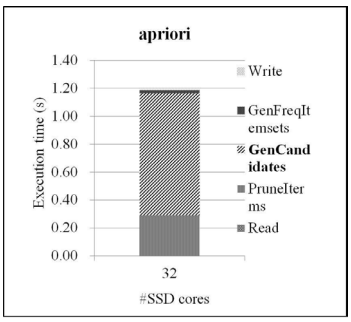
4) decisionTree: Figure 6(g) shows the execution times with changing the number of SSD cores for decisionTree with ISP on iSSD. Although execution times decrease with the number of SSD cores, the performance does not get improved anymore when the number of SSD cores is more than 32. Function *ComputeInfoGain* which is composed of most of the total execution time in decisionTree as shown in Figure 6(h) computes information gain of every attribute in parallel by employing multiple threads. Thus, some SSD cores would not be used in building a decision tree in case there are more SSD cores than the number of attributes.

TABLE III: Execution times of IHP and ISP on iSSD

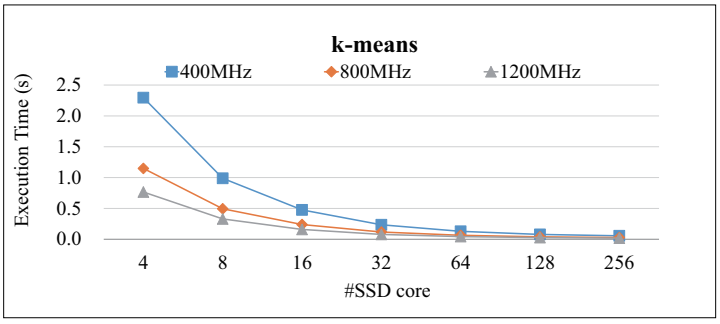
Algorithm	**Execution time (sec)**	
	IHP	**iSSD**
apriori	0.4795	0.3425
k-means	0.0555	0.0185
pageRank	0.0164	0.0117
decisionTree	0.0116	0.0797



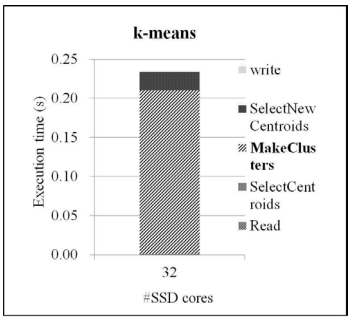
(a) Execution times with changing the number of SSD cores apriori



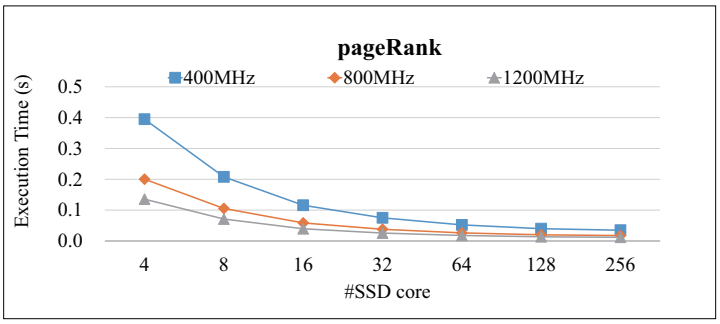
(b) Detail of execution time for apriori



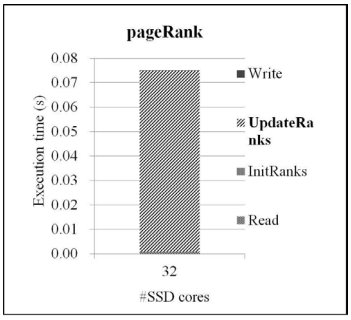
(c) Execution times with changing the number of SSD cores k-means



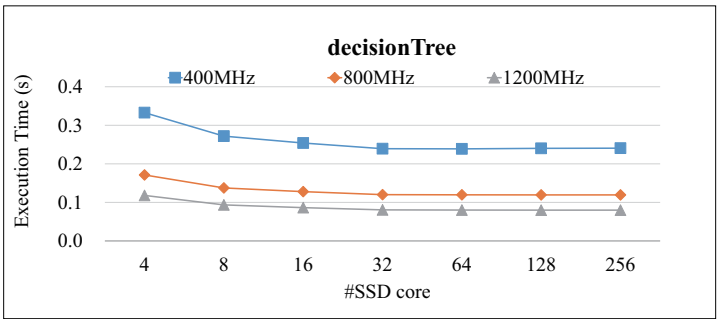
(d) Detail of execution time for k-means



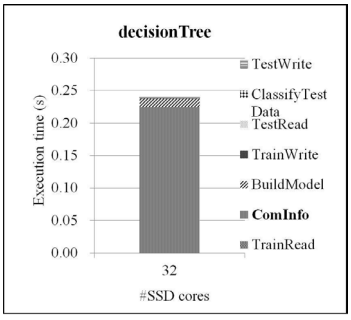
(e) Execution times with changing the number of SSD cores pageRank



(f) Detail of execution time for pageRank



(g) Execution times with changing the number of SSD cores decisionTree



(h) Detail of execution time for decisionTree

Fig. 6: Execution times of data mining algorithms with ISP on iSSD.

C. *Comparisons of ISP on iSSD with IHP*

In this subsection, we compare the performance of ISP on iSSD with that of IHP. To evaluate the performance of IHP, we measured the time of executing the same four 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 1,200MHz, respectively, in our iSSD simulator.

Table III shows the results of the experiments. ISP on iSSD shows performance improvement over IHP up to 140%, 300%, and 140% in the apriori, 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 smaller execution time.

An exceptional case happens in decisionTree. IHP performed faster than ISP on iSSD by 6.83 times, considering both training and test stages. Data used in our experiment keeps SSD cores from being fully utilized. This is because the data is composed of a small number of attributes. We expect that the performance of ISP on iSSD could be better than that of IHP in case data has a more number of attributes.

V. Conclusion

Because of rapid advances of social network & media services and Internet & mobile businesses, the amount of data has been increasing explosively. There have been a number of research efforts to deal with this data more efficiently by exploiting computing resources in addition to CPU in host. A typical example is ISP on iSSD, which is SSD equipped with data processing power inside.

The purpose of this paper is to verify the potential of iSSD by running data mining algorithms as data-intensive applications on it. Because there is no real implementation of iSSD announced by manufacturers, we developed an iSSD simulator and then evaluated how in its data mining algorithms perform well inside iSSD by using the simulator.

Our contributions are summarized as follows:

- First, we developed the iSSD simulator based on the gem 5 simulator. Due to the limitations of the gem 5 simulator, channels having FMC cores and its DRAM were implemented.
- Second, we implemented typical four data mining algorithms (i.e., apriori, k-means, pageRank, and decisionTree) to be performed on top of the iSSD simulator. To exploit SSD cores as much as possible, we identified the functions in the algorithms that can be performed in parallel, and then made them executed in multiple threads.
- Third, we verified the potential of iSSD by a series of experiments. The results show that apriori, k-means, and pageRank with ISP on iSSD outperform those with IHP up to 140%, 300%, and 140%, respectively. However, IHP shows performance better than ISP on iSSD in decisionTree.

Acknowledgment

This work 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) (NRF-2014R1A2A1A10054151)

References

[1] Y. Hu et al., "Exploring and exploiting the multi-level parallelism inside SSDs for improved performance and endurance," *IEEE Trans. on Computers*, Vol. 62, No. 6, pp. 1141-1155, 2013.

[2] S. Lee, B. Moon, C. Park, J. Kim, and S. Kim, "A Case for flash memory SSD in enterprise database applications," In *Proc. ACM Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 1075-1086, 2008.

[3] S. Lee, B. Moon, and C. Park, "Advances in flash memory SSD technology for enterprise database applications," In *Proc. of ACM Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 863-870, 2009.

[4] D. Bae, J. Kim, S. Kim, H. Oh, and C. Park, "Intelligent SSD: a turbo for big data mining," In *Proc. of ACM Int'l Conf. on Information and Knowledge Management*, ACM CIKM, pp. 1553-1556, 2013.

[5] J Do, Y. Kee, J. Patel, C. Park, K. Park, and D. DeWitt, "Query processing on smart SSDs: opportunities and challenges," In *Proc. ACM Int'l Conf. on Management of Data*, ACM SIGMOD, pp.1221-1230, 2013.

[6] A. Acharya, M. Uysal, J. Saltz "Active disks: programming model, algorithms and evaluation," In *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOSVIII, pp. 81-91, 1998.

[7] C. Smullen, S. Tarapore, S. Gurumurthi, P. Ranganathan, and M. Uysal, "Active storage revisited: the case for power and performance benefits for unstructured data processing applications," In *Proc. of Conf. on Computing Frontiers*, CF, pp. 293-304, 2008.

[8] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. Ganger, "Active disk meets flash: a case for intelligent SSDs," In *Proc. of ACM Int'l Conf. on Supercomputing*, SC, pp. 91-102, 2013.

[9] Y. Lee, L. Quero, Y. Lee, J. Kim, and S. Maeng, "Accelerating external sorting via on-the-fly data merge in active SSDs," In *Proc. of USENIX Conf. on Hot Topics In Storage and File Systems*, Hotstorage, pp. 14-19, 2014.

[10] Y. Jo, S. Cho, S. Kim, D. Bae, and H. Oh, "On running data-intensive algorithms with intelligent SSD and host CPU: a collaborative approach," In *Proc. Int'l Conf. on ACM Symp. on Applied Computing*, ACM SAC, pp. 2060-2065, 2015.

[11] Website for the gem 5 simulator, http://www.gem5.org/main_page

[12] N. Binkert et al., "The Gem5 Simulator," *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1-7, 2011.

[13] IO performance of SSD, http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-530-sata-specification.pdf

[14] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Data Bases," In *Proc. Int'l Conf. on Very Large Data Bases*, VLDB, pp. 487-499, 1994.

[15] J. MacQueen et al., "Some Methods for Classification and Analysis of Multivariate Observations," In *Proc. of Berkeley Symp. on Mathematical Statistics and Probability*, Vol. 1, No. 14, pp. 281-297, 1967.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford University, 1999.

[17] M. Friedl and C. Brodley, "Decision Tree Classification of Land Cover from Remotely Sensed Data," *Remote Sensing of Environment*, Vol. 61, No. 3, pp. 399-409, 1997.

[18] pageRank dataset (WikiVote dataset), http://snap.stanford.edu/data/

[19] decisionTree dataset (Complete prepared case study buiness dataset), http://www.wiley.com/legacy/compbooks/soukup/downloads.html