# A Case for Flash Memory SSD in Hadoop Applications

Seok-Hoon Kang, Dong-Hyun Koo, Woon-Hak Kang and Sang-Won Lee

*Dept of Computer Engineering, Sungkyunkwan University, Korea*
*x860221@gmail.com, smwindy@naver.com, woonagi319@skku.edu,*
*wonlee@ece.skku.ac.kr*

***Abstract***

*As the access speed gap between DRAM and storage devices such as hard disk drives is ever widening, the I/O module dominantly becomes the system bottleneck. Meanwhile, the map-reduce parallel programming model has been actively studied for the last few years. In this paper, we will show empirically show that flash memory based SSD(Solid State Drive) is very beneficial when used as local storage devices in IO-intensive map-reduce applications (e.g. sorting) using Hadoop open source platform. Specifically, we present that external sorting algorithm in Hadoop with SSD can outperform the algorithm run with hard disk by more than 3. In addition, we also demonstrate that the power consumption can be drastically reduced when SSDs are used.*

***Keywords:*** *HADOOP, MAPREDUCE, HDD, SSD, I/O bottleneck*

## 1. Introduction

Today, mapreduce-based studies have been actively carried out for the efficient processing of big data hadoop. Hadoop runs on clusters of computers that can handle large amounts of data and java software that support distributed applications. Since the mapreduce model is proposed, a lot of research has been carried out to improve the performance of hadoop. The traditional MapReduce model occur I /O. In this paper, the flash memory based SSD (Solid State Drive) is used and analyzed in order to improve the performance of sort-merge which appears in I / O bottlenecks as previous study [1]. Previous study [1] used the SSD without considering the characteristics of the SSD [3]. So we optimized each parameter on the SSD and tried to improve the performance of the MapReduce when the data is sorted and merged. Section 2 describes the overall structure and characteristics of Hadoop, section 3 describes the characters of SSD and in section 4, the SSD using Hadoop performance test experiments will be described. Finally, section 5 concludes the paper.

## 2. Hadoop Structure

Hadoop is a software framework that supports large-scale distributed computing applications in cluster environments. MapReduce is divided into map task and reduce task. Map task is divided into key (index) and its value. Reduce task processes the data to suit the user's purposes. Then the reduce task must go through the process of sort-merge. Figure 1 depicts a map and reduce operations in progress, based on the data flow [2]. Hadoop is a system that supports parallel processing, so the data is divided into chunk sizes and it gives each node the chunk to be processed. The nodes of the map function receives the chunk and processes it to the <key, value> pairs. After normalized data is sorted on the local disk and is partitioned as a hash function, it is stored on the local disk. These outputs of the map are sent to the other nodes. Reduce task receives the outputs of each node and merges them before the

reduce function goes on. This final output is stored on the HDFS (Hadoop Distributed File System). In this case, if data exceeds the capacity of the memory in order to sort/merge the large data, external sort-merge would run and I/O would be occurred. The performance of the sort-merge has a significant impact on the total execution time of hadoop because I / O time is slower than the data processing in the memory. Figure 2 shows detailed data processing with hadoop. This is an example of finding the highest temperature for each year [2].

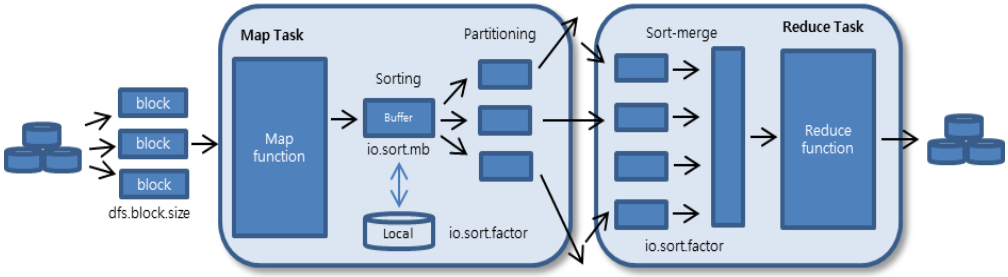



**Figure 1. Hadoop Data Flow**

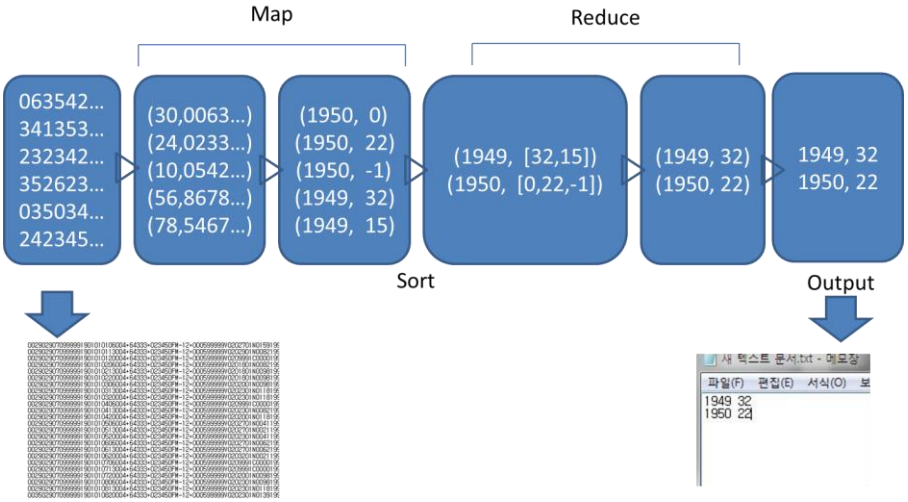



**Figure 2. Hadoop Processing**

## 3. SSD (Solid State Disk)

SSD is the next generation storage device which complements the drawbacks of the conventional magnetic storage devices. Due to the movement of the physical header, magnetic storage devices have a limit to the data processing speed. Otherwise SSD store information using a semiconductor which is very fast. SSDs' high-speed data input and output, and significantly less mechanical delay or failure. Especially, random read speed boasts excellent speed. Because of these advantages, systems which occur random access I/O seems a favorable performance. In addition, the frequent movement of the header HDD power consumption is huge but the SSD is not. Although the current price of SSDs per capacity is expensive, if used properly by calculating the cost, the overall performance of any system will be very powerful. It is an initial idea that we wonder what happens when hadoop, big data processing caused by a lot of I / O meets the SSD. So we think an SSD research can benefit from a vantage point on the random read.

## 4. Performance Test

The experiment was designed to compare the performance of HDD and SSD in the Hadoop system. Intel(R) quad-Core i5-2500 CPU 3.30GHz, 3.2GB RAM, Linux Fedora14, Hadoop 0.20.2, Sun Java 1.6u31 was used and we set up Western Digital HDD SATA 500GB and Samsung SSD 830 Series 256GB. To avoid network bottlenecks, the experiment was run in virtual distributed mode environments (the number of the node is 1).

### 4.1. *Sort* Benchmark

*Sort* benchmark is sorts the data in ascending order based on the key. First, the other hadoop benchmark *Randomwriter* produces the random binary data. And after *Sort* divides this binary data into <key,value> pairs, it's sorted. Figure 3 extracted intermediate output I / O that was read and written to the local disk when 1GB data is sorted. Read part after about 40 seconds is the sort-merge part and function part of reduce. When map output files move the reduce, it piles up Merge-Queue. The data in Merge-Queue is sort-merged and its offset is no sequential. So random access is occurred because the reduce function calls the record in the queue.

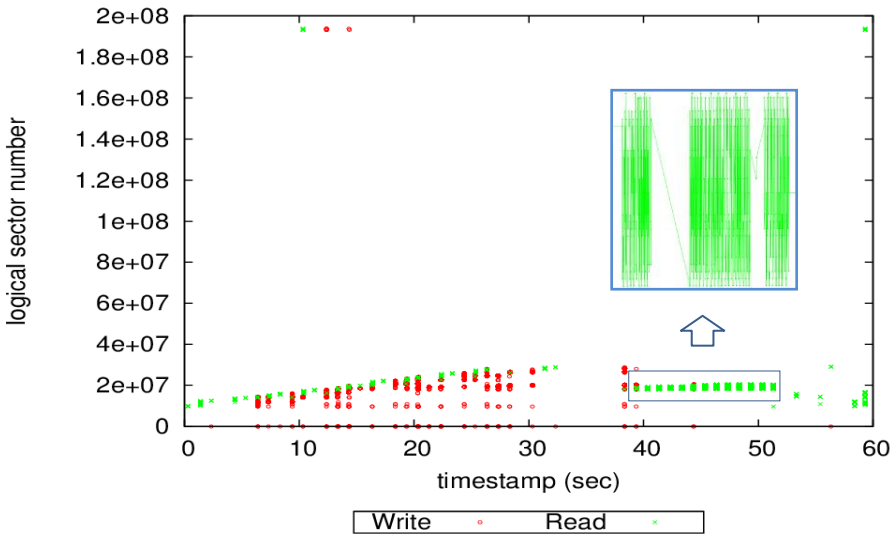



**Figure 3. Sort I/O Blocktrace (1GB)**

### 4.2. *Join* Benchmark

*Join* benchmark combines two or more data sets that are considered as duplicate records. There are inner, outer, override join in the hadoop. In this paper, the output data is combined to form a union (outer-join). Figure 4 extracted I/O to join three data sets of 1GB. These data were made by *Randomtextwriter*. *Join* merges large data sets in the map as well as reduce and a large amount of random read happens. When considering SSD's superior random read, *Join* and *Sort* benchmarks should show the superior performance on the sort-merge of reduce part. But there are not only read operations. As shown in Figure 4, the write command to write the output file to the disk

are performed in parallel. Because of these parallel I / O, the performance of SSD's random read did not take fully. Reading the page unit at a time was generally 128KB. This was worse than the unit of 4KB small random read. Figure 5 is a graph comparing the execution time of the HDD and SSD when the size of data is increased. As the data size is larger, memory caching is not affected, you can see clearly the difference in the performance of SSD and HDD. The performance of SSD was 3times better than a HDD on the *Sort*. On the *Join*, the SSD was up to 2.3 times. This point tells us that The larger size is the advantage of an SDD over a HDD. This experiment must consider the capacity of storage device. If 20GB data is sorted, 20GB data create intermediate output files on the temporary folder. And 20GB final output file is created. So a total of 60GB (20GB X 3) space is needed at least.

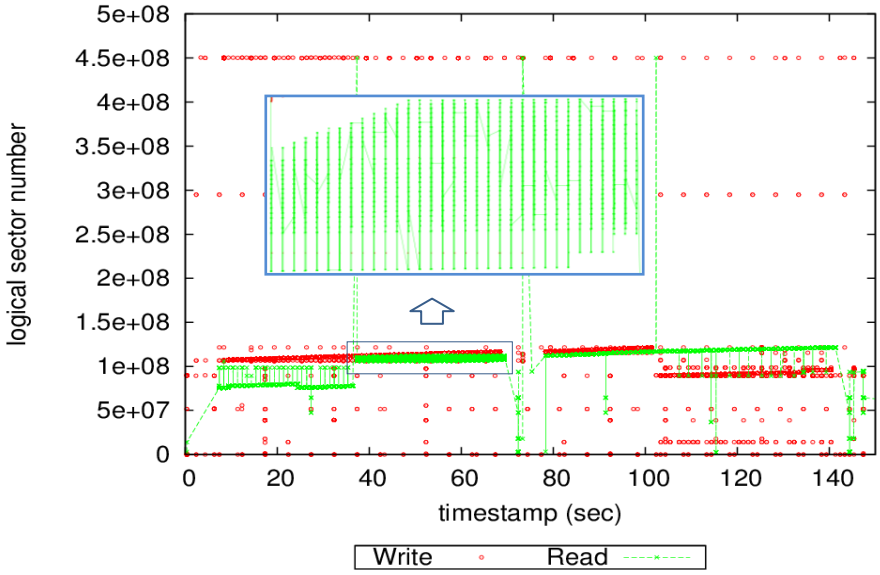



**Figure 4. Join I/O Blocktrace (1GB X 3)**

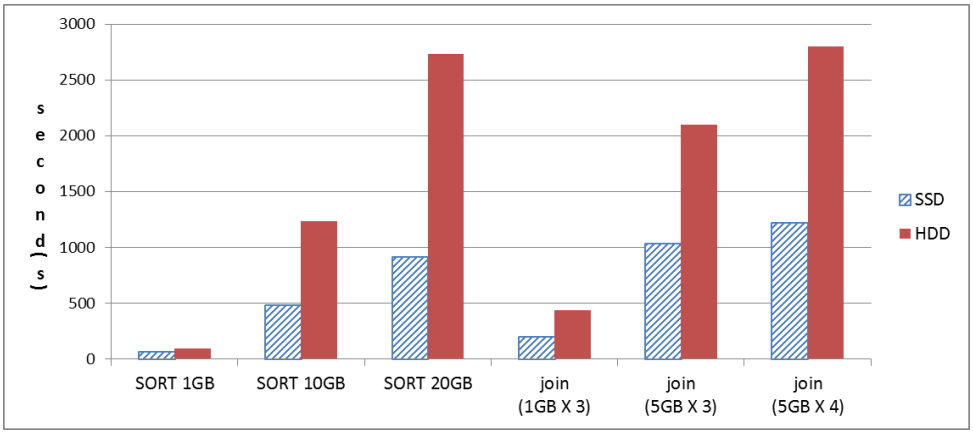



**Figure 5. Sort & Join Varing Data Size (HDD vs SSD)**

### 4.3. Optimizing Hadoop

Experiments were carried out to optimize Hadoop parameter value adjustment [8].

**Table 1. Hadoop I/O parameter**

| Parameter | Description |
|---|---|
| io.sort.mb | Sort buffer size in MAP (default = 100MB) |
| io.sort.factor | The number of streams when merging at once (default = 10) |
| dfs.block.size | The chunk(block) size (default = 64MB) |

Table 1 lists a collection of parameters that are directly relevant to I/O. io.sort.mb means the size of the buffer that is used when one block from the map task is sorted. The default value is 100MB and usually it is enough to hold the block of 64MB. But if the size of the buffer is less, unnecessary I / O happens. We should consider the capacity of actual hardware and JVM memory. io.sort.factor means the number of streams to be merged at once on the reduce task. It is applied when an external-sort is executed on the map task. If we consider the bandwidth of the storage device and set it greatly (about 200), we will have less run-time. dfs.block.size is the block size of a basic unit on hadoop. Nowadays, the trend is to optimize the set to 128MB. In map task, local sorting is that one block in the buffer is sorted. However, if the block size is bigger than the buffer size, external-sorting is occurred. After external-sorting is finished, merge starts on the local disk. This number of merge streams is io.sort.factor. Setting io.sort.factor largely is better considering the bandwidth of the storage device and a number of blocks will be merged. Figure 6 is the result of the 10GB sort and 5GB join varying io.sort.mb and dfs.block.size. This io.sort.fator was fixed at the value of 200 because the number of blocks was 161 in the *Sort* of 10GB and merging them at once was faster. The 64MB block size showed the best performance and it showed a tendency to decrease the overall execution time, as the size of the buffer becomes larger. The reason is that when the buffer size is large, in-memory sorting is executed or the number of I/O decreases. On the other side, when the block size is very small, the number of block increases. Accordingly, consumption per block overhead increases and execution time increase, too. Besides, when the block is greater than the size of the buffer, unnecessary I / O occurs and the overall performance is affected. As a result, when the buffer size is small and the block size is large, external-sorting occurs I/O and the run time increases. So, this is sensitive to adjust the buffer size and the block size compared to the HDD, the SSD was almost not changed. Although I/O are occurred, because the SSD has fast access speed and low latency, it can cover these changes and is hardly affected. When the amount of data to process is large on the node and there is a limit on the size of the buffer, the SSD has good performance.

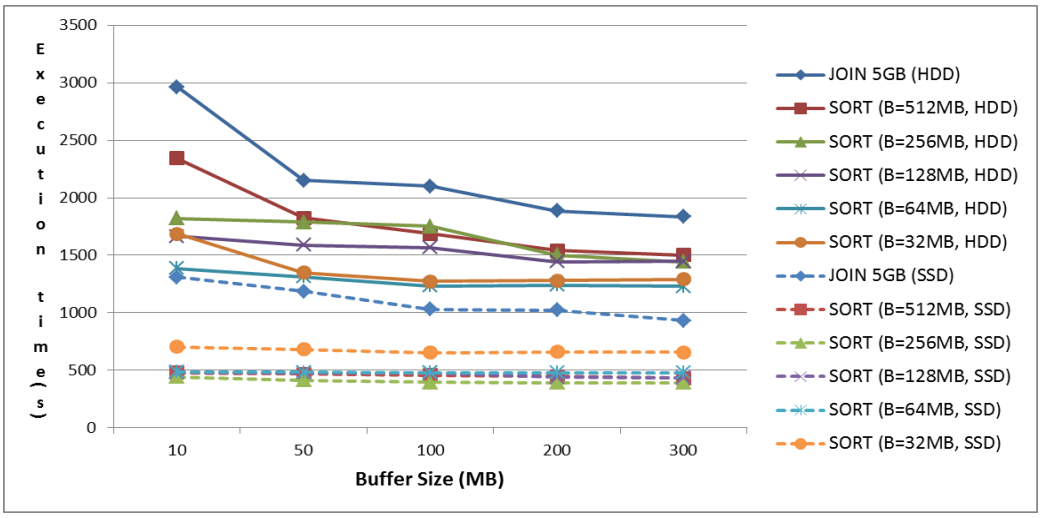



**Figure 6. Sort & Join Varing Buffer and Block Size (HDD vs SSD)**

### 4.4. Other Benchmarks

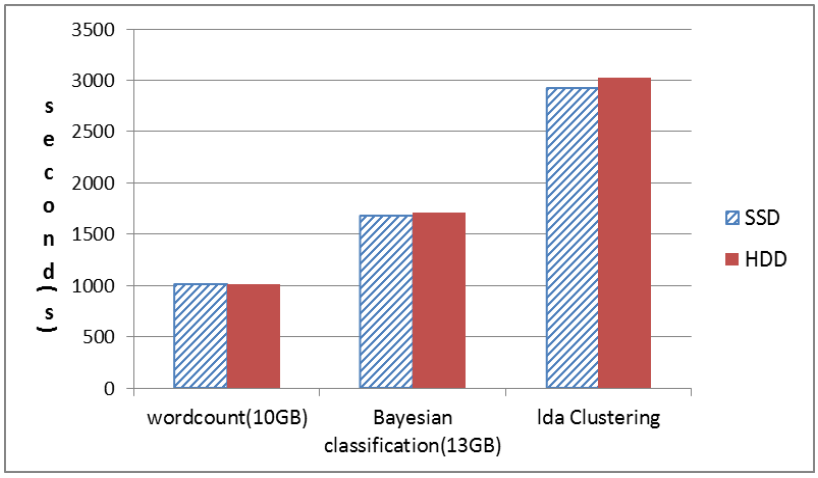



**Figure 7. Hadoop Benchmark (CPU bound)**

*Sort* and *Join* in addition to the experiments with various Hadoop benchmarks and applications were analyzed [6]. Figure 7 shows high CPU utilization, representing less amounts of I/O examples. Accordingly, there is almost no gap between the performance of the SSD and HDD. *Wordcount* which counts the number of words in the data listed is an example. This example is an example in hadoop-0.20.2-example.jar on the version of hadoop-0.20.2[10]. This was based on a 10GB text file created by *radomtextwriter*. *Wordcount* has the final result which emerges as a type of < word, the number of word > so the output file size is very small. Also, the size of map and reduce output files compared to the size of the initial input was very small, and I / O was rarely occurred. *Bayesian* and *LDA* experiments were a reference to cloud site [9]. *Bayesian classification* is a learning method using hadoop for big data based on Bayesian rule. This is a benchmark machine learning library through mahout. After the 13GB input data is generated from the article of the wiki page log, it was classified in the category as health, society, culture, and politics. It has CPU and I/O bound

but its map output file is large and I/O from reduce is small. So there is rarely read I/O. The basic example of *LDA (Linear Discriminant Analysis) Clustering* contained in the mahout is a kind of clustering as like *Bayesian* and it is only a CPU bound example, too. Figure 8 shows high I/O utilization examples. *DFSIO* is a test to extract the hadoop I/O performance and throughput. This is contained in hadoop-0.20.2-test.jar and when an I/O bottleneck is tested and I/O throughput is measured, it is necessary. DFSIO write 10GB data and read 10GB. It makes the gap between the HDD and SSD. Data grouping to find a specific word in a *Grep* and *Randomwriter* to produce a random binary data have a lot of I/O as *Sort* and *Join*. *Randomwriter* make I/O as expected because of the amount of random binary data was written. *Grep* uses the data generated from *randomtextwriter*, and the SSD's performance was highlighted because finding certain letters occur by random access. These examples such as *Grep*, *Join*, *randomwriter* and *Sort* are the basic examples in hadoop-0.20.2-example.jar. They both have that a lot of random read in the sort-merge part in common. As random read performance, SSD showed an outstanding performance in these examples.

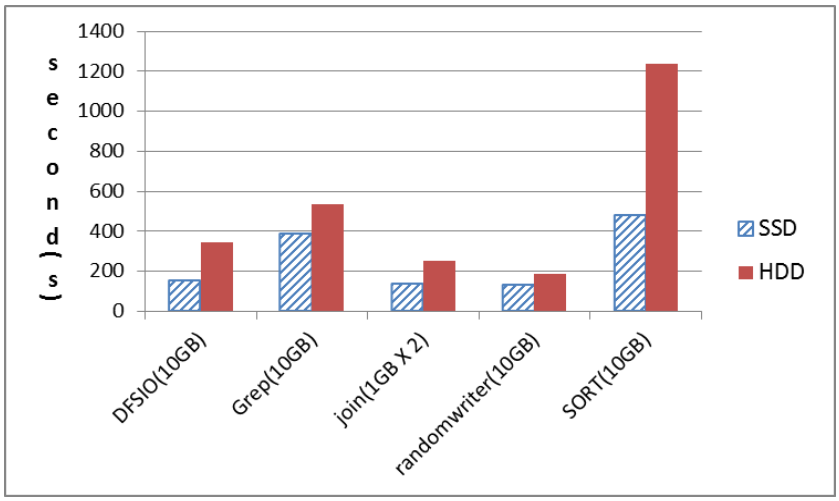



**Figure 8. Hadoop Benchmark (I/O bound)**

### 4.5. Electricity Consumption

We measured the electricity consumption of a PC using HDD and SSD. We used the wattmeter (HPM-100A) product. This was installed in the middle of PC in a plug & play manner. The electricity used in benchmarking is shown in the following table.

**Table 2. Hadoop Electricity Consumption**

| Benchmark(execution time) | HDD | SSD |
|---|---|---|
| *SORT* 10GB | 18.49Wh(1235s) | 9.13Wh(479s) |
| *JOIN* 5GB X 3 | 31.85Wh(1768s) | 15.74Wh(1030) |

Because of the less-runtime, the electricity consumption of the SSD was 2 times less than the HDD. Companies which use the Hadoop (like as Yahoo, Facebook, Google) have many nodes about some of thousands. If these nodes which are consist of the SSD sort (or join) or

the data of petabyte, they will be able to save a large amount of power. When 1000 nodes mounted on SSD are more expensive about $ 180 than HDDs, sorting more than 2000 times turn can take on the cost advantages of the SSD considering electricity fees. Present SSD's high cost does not fit on the big data processing but we expect this device to be suitable if we find ways to reduce the price and time.

## 5. Conclusion

During the hadoop test, I / O bottleneck problems showed a better performance by using a SSD. Hadoop's each node replicates with the pipeline method, because they must preserve data recovery. So this is basically a system which has many I/O. In addition, because the system is dealing with big data, the importance of the storage device performance impact is enormous. But that causes the patterns of I / O which are quite different depending on how it is being used and what analyzes the data. Conclusively, the impact of SSD on hadoop may be less. However, the size of the data is increasing, the bottlenecks are coming from the storage device is increasing, too. The SSD is clearly showing the strengths on I/O bottleneck. Big data sort or requiring a task to combine a big data, big data analysis to derive a tremendous amount of work, and finding a specific record in the Big Data as *Grep*, are the tasks that can exert a large force. In a recent study, hadoop was found to have the biggest bottleneck in the network portion [4, 7]. Considering the rapidly evolving network transmission technology, when the network bottleneck is reduced, the SSD will help to remove the I/O bottleneck of hadoop. In the future work, we will minimize the CPU and network bottlenecks to analyze the impact of SSD on hadoop in a real cluster environment. The number of nodes for analyzing the impact of SSD is worthy to notice, too. It will be more clearly applied on the cost advantages of SSDs.

## Acknowledgements

This work was supported in part by MKE, Korea under ITRC NIPA-2010-(C1090-1021-0008)(NTIS-2010-(1415109527)) and the IT R&D program of MKE/KEIT (10041244, SmartTV 2.0 Software Platform).

## References

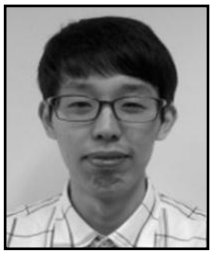
[1] B. Li, E. Mazur, Y. Diao, A. McGregor and P. Shenoy, (Eds.), "A Platform for Scalable One-Pass Analytics using MapReduce", Proceedings of the SIGMOD'11, **(2011)** June12-16; Athens, Greece.
[2] T. White, "Hadoop The Definitive Guide Second Edition", Editor, O'Reilly Media **(2010)**.
[3] S. -W. Lee, B. Moon, C. Park, J. -M. Kim, S. -W. Kim, (Eds.), "A Case for Flash Memory SSD in Enter-prise Database Applications", Proceedings of the SIGMOD'08, **(2008)** June12-16; Vancouver, BC, Canada.
[4] S. Sur, H. Wang, J. Huang, X. Ouyang and D. K. Panda, (Eds.), "Can High-Performance Interconnects Benefit Hadoop Distributed File System?", Proceedings of the MASVDC Workshop 2010, **(2010)** December 5; Atlanta, GA, USA.
[5] J. Dean and S. Ghemawat, (Eds.), "MapReduce: Simplified Data Processing on Large Clusters", Proceedings of the OSDI 2004, **(2004)** December 6-8; San Francisco, CA.
[6] S. Huang, J. Huang, J. Dai, T. Xie and B. Huang, (Eds.), "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis", Proceedings of the ICDE Workshops 2010, **(2010)** March 1-6, Long Beach, California, USA.
[7] J. Dai, (Eds.), "Toward Efficient Provisioning and Performance Tuning for Hadoop", Proceedings of the Apache Asia Roadshow 2010, **(2010)** August 14-15, Shanghai, China.

[8] S. Babu, "Towards Automatic Optimization of MapReduce Programs", Communications of the ACM - 50th anniversary, vol. 51, Issue 1, **(2008)** January.
[9] http://parsa.epfl.ch/cloudsuite/analytics.html.
[10] http://hadoop.apache.org/.
[11] J. Schmid, supervised by Prof. Dr. M. Böhlen and A. Dignös, "Implementation and Evaluation of a Key-Value Store for Flash-based Storage", (2012) Februrary 3.
[12] http://wiki.apache.org/hadoop/.

## Authors

**Seok-Hoon Kang**

He received the bachelor's degree in computer engineering from Sungkyunkwan University, Korea, in 2009. His research interest is in flash-based database technology, clouding system, and distributed systems. Currently, he is a master student of computer engineering at Sungkyunkwan University, Korea.



**Dong-Hyun Koo**

He received the bachelor's degree in computer engineering from Sungkyunkwan University, Korea, in 2012. His research interest is in flash-based database technology, mobile platform DBMS, and distributed systems. Currently, he is a master student of computer engineering at Sungkyunkwan University, Korea.

**Woon-Hak Kang**

He received the master's degree in computer engineering from Sungkyunkwan University, Korea, in 2010. His research interest is in flash-based database technology, mobile platform DBMS, and flash file systems. Currently, he is a PhD student of computer engineering at Sungkyunkwan University, Korea.

**Sang-Won Lee**

He received the PhD degree from the Computer Science Department of Seoul National University in 1999. He is an associate professor with the School of Information and Communication Engineering at Sungkyunkwan University, Suwon, Korea. Before that, he was a research professor at Ewha Women University and a technical staff at Oracle, Korea. His research interest includes flash-based database technology.