JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.15, NO.5, OCTOBER, 2015 ISSN(Print) 1598-1657
http://dx.doi.org/10.5573/JSTS.2015.15.5.511 ISSN(Online) 2233-4866

An Analytical Approach to Evaluation of SSD Effects under MapReduce Workloads

Sungyong Ahn and Sangkyu Park

***Abstract*—As the cost-per-byte of SSDs dramatically decreases, the introduction of SSDs to Hadoop becomes an attractive choice for high performance data processing. In this paper the cost-per-performance of SSD-based Hadoop cluster (SSD-Hadoop) and HDD-based Hadoop cluster (HDD-Hadoop) are evaluated. For this, we propose a MapReduce performance model using queuing network to simulate the execution time of MapReduce job with varying cluster size. To achieve an accurate model, the execution time distribution of MapReduce job is carefully profiled. The developed model can precisely predict the execution time of MapReduce jobs with less than 7% difference for most cases. It is also found that SSD-Hadoop is 20% more cost efficient than HDD-Hadoop because SSD-Hadoop needs a smaller number of nodes than HDD-Hadoop to achieve a comparable performance, according to the results of simulation with varying the number of cluster nodes.**

***Index Terms*—MapReduce, Hadoop, performance modeling, SSDs, cost-per-performance**

I. INTRODUCTION

MapReduce [1] is a programing model which was suggested for large data parallel processing by Google. Apache Hadoop [3, 4] is an open source implementation of MapReduce and Google File System (GFS) [2]. Recently, Hadoop was become a de facto standard in the area of BigData analytics.

As the cost-per-byte of SSDs dramatically decreases and high performance data processing is increasingly important, the introduction of SSDs to Hadoop becomes an attractive choice [5, 6]. SSDs have much higher I/O performance than HDDs. As a result, SSDs can improve the performance of Hadoop cluster by removing the I/O bottleneck caused by low I/O performance of HDDs. However, the cost-per-byte of SSDs is still higher than that of HDDs.

The previous researches compared the cost-per-performance of SSDs with that of HDDs to highlight the high I/O performance of SSDs. Moon et al [5] evaluated different storage configurations using Hadoop benchmark. The results indicated that SSDs are suitable for Hadoop as intermediate data storage to increase the cost efficiency. Kambalta et al [6] compared the cost-per-performance of SSDs and HDDs under equal-bandwidth constraints. However, they concerned only the cost efficiency of Hadoop clusters having same number of nodes, regardless of the performance of data processing.

The performance of Hadoop cluster can be improved not only by using SSDs in place of HDDs but also by increasing the number of nodes. Therefore, it is needed to compare the cost efficiency of SSD-based Hadoop cluster (SSD-Hadoop) and HDD-based Hadoop cluster (HDD-Hadoop), displaying equal throughput, regardless of the number of nodes in the cluster. In this paper we propose a novel MapReduce performance model using queuing network to predict the execution time of MapReduce jobs. The simulation and experiment results reveal that the proposed performance model can predict the execution time of MapReduce jobs with less than

Manuscript received Apr. 17, 2015; accepted Jun. 8, 2015
DS Software R&D Center, Samsung Electronics Co., Ltd.
E-mail : {sungyong.ahn, skyu7.park}@samsung.com

about 7% difference for most cases. Using the performance model, the execution time of MapReduce jobs with varying number of cluster nodes is also investigated. It reveals that HDD-Hadoop needs 1.4 times more Data nodes than SSD-Hadoop to achieve the same performance.

The remainder of this paper is organized as follows. Section II introduces the related works. Section III describes MapReduce workloads, and Section IV proposes the MapReduce queuing network model. Section V validates the proposed model and compares the cost efficiency of HDD-Hadoop and SSD-Hadoop cluster. The conclusion is given in Section VI.

II. Related Work

There exist numerous studies evaluating the performance implication of SSDs in MapReduce workloads. Moon et al [5] compared the cost efficiency of various storage configurations using TeraSort, a representative Hadoop benchmark. Their study reveals that using SSDs as intermediate data storage of Hadoop is the most cost efficient solution because intermediate data of Hadoop are accessed mainly by random I/O request. Using SSDs for storing the intermediate data of Hadoop increases the cost efficiency by 15%, in comparison to HDD-only Hadoop cluster. Kambalta et al [6] compared the cost-per-performance of HDDs and SSDs, both of which have a same aggregated bandwidth, not capacity. According to their results of experiment, SSD-Hadoop is 2.5 times more expensive in terms of cost-per-performance, while achieving up to 70% higher performance compared to HDD-Hadoop. The previous studies compared the cost-per-performance of HDD-Hadoop and SSD-Hadoop under the constraint of equivalent cluster size. However, to get more practical insight, the cost efficiency should be estimated with the Hadoop clusters displaying the same bandwidth of data processing, which it reveals the cost efficient way for building Hadoop cluster allowing the required performance. We investigate this problem using the performance model of MapReduce.

Building an effective performance model of MapReduce is difficult because MapReduce jobs are processed in the distributed and parallel fashion. There are previous studies proposing the performance models and performance optimization method for MapReduce. Yang suggested optimal number of Map tasks and Reduce tasks using their own performance model in [7]. Here it can be applicable only to restricted input size. Krevat et al [11] proposed an analytical performance model of MapReduce job displaying optimal performance of Hadoop cluster. The model does not predict the execution time of MapReduce jobs.

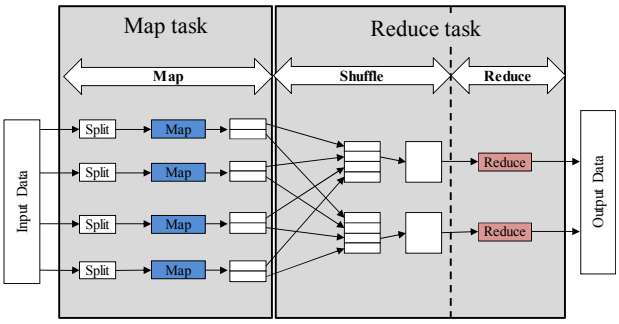

Fig. 1. The dataflow of MapReduce

III. MapReduce Operation

1. Overview of MapReduce

Fig. 1 briefly describes the dataflow of MapReduce jobs. The MapReduce jobs are processed in two phases: *Map* and *Reduce*. Each phase is divided into multiple tasks, which are performed in parallel among multiple nodes.

First, Map phase is divided into multiple Map tasks, each of which is in charge of a part of input data, called '*Input split*'. They are all identical sizes. As the size of input data increases, the number of Map tasks also linearly increases because the size of *Input split* is fixed and the number of *Input splits* is same as the number of Map tasks. Each Map task performs user-defined Map function for its own *Input split*, and then writes the results at local disk. Note that the output of Map tasks is called '*Intermediate data'*, which is transferred to Reduce tasks.

Reduce phase starts only after all Map tasks are complete. Like the Map phase, Reduce phase is divided into multiple Reduce tasks. At first, Reduce tasks copy the required *Intermediate data* from remote nodes. Following the operation called shuffling, the collected chunks of *Intermediate data* are merged. Note that

Table 1. Parameters of MapReduce framework

Category	Symbol	Definition
System parameters	S_m	The number of Map slots
	S_r	The number of Reduce slots
	N	The number of cluster nodes
	B_r, B_w	The read/write bandwidth of storage
	B_n	The network bandwidth
	HB	The size of HDFS block
Workload parameters	I_{Total}	The size of total input data
	I_{Split}	The size of *Input split* (= *HB*)
	N_m	The number of Map tasks (= $\frac{I_{Total}}{I_{Split}}$)
	N_r	The number of Reduce tasks
	$f()$	The Map function
	$g()$	The Reduce function
	R_m	The ratio between input and output of Map function
	R_r	The ratio between input and output of Reduce function

shuffled and merged *Intermediate data* are input to the user-defined Reduce function. Finally, the output of Reduce function is written to the HDFS. In this paper, we propose to split a Reduce task into two sub-tasks: *Shuffle* sub-task and *Reduce* sub-task because they have different type of workloads. A Shuffle sub-task generates massive network traffic for copying *Intermediate data* from remote nodes. On the other hand, Reduce sub-task performs user-defined Reduce function and writes final output to the HDFS. Therefore, I/O resource is the mainly consumed resource. Because it is difficult to analyze a system performing various operations as a single unit, Reduce task is split into two sub-tasks for more precise and tractable modeling.

2. Execution Time of MapReduce Job

In this paper, execution time of MapReduce jobs is employed as the performance metric of Hadoop. It is affected by various system parameters and workload parameters are listed in Table 1. The system parameters define the configuration of Hadoop cluster, while the workload parameters do the characteristic of MapReduce jobs. Here execution time of MapReduce jobs is analyzed.

First of all, Map phase consists of multiple iterations of Map tasks, while each iteration includes the same number of Map tasks, equivalent to the number of Map slots. Therefore, execution time of Map phase (T_{MP}) is as follows.

$$T_{MP} = \left\lceil \frac{N_m}{S_m} \right\rceil \times T_m .$$

Execution time of a Map task is comprised of reading *Input split*, processing Map function, and writing the Map output. Supposing linear function for Map function, execution time of Map tasks (T_m) is determined by the size of *Input split* (I_{Split}). Note that the size of *Input split* is determined according to the size of block of HDFS.

$$T_m = \frac{I_{Split}}{B_r} + f(I_{Split}) + \frac{I_{Split} \times R_m}{B_w} .$$

As mentioned above, Reduce phase is divided into Shuffle sub-task and Reduce sub-task, which are executed sequentially. Recall that the number of Reduce task need to be equal to the number of Reduce slot for optimized performance [7]. In other words, all Reduce tasks are processed simultaneously. Therefore, execution time of Reduce phase (T_{RP}) is the longest execution time of Reduce tasks as follows.

$$T_{RP} = Max(T_s + T_r) .$$

A Shuffle sub-task copies the required parts of *Intermediate data* from Map tasks to Reduce tasks. Hence execution time of Shuffle sub-task (T_s) is proportional to the size of Input data (I_{Total}).

$$T_s = \frac{I_{Total} \times R_m}{N_r \times B_n} .$$

A Reduce sub-task processes the Reduce function, and then writes the final output to the HDFS. Supposing linear Reduce function, execution time of Reduce sub-task (T_r) increases linearly in proportion to the size of Input data (I_{Total}).

$$T_r = g\left(\frac{I_{Total} \times R_m}{N_r} \right) + \frac{I_{Total} \times R_m \times R_r}{N_r \times B_w} .$$

Since the Shuffle and Reduce sub-task are proportional to the size of Input data (I_{Total}), linear regression technique can be used to find the average service demand

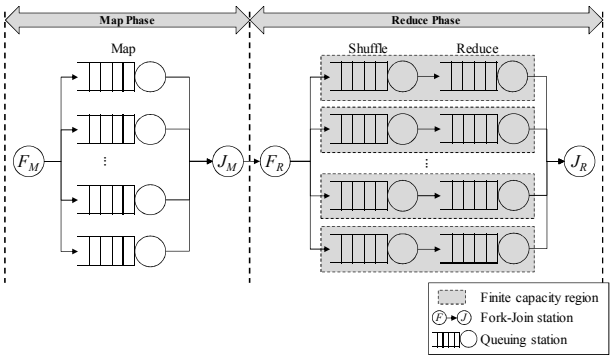

Fig. 2. The queuing network model of MapReduce

of each sub-task.

IV. MODELING OF MAPREDUCE

In this section a new performance model for MapReduce is proposed which can predict the execution time of MapReduce job using queuing network. A queuing network model is useful for system modeling because of its simplicity and flexibility. In addition, the well formulated queuing network theory can be adopted to analyze the target system.

1. Modeling Strategy

Fig. 2 shows the queuing network describing the MapReduce operation. Observe from the figure that the MapReduce operation is handled in two phases, Map and Reduce, while the Reduce phase consists of Shuffle stage and Reduce stage.

A. Map tasks

In the MapReduce framework, the maximum number of Map tasks processed simultaneously is limited by the number of Map slots. Each Map task is assigned to available Map slot. However, if there is no available Map slot, Map tasks should be delayed until Map slot becomes available. Therefore, each Map slot can be represented as a queuing station (*Map station*) in Fig. 2.

B. Fork-Join stations

The Fork-Join station of queuing network is used to describe the parallel processing of MapReduce job. In the Fork station (F_M), MapReduce job is divided into multiple Map tasks which are distributed to multiple Map stations. The MapReduce job is queued in the Join station (J_M) until all Map tasks are completed. Note that the Fork-Join mechanism is very suitable to model the parallelism of MapReduce operation. Likewise, Reduce phase is also described with Fork-Join stations (F_R - J_R)

Table 2. The hardware and software configuration of a cluster node.

Component	Description
Processor	Intel Xeon E5-2670 (8 cores) 2.6 GHz * 2
Memory	1600MHz DDR3 16GB * 16 = 256 GB
HDD	WD VelociRaptor 600 GB (SATA3/10 Krpm/32M)
SSD	Samsung SSD840 512 GB (SATA3)
Network	10 Gigabit Ethernet NIC
OS	Ubuntu 12.04 LTS
Hadoop	Cloudera CDH 4.7.1

C. Shuffle and Reduce sub-tasks

When all Map tasks are finished, the Reduce phase begins. Similar to Map phase, Reduce tasks are distributed to multiple Reduce slots in Fork station (F_R) and merged in Join station (J_R). Recall that a Reduce task is partitioned into two sub-tasks, Shuffle sub-task and Reduce sub-task, which are performed sequentially. Therefore, each Reduce slot is described as two type of queuing stations; *Shuffle station* and *Reduce station* which take charge of Shuffle sub-task and Reduce sub-task, respectively. Then, the Shuffle station and Reduce station are grouped into a Finite Capacity Region in Fig. 2. In the Finite Capacity Region, the aggregated number of shuffle sub-tasks and Reduce sub-tasks is bound by the number of Reduce slots.

As described above, the MapReduce operation can be effectively represented with queuing network. In the next section, the results of an experiment are introduced to determine the distribution of service time of each queuing station.

2. Experiment Setup

To profile the execution time of each task, a Hadoop cluster consisting of one *Name node* and eight *Data nodes* is built. Here all cluster nodes are identically configured as summarized in Table 2. For the convenience of installation and management, CDH 4.7.1 (Cloudera Distributed Hadoop) [8] is employed. For

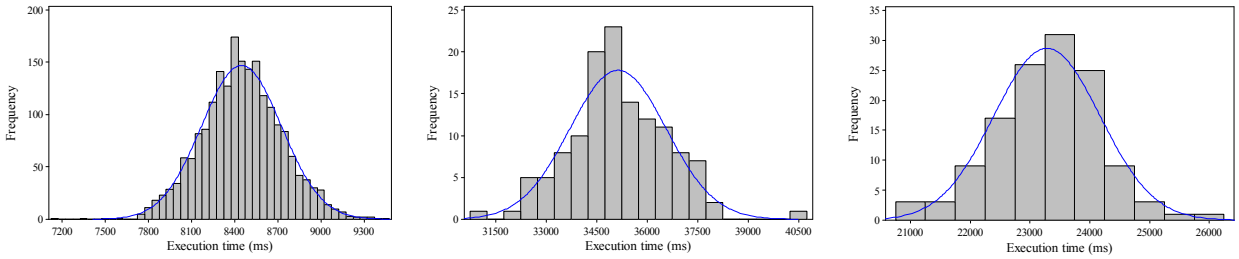

Fig. 3. The distribution of task execution time (a) Map task, (b) Shuffle sub-task, (c) Reduce sub-task

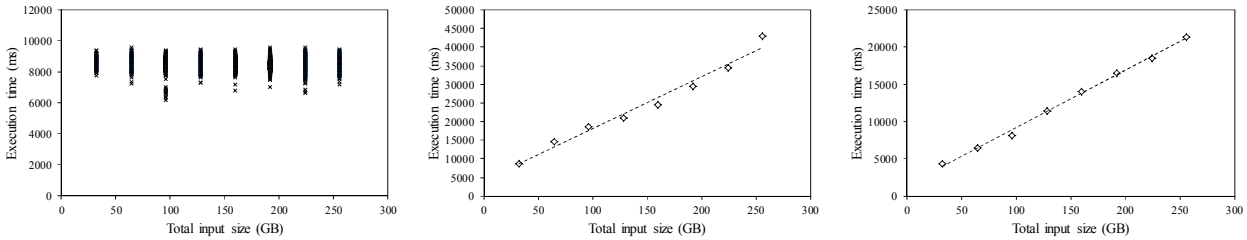

Fig. 4. Task execution time with varying input size (SSD-Hadoop) (a) Map task, (b) Shuffle sub-task, (c) Reduce sub-task

profiling, execution time of TeraSort [9], a representative Hadoop benchmark, is measured with varying size of input data. TeraSort is widely used to evaluate the performance of Hadoop cluster because it requires both of CPU and I/O resources, while sorting is a main task of MapReduce job [11]. The number of Map slots and Reduce slots of each cluster node is configured as 16, identical to the number of CPU core of each cluster node. The size of input data varies from 32 GB to 256 GB, while the size of *Input split* is 128 MB.

3. Distribution of Service Time

The execution time distribution of each task is a key factor in configuring the queuing network for MapReduce modeling. The previous researches suppose that it is deterministic. However, according to our observation, it is widely varies. Fig. 3 shows the execution time distribution of each type of task. As shown in Fig. 3(a), execution time of Map task follows normal distribution, not deterministic. Fig. 3(b) and (c) are for the Shuffle sub-task and Reduce sub-task, respectively. As seen from the figures, they also show normal distribution. Therefore, normal distribution is assumed as the distribution of service time of each queuing station.

4. Average Service Time

In this subsection average service time of each queuing station is obtained through profiling. Fig. 4(a) shows the execution time of Map tasks with varying size of input data. As mentioned earlier, the execution time of Map task is determined by the size of *Input split* (I_{Split}), not by the size of Input data (I_{Total}). Therefore, the average service time of Map station is obtained by averaging all measured execution time of Map tasks. Fig. 4(b) and (c) show average execution time of Shuffle sub-task and Reduce sub-task, respectively. Unlike Map task, the average execution time of them increases linearly in proportion to the size of input data. Therefore, the average service time of Shuffle and Reduce stations can be decided using the linear regression technique. Fig. 5 shows the average execution time of Shuffle and Reduce sub-task in HDD-Hadoop cluster with varying size input data. Like SSD-Hadoop, the average execution time of each sub-task increases linearly. The linear regression is thus applicable to the HDD-Hadoop. Table 3 summarizes the average and standard deviation of service time of Map, Shuffle and Reduce stations, respectively. Finally, the distribution of service time of each queuing station on MapReduce queuing network is determined.

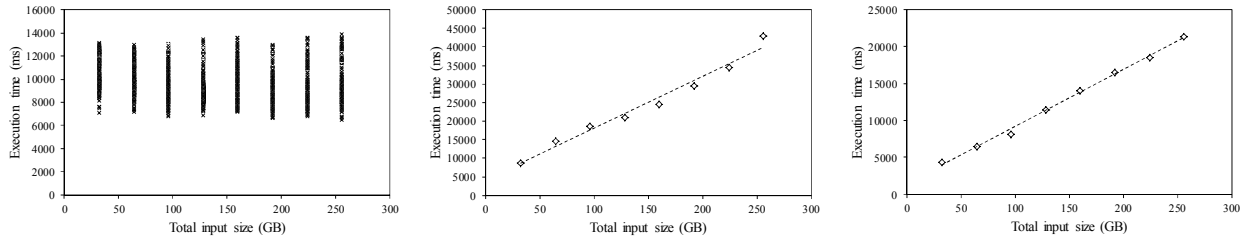

Fig. 5. Task execution time with varying input size (HDD-Hadoop) (a) Map task, (b) Shuffle sub-task, (c) Reduce sub-task

Table 3. The service time distribution of each type of queuing station

	Map		Shuffle		Reduce	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
SSD-Hadoop	8473	290	1973	139	1319	58
HDD-Hadoop	10962	2599	13898	5240	1026	74

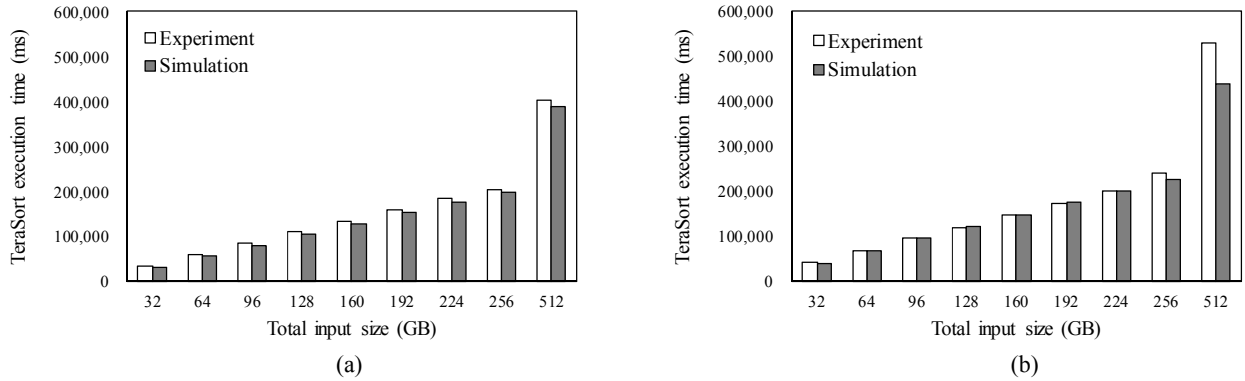

Fig. 6. The execution time of TeraSort with varying size input data on (a) SSD-Hadoop, (b) HDD-Hadoop

V. VALIDATION AND ANALYSIS

The effectiveness of the MapReduce queuing network presented in the previous section is verified by comparing the simulation results with the measured execution time. In addition, the number of nodes required by HDD-Hadoop to deliver an equivalent performance to SSD-Hadoop is estimated using the developed model. It reveals that SSD is more cost effective than HDD as the storage media of Hadoop cluster.

1. Validation of Performance Model

The execution time of TeraSort is simulated using JMT [10], a queuing network simulator.

Fig. 6(a) shows the result of simulation on SSD-Hadoop. As seen in the figure, the simulated results are very close to those of experiment regardless of the size of input data. The maximum difference is less than 7%.

The execution times for HDD-Hadoop are shown in Fig. 6(b). Like the results for SSD-Hadoop, the simulation results are also close to those of experiment on HDD-Hadoop. For small input sizes (32 GB~256 GB), the difference is less than about 6%. Notice from Fig. 6(b) that the difference for large input data of 512GB is 17.6%. This is because the fluctuation of execution time in HDD-Hadoop is larger than SSD-Hadoop due to mechanical movement of HDDs.

The performance of Hadoop cluster is significantly influenced by the number of nodes organizing the Hadoop cluster. Therefore, the proposed performance model of MapReduce is verified with varying number of Data nodes. Note that there is only a single Name node in all the test cases. Fig. 7(a) and (b) compare the simulation results with experiment results with varying number of Data nodes. As seen in the figures, the

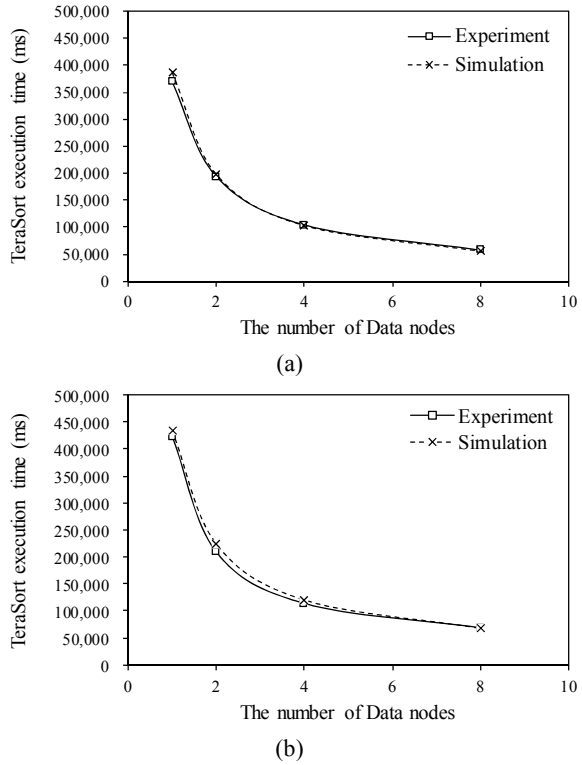

Fig. 7. The execution time of TeraSort with varying number of Data nodes (a) SSD-Hadoop, (b) HDD-Hadoop

proposed performance model can precisely predict the execution time of TeraSort for the Hadoop cluster consisting of different number of nodes. Observe that the maximum difference is about 5% and 7% for SSD-Hadoop and HDD-Hadoop, respectively.

2. Cost Efficiency

The cost-per-performance is the metric commonly used to compare the cost efficiency of SSD-Hadoop and HDD-Hadoop in the previous researches. However, the influence of the number of nodes on the performance of Hadoop cluster was not considered. In this section, using the proposed performance model, the number of Data nodes required for HDD-Hadoop to allow equivalent performance as SSD-Hadoop is estimated.

We first obtain the execution time of TeraSort on HDD-Hadoop with increasing number of Data nodes as shown in Fig. 8. Here, the size of input data is 64 GB, and SSD-8 denotes the execution time of SSD-Hadoop consisting of 8 Data nodes. The simulation results indicate that HDD-Hadoop requires 11 Data nodes to achieve an equivalent performance as SSD-8.

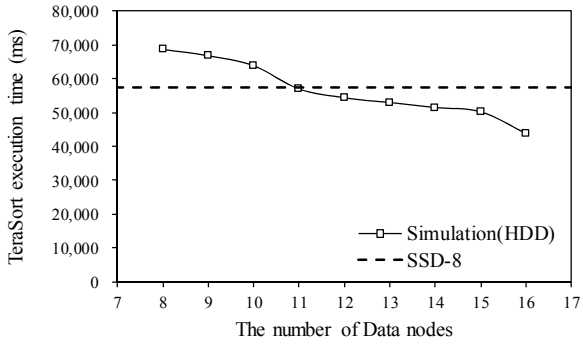

Fig. 8. The execution time of TeraSort with varying number of Data nodes on Hadoop cluster

Table 4. Price of Hardware Components Constituting Hadoop Cluster Node (at Amazon)

Component	Description	Price($)
Motherboard	SuperMicro X9DR7-LN4F	3,150
Processor	Intel Xeon E5-2600 x2	558
Memory	DDR3 ECC 16GB (256 GB) x16	2,448
NIC	Intel X-540 T2 10 GbE	439
HDD	WD VelociRaptor 600 GB x4	576
SSD	Samsung SSD840 512 GB x4	1,292

Table 5. Cost-per-Performance (SSD-Hadoop vs. HDD-Hadoop)

Setup	Required Num. of Nodes	Total Cost ($)	Cost per Performance
SSD-Hadoop	8	63,096	0.8
HDD-Hadoop	11	78,881	1.0

Table 4 describes the price of hardware components each cluster node has. Based on that, the cost for building an SSD-Hadoop and HDD-Hadoop displaying an equivalent performance are compared. As mentioned above, the HDD-Hadoop cluster needs 3 more Data nodes to achieve a same performance as SSD-8. As a result, SSD-Hadoop is 20% more cost efficient than HDD-Hadoop as shown in Table 5. This observation indicates that adoption of SSDs is a more cost efficient solution than increasing the number of nodes to build Hadoop clusters. Moreover, small size Hadoop cluster allows Reducing server management cost including the power, space and cooling as well as the system building cost.

VI. CONCLUSIONS

In this paper MapReduce performance model has been proposed to predict the execution time of MapReduce job

using queuing network. In addition, the cost efficiency of SSDs and HDDs are compared as the storage media of Hadoop clusters. The simulation and experiment reveal that the difference is up to 6.8% except for HDD-Hadoop of large input data (17.6%). Moreover, the proposed model is accurate even with varying number of Data nodes. The simulation results show that HDD-Hadoop needs 1.4 times more Data nodes than SSD-Hadoop to achieve the same performance. Consequently, SSD-Hadoop can said to be more cost efficient than HDD-Hadoop.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Operating Systems Design and Implementation, 2004, OSDI 2004, 6th Symposium on*, *Dec., 2004.*

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *Symposium on Operating systems principles, 2003. SOSP 2003, 19th ACM symposium on*, pp. 29-43, Dec., 2003.

[3] Apache Hadoop Project, http://hadoop.apache.org

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *Mass Storage Systems and Technologies, 2010, MSST 2010, IEEE 26th Symposium on*, May, 2010.

[5] S. Moon, J. Lee, and Y. Kee, "Introducing SSDs to the Hadoop MapReduce Framework," *Cloud Computing, 2014, CLOUD 2014, IEEE 7th International Conference on*, July, 2014.

[6] K. Kambatla, and Y. Chen, "The Truth About MapReduce Performance on SSDs," *Large Installation System Administration, 2014, LISA 2014, 28th USENIX conference on*, pp. 109-117, Nov., 2014.

[7] X. Yang and J. Sun, "An analytical performance model of MapReduce," *Cloud Computing and Intelligence Systems, 2011, CCIS 2011, IEEE International Conference on*, pp. 306-310, Sept., 2011.

[8] Cloudera Inc., CDH (Cloudera Distributed Hadoop), http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html

[9] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis," *Data Engineering Workshops, 2010, ICDEW 2010, IEEE 26th International Conference on*, pp. 41-51, Mar., 2010.

[10] JMT (Java Modeling Tools), http://jmt.sourceforge.net/

[11] E. Krevat, T. Shiran, E. Anderson, J. Tucek, J. J. Wylie, and G. R. Ganger, "Understanding Inefficiencies in Data-Intensive Computing," Carnegie Mellon University Technical Report, Jan., 2012.

Sungyong Ahn received the B.S., Ph.D. degree in the Department of Computer Science and Engineering from Seoul National University, Korea, in 2003 and 2012, respectively. He has been a Senior Engineer at Samsung Electronics since 2012. His interests include operating system, solid-state disk, and Bigdata.

Sangkyu Park received the B.S., M.S. degrees in the Department of Electrical Engineering from University of Seoul, Seoul, Korea, in 1998 and 2000 respectively. In 2012, he joined at Samsung Electronics, where he has been working in the area of flash device SW. His interests include flash storage and middleware for Big Data.