An Empirical Performance Analysis on Hadoop via Optimizing the Network Heartbeat Period

Jaehwan Lee[1], June Choi[2], Hongchan Roh[3] and Ji Sun Shin[4*]
[1] School of Electronics and Information Engineering, Korea Aerospace University
[e-mail: jlee@kau.ac.kr]
[2] School of Electronics and Information Engineering, Korea Aerospace University
[e-mail: wnsdl1210@gmail.com]
[3] SK Telecom
[e-mail: hongchan.roh@sk.com]
[4] Department of Computer and Information Security, Sejong University
[e-mail: jsshin@sejong.ac.kr]
*Corresponding author: Ji Sun Shin

Received March 21, 2018; revised May 8, 2018; accepted June 21, 2018; published November 30, 2018

Abstract

To support a large-scale Hadoop cluster, Hadoop heartbeat messages are designed to deliver the significant messages, including task scheduling and completion messages, via piggybacking to reduce the number of messages received by the NameNode. Although Hadoop is designed and optimized for high-throughput computing via batch processing, the real-time processing of large amounts of data in Hadoop is increasingly important. This paper evaluates Hadoop's performance and costs when the heartbeat period is controlled to support latency sensitive applications. Through an empirical study based on Hadoop 2.0 (YARN) [1] architecture, we improve Hadoop's I/O performance as well as application performance by up to 13 percent compared to the default configuration. We offer a guideline that predicts the performance, costs and limitations of the total system by controlling the heartbeat period using simple equations. We show that Hive performance can be improved by tuning Hadoop's heartbeat periods through extensive experiments.

Keywords: Hadoop, Heartbeat, Hadoop Ecosystem, Hive, TPC-H, Terasort Benchmark

This work is supported by a grant of the SKT-KAU Big Data R&D Collaboration Program through the SK Telecom Corporate R&D Center funded by SK Telecom under Grant 4160035945 and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT(No. 2015R1C1A1A02036524).

1. Introduction

Hadoop is one of the most popular distributed processing frameworks for big data. A Hadoop cluster is typically composed of tens or hundreds of machines that process large amounts of data. To support a large-scale cluster, Hadoop heartbeat messages are designed to deliver significant messages, including task scheduling and completion messages, via piggybacking to reduce the number of messages received by a NameNode. Hadoop was originally designed and optimized for high-throughput computing via batch processing, so job turn-around times between submission and completion are not a primary focus in Hadoop architecture. As such, Hadoop heartbeat periods are infrequent. For example, the default heartbeat period between the NameNode and DataNode in the Hadoop distributed file system (HDFS) [2] is three seconds. It is too long for short tasks, because it can exceed the task execution time.

In this paper, we suggest a solution to improve Hadoop's system performance by optimizing its heartbeat periods. We determine the types of heartbeats that provide greater impact on performance improvements and characterize the types of queries that will enable performance improvements when the heartbeat periods are reduced. Our contributions are as follows.

First, we evaluate the performance of Hadoop 2.0 through experiments that deliberately tune Hadoop heartbeat periods. In Hadoop 2.0 architecture, three kinds of heartbeats are used to control the Hadoop distributed file system and manage cluster resources. We characterize the I/O performance and application performance according to changes in the heartbeat periods and compare the performance results to those of the default configurations in Hadoop. We use the DFSIO benchmark, a built-in application to measure storage performance, to measure storage within the Hadoop file system. The experimental results show that the read I/O performance is improved by 12 percent, and the Terasort Map task execution time is shortened by 5.5 percent over the performance results of the default heartbeat periods. Moreover, we analyze the expected performance, system costs, and limitations using simplified mathematical equations.

Second, we evaluate Hive performance according to changes in heartbeat periods using TPC-H on a Hive workload [3]. We choose Hive as target application because it is the most widely used application in the Hadoop ecosystem. When we reduce all of the heartbeat periods in Hadoop, the execution time decreases by up to 18.9 percent compared to the execution times in the default configurations, depending on query type. Our analysis of the experimental results shows that queries with more nested select loops provides greater performance improvements than other types of queries.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 offers background information on Hadoop and Hive. Sections 4 and 5 describe the performance experiment results and analysis of Hadoop and Hive, respectively. Section 6 draws conclusions.

2. Related Work

There are several studies on Hadoop performance improvements, which focus on tuning the option parameters in Hadoop. For example, one study [4] achieved performance improvements by modifying the HDFS data block size configurations, data compression, Java

Virtual Machine (JVM) policy, and the copy phase in the Map and Reduce process. However, unlike in our work, they did not try to optimize the heartbeat periods to improve system performance.

The study [5] proposes HDFS-AIO using Adaptable I/O system(ADIOS) [6] which supports many different I/O methods, data formats and parallel file systems. It supports many high performance I/O techniques, such as data staging, asynchronous I/O and collective I/O. And it enables HDFS to select optimal I/O routines and parameter values for a particular platform without modifying the source code. It needs additional middleware to achieve performance gains, but our work just requires changing configurations.

The Starfish system [7] attempts to optimize Hadoop by modifying configurations. It first collects the previous job's profile information, then estimates the virtual job's running time through simulating this job's execution with different searches through the parameter space to find the optimal configuration settings with shortest estimated job execution time [8]. However, Starfish system adopted recursive random search algorithm. Yet certain features render it inefficient in achieving a global optimum. In contrast, we propose a way to select the parameter's optimal value by setting up an equation.

Another study [9] found that optimizing small MapReduce jobs improved overall system performance. The optimization shortened the time used to first initialize and configure the job. It also replaced the allocation process from the pull model to the push model. Like our work, the study focused on performance improvements for a bundle of short tasks. However, they approached the problem by changing the message direction incurred by the scheduling model difference, whereas our work does not change the Hadoop push-pull message architecture.

In a fifth paper, the focus was placed on Hadoop's heartbeat messaging. In this study [10], Worker failure was detected through a heartbeat during the job execution. When the failure occurred, the total execution time experienced further delays. Accordingly, they reduced the failure recovery time by sensing heartbeat activity. Our work achieves performance improvements in the Hadoop system by reducing heartbeats, which is applicable to a normal job execution environment.

3. Background

In this section we describe the background knowledge relevant to the research conducted. First, we describe types of heartbeats in Hadoop, as well as their functionalities and usage. Then we introduce the internal structure, data structure, and query statement of Hive.

3.1. Hadoop Heartbeat

In distributed processing systems, a heartbeat is basically intended to periodically check the node's liveness. Beyond checking liveness of daemons, controlling tasks, and other information, tasks are incorporated inside heartbeats to reduce the number of messages between nodes. In particular, the NameNode and JobTracker are central entities that manage the other nodes in the cluster. Thus, Hadoop restricts the number of messages sent to these nodes in order to reduce their loads. This paper focuses on Hadoop 2.0 architecture. In Hadoop 2.0 there are two kinds of heartbeats: the HDFS heartbeat and the Yarn heartbeat.

3.1.1. HDFS Heartbeat

The HDFS is a file system that supports a distributed environment. It is composed of a NameNode and DataNodes. The NameNode is a server that manages the metadata of the file system. DataNodes save the actual data blocks and deliver those blocks to the read and write data when clients need that data. The HDFS heartbeat is the heartbeat between the NameNode and DataNodes. A DataNode sends information about its liveness as well as its status change. When a client reads a certain file from the HDFS or writes to the HDFS, the client accesses the NameNode and receives the position of the block (e.g. the node's IP address that has the block.). Then the client directly accesses the DataNode and requests block read and write operations. Because the NameNode has scalability issues, it does not send a message to a DataNode proactively. Instead, the NameNode communicates with a DataNode in response to a message from that DataNode when needed. A DataNode sends periodic heartbeat signals to the NameNode. Thus, if there is a change in a DataNode's status caused by the completion of a read or write operation, a time delay might exist for the status update to the NameNode. The maximum delay is the heartbeat cycle. For this reason, the heartbeat cycle can influence the execution times of the read and write operations in the HDFS. The described process in which clients write blocks in the HDFS is shown in **Fig. 1**.

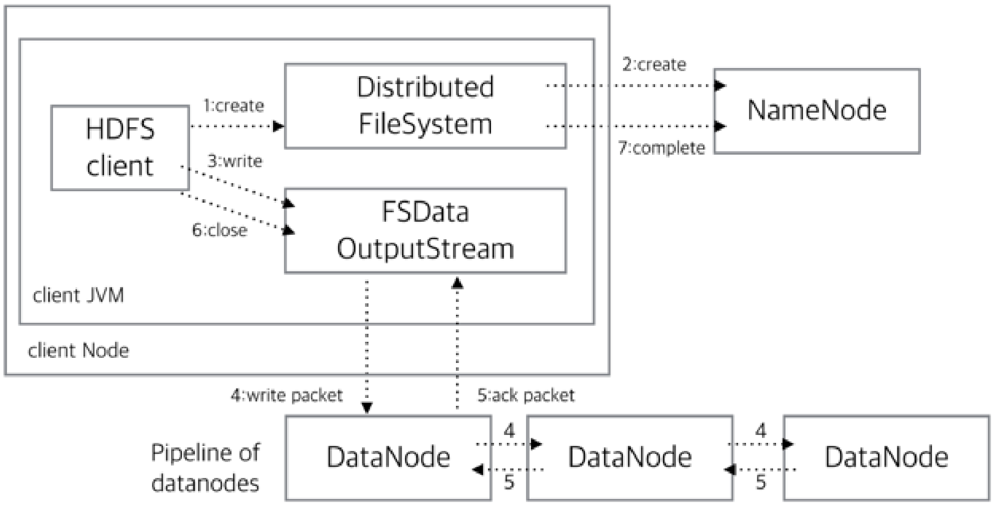


Fig. 1. The message passing flows in HDFS File write.

In the HDFS, the heartbeat signal period for DataNode → NameNode is set in the Hadoop folder as /etc/hadoop/hdfs-site.xml. The default value is set to three seconds. In large-scale systems with a lot of DataNodes, a short heartbeat cycle can place excessive loads on the NameNode. However, in the case of Hadoop clusters with less than a hundred DataNodes, DataNode updates with a shorter heartbeat period do not lead to a large load on the NameNode. Therefore by reducing the heartbeat period below the default values an improvement in HDFS performance is expected.

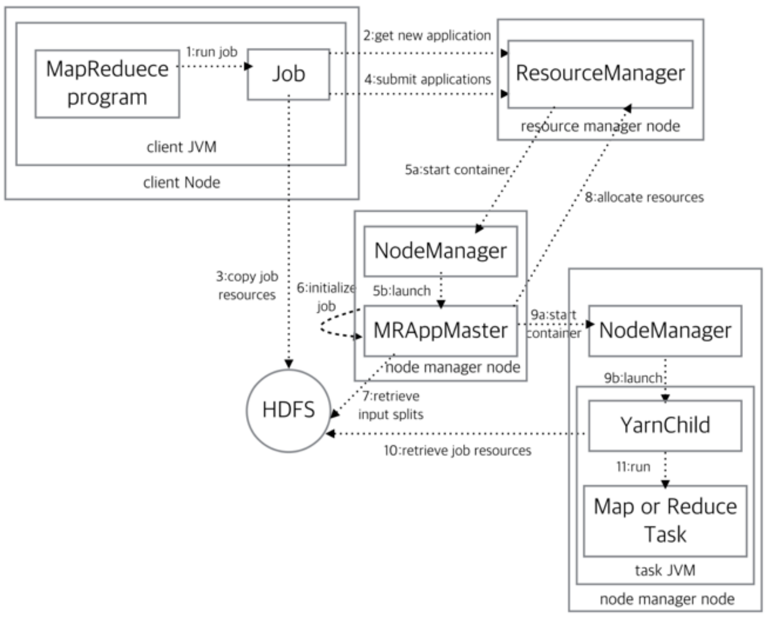


Fig. 2. The job execution process in YARN.

3.1.2. Yarn Heartbeat

In Hadoop 2.0, Yarn [1] is supported, as shown in **Fig. 2**. Yarn is a new resource management and scheduling method that separates the job scheduling and monitoring functions that were managed by JobTracker in Hadoop 1.0. Yarn's Resource Manager (RM) and Application Master (AM) manage the necessary functions. In particular, RM manages all of the worker nodes in clusters and receives reports on the status of each worker node. When jobs require resources, RM assigns the necessary resources to meet the specific requirements of each job. AM manages each job and is located in each server. It manages worker nodes, determining whether they satisfactorily execute tasks. If the tasks are completed, AM disappears after collecting and sending the results to RM and the client. Each worker node is managed by Node Manager (NM). In Yarn, each state is reported via heartbeat to reduce the load on the central server in the same manner as the HDFS. Heartbeats can be divided into two types:

I. Node Manager → Resource Manager heartbeat

II. Application Master → Resource Manager heartbeat

In Type I, NM sends a heartbeat to RM for each cycle, and this heartbeat contains the status information of the current node.□In Type II, AM sends a heartbeat to RM for each cycle, and this heartbeat contains the status information for the current job being performed, as well as the job results.

The default value of each heartbeat is one second and it can be configurable if needed. For Type I NM → RM heartbeat can be set in Hadoop the /etc/hadoop/yarn-site.xml file. For Type II AM → RM heartbeat can be set in the /etc/hadoop/mapred-site.xml file.

3.2. Hive

The HDFS has a completely different underlying structure from common databases that support SQL queries. Hive [3], often called SQL in Hadoop, enables database query processing by handling an SQL-like declarative language to access the HDFS. The structure of the Hive is shown in **Fig. 3.** The Hive Thrift Server supports multiple computer languages for the client. The query from the client is passed to the Hive Driver through the Hive Thrift Server. In the driver, the query goes through the process of parsing, planning, and optimizing. After the process, the query is executed by Hadoop MapReduce and the HDFS.

Because the Hive query is performed by Hadoop MapReduce and the HDFS, the system performance is relevant to the length of the Hadoop heartbeat period. In this paper, we show how changes in the heartbeat period can influence system performance.
Fig. 4 shows an example of Hive data being stored in the HDFS. Hive stores its Rational Database table structure as multiple files in the HDFS. The default storage location for data processed by Hive is the user/hive/warehouse folder. The first-level folder, /myScore, is the database, followed by /univ, which is the table. The next level, /year=2014, is a partition of the table. In the partition, data is stored (i.e. /year=2014). In this way, the data is stored in the HDFS.

To provide access to the Hive data, Metastore is added to the Hive architecture. Metastore stores Hive database structure information, which is metadata about the table and partitions. As shown in **Fig. 3**, when a client sends a query to the driver, the driver gets metadata from Metastore to find the data corresponding to the query. Then the query is performed with the data taken from the HDFS.

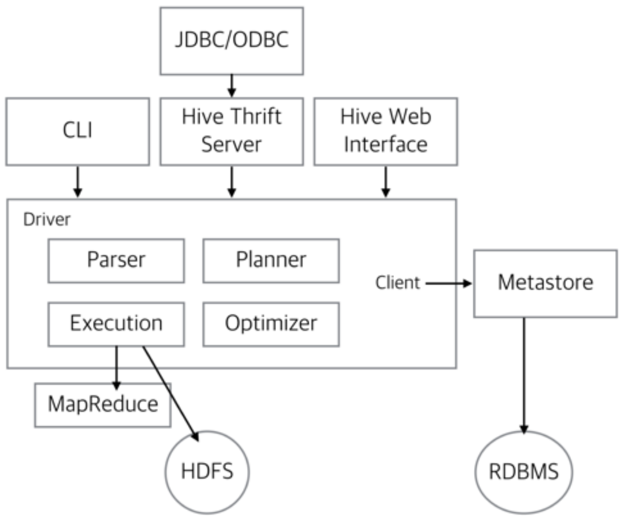
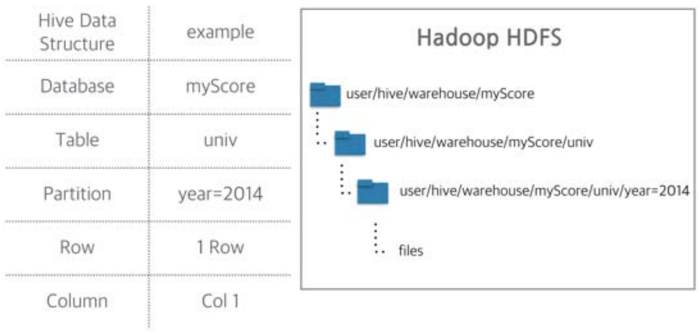


Fig. 3. Hive system architecture

3.2.1. Hive Query

Hive supports HiveQL, which is similar to SQL. HiveQL provides all of the queries, including **select, project, join, aggregate, and union**. The difference between HiveQL and RDBMS is that HiveQL is a schema on read that provides very fast data load times by verifying the data at the time of query execution. HiveQL enables the use of various queries for the same data and supports query optimization. In this study, we use TPC Benchmark (TPC-H) for our experiment. It consists of 22 SQL statements for the data and allows us to measure system performance via queries.



Hive Data Structure	example
Database	myScore
Table	univ
Partition	year=2014
Row	1 Row
Column	Col 1

Hadoop HDFS
user/hive/warehouse/myScore
user/hive/warehouse/myScore/univ
user/hive/warehouse/myScore/univ/year=2014
files

Fig. 4. Hive data structure

4. Hadoop Performance

In this section, we present the experiment results and analysis. We show how Hadoop's basic performance shifts according to changes in the heartbeat period. In particular, to understand the performance characteristic of Hadoop, we first examine the performance of the basic read/write I/O operations on the HDFS by measuring the DFSIO execution time of the read/write on the HDFS while varying the HDFS and YARN heartbeats. Next, we run the Terasort benchmark to determine the total application performance. Because the Hive query is performed by Hadoop MapReduce and the HDFS, the system performance is relevant to the length of the Hadoop heartbeat period. In this paper, we show how changes in the heartbeat period can influence system performance. The Hadoop cluster used in the experiment is shown in **Fig. 5**.

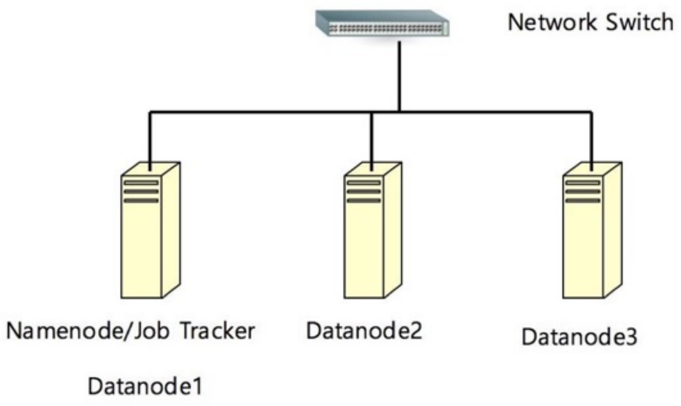


Fig. 5. The testbed cluster for experiments.

The server hardware consists of an Intel Xeon E3-1240V3 Quad-core processor, 4GB DDR3 RAM, a 1TB HDD (data storage), a 256GB SSD (data storage), a 120GB SSD (operating system), and SWAP/System space. The software environment includes the operating system (Ubuntu Desktop 14.04 LTS), a Java virtual machine (OpenJDK 1.7.0), and Hadoop (Apache Hadoop 2.6.0).

4.1. DFSIO

DFSIO is a workload that measures the basic file I/O performance of the read/write, which is available in the default Hadoop package. In this study, we carry out an experiment on the

10GB file read and write with a replication factor of 3. The detailed heartbeat period is configured as follows. For the experiment, we reduce the HDFS heartbeat from 3 seconds to 1 second, and reduce the other two heartbeats from 1 second to 0.2 seconds. We have done many experiments with different values, but the following values are representative values that clearly show the experiment result. The way to find optimal heartbeat period is presented in Section 4.6 as Equation (1).

<Heartbeat Configuration Cases for Experiments>

- Case 1. HDFS HB:3s, AM→RM HB:1s, NM→RM HB:1s (default)
- Case 2. HDFS HB:3s, AM→RM HB:0.2s, NM→RM HB:0.2s
- Case 3. HDFS HB:1s, AM→RM HB:1s, NM→RM HB:1s
- Case 4. HDFS HB:1s, AM→RM HB:1s, NM→RM HB:0.2s
- Case 5. HDFS HB:1s, AM→RM HB:0.2s, NM→RM HB:1s
- Case 6. HDFS HB:1s, AM→RM HB:0.2s, NM→RM HB:0.2s

We carry out experiments on six different heartbeat configuration cases, including the default configuration (Case 1 above). The heartbeat periods for Cases 2 to 6 have reduced from the default setting. We experiment on each configuration case five times and then average the execution times. **Fig. 6** and **Fig. 7** show the write and read execution time respectively, for each heartbeat configuration case. Note that the Y-axis, which indicates the execution time, starts from 260 seconds in **Fig. 6** and starts from 25 seconds in **Fig. 7**. Further discussion is provided below for DFSIO write and DFSIO read.

4.2. DFSIO Write

When the HDFS heartbeat periods are reduced, overall performance improves. Compared to the experimental results for Case 1, the results for Case 3 show a performance increase of 1.4%. Compared to Case 2, Case 6 show a performance increase of 0.8%. With file write, DataNodes must update the status to the NameNode. Therefore, reducing the HDFS heartbeat periods increases the total processing time.

When the Yarn heartbeats (AM→RM heartbeats and NM→RM heartbeats) are shortened to the same value, the overall performance improves. In particular, compared to Case 1, the Case 2 results show a performance increase of 2.8%. Compared to Case 3, the Case 6 results show a performance increase of 2.3%. This is because the task scheduling latency is reduced.

Our experiments also show which Yarn heartbeats have greater impacts on performance than others. When the AM→RM heartbeat is shortened (compare Case 3 to 5, and Case 4 to 6), there are noticeable increases in performance. However, when the NM→RM heartbeat is shortened (compare Case 3 to 4, and Case 5 to 6), only a slight improvement is observed, and no significant differences exist. This indicates that the process time between the RM and AM is a performance bottleneck in which the RM allocates resources to the AM, and the AM reports the results back to the RM. Therefore, we can achieve the most performance gains by reducing the AM→RM heartbeat. Finally, if we shorten all three heartbeats (Case 6), we obtain a performance improvement of 3.6% over the default case.

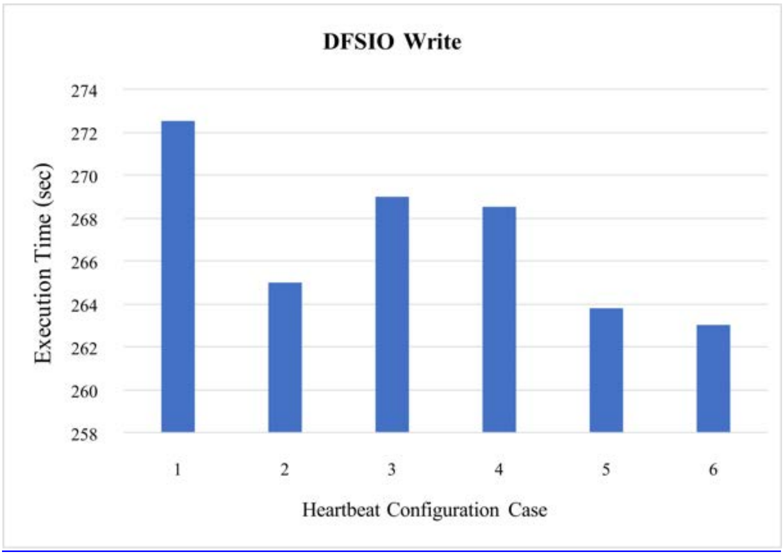


Fig. 6. The execution time for 10GB file write.

4.3. DFSIO Read

Next, we move on to the experiments on the DFSIO read executions with the same six configuration cases described above, and we evaluate the effects of the heartbeat periods on performance.

When the HDFS heartbeat is shortened to 1 second, the changes in performance are not significant (compare Case 1 to 3, and Case 2 to 6). This result differs from the aforementioned write results. Because no file blocks are created or removed from the perspective of the DataNode, it makes sense that the HDFS heartbeat is unrelated to performance in the case of read operation.

When two Yarn heartbeats (the AM→RM and NM→RM heartbeats) are reduced to the same value, the performance improves (a 13% improvement from Case 1 to 2, and a 12.5% improvement from Case 3 to 6). This occurs because reducing the Yarn heartbeat periods reduces the latency from the task scheduling. We notice that the performance improves more in read execution compared to write execution. This is because the read operation execution time is relatively shorter than the write operation execution time, which makes the effect more visible. Therefore, in the case of latency-sensitive workloads with intensive reads and not many data transfers, shortening the Yarn heartbeat periods can be very effective in improving performance.

Like the write execution experiments, our experiments show which Yarn heartbeats offer greater impacts on performance. When the AM→RM heartbeat is shortened (Case 3 → 5 and Case 4 → 6), there are noticeable increases in performance. However, when the NM→RM heartbeat is shortened (Case 3 → 4 and Case 5 → 6), the improvements were only very slight or nonexistent. As with write, we see that most of the performance gains come from reducing the process time required by RM to allocate resources to AM, and for AM to report the results to RM.

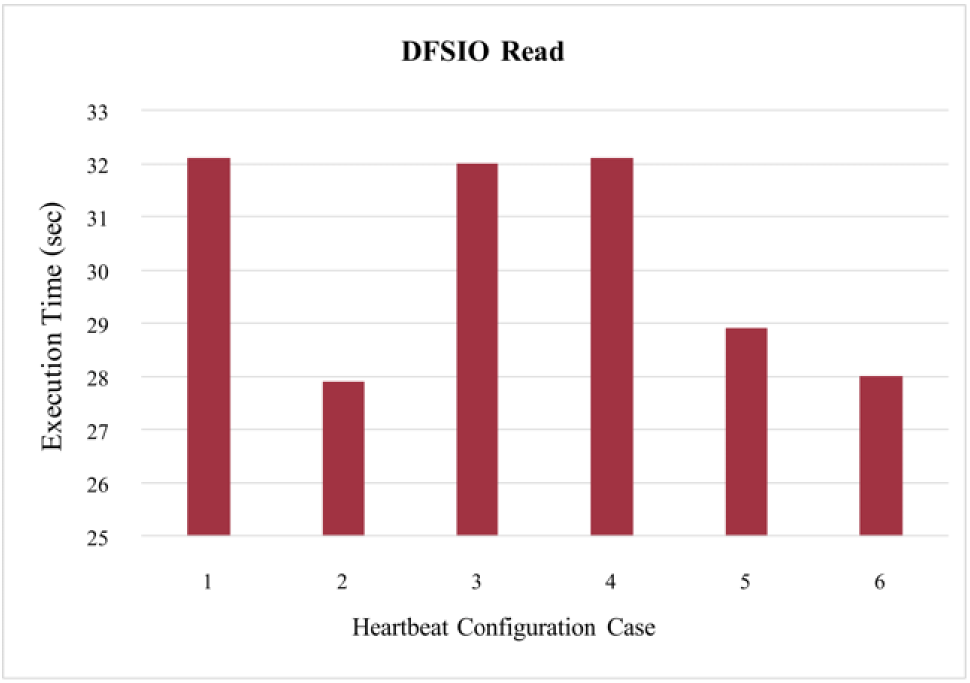


Fig. 7. The execution time for 10GB file read.

4.4. Discussion: DFSIO Read/Write

Summarizing the experimental results for DFSIO write/read, we obtain the following relevant conclusions.

- Shortening the HDFS heartbeat period improves write execution performance but does not have a significant effect on read performance.
- Shortening the AM→RM heartbeat period improves performance significantly for both read execution and write execution.
- The duration of the NM→RM heartbeat does not have a significant impact on improving performance in either read or write.

The NM→RM heartbeat is a system-wide parameter that directly affects the cluster system. Shortening its period increases the heartbeat frequency regardless of job execution status. Therefore, we need to be careful when changing the NM→RM heartbeat period. On the other hand, the AM→RM heartbeat period can be adjusted for the lifetime of each job without affecting the entire system. Therefore, we shorten the AM→RM heartbeat period for applications that require faster responses and effectively obtain performance improvements.

4.5. Terasort

Terasort is a benchmark test used to measure application performance in the Hadoop Mapreduce framework. Terasort is included in the default Hadoop package. In this experiment, we use the 10GB input file for sorting, and we compress the intermediate data (Map output) to reduce the shuffle data exchange. There are three simultaneous map tasks and one reduced task per DataNode. We use Slow Start, which delays the task reduction after all of the map tasks are completed. In this experiment, the total execution time is measured to represent the overall performance according to heartbeat cycle.

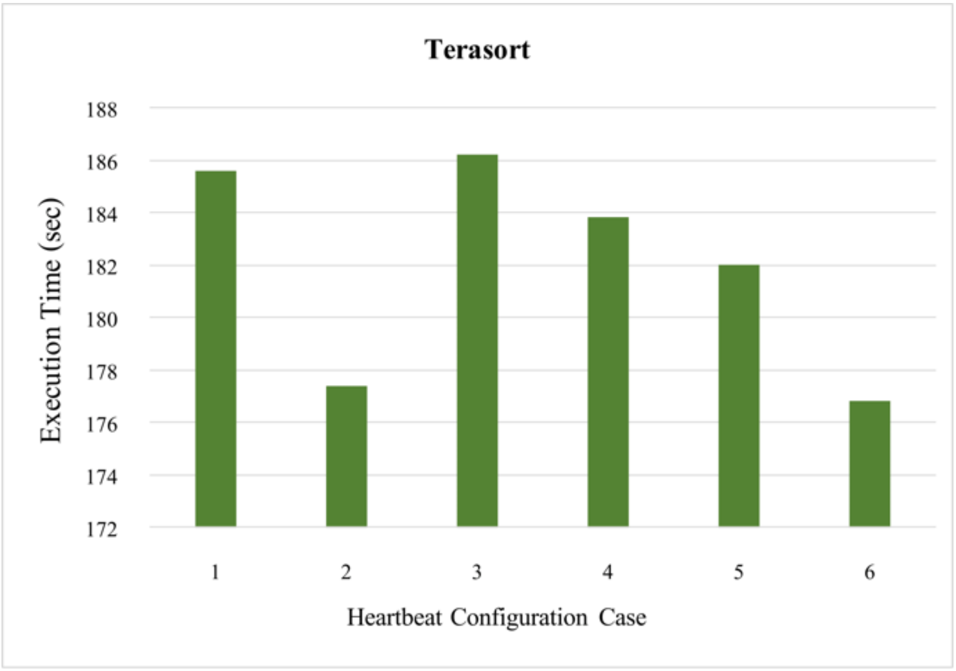


Fig. 8. All map task execution time by heartbeat setting(sec)

Fig. 8 shows the entire map task execution time per configuration. Each experimental setup is the same as in the DFSIO experiment. The reduce and shuffle phase results are not presented because they do not show significant differences.

When the HDFS heartbeat is shortened to 1 second, there are no significant changes in performance (Case 1 to 3, and Case 2 to 6). Because the read only operation of the HDFS block is performed in the map phase, the results are logically consistent with the results of the DFSIO read case.

When two heartbeats related with YARN are shortened to the same value, the overall performance improves (a 5.5% improvement from Case 1 to 2, and a 5.5% improvement from Case 3 to 6). This is because the shortened heartbeats lead to latency reductions via task scheduling. Because there are 76 map tasks, nine task execution phases (waves) can take place. When a map task is finished and the next task requires resource allocation, the RM should receive a message. Thus, a shortened heartbeat period can contribute to performance improvements over a series of map task executions.

With Terasort, when the AM→RM heartbeat is shortened (Case 3 → 5, and Case 4 → 6) and when the NM→RM heartbeat is shortened (Case 3 → 4, and Case 5 → 6), all cases show similar performance improvements. This indicates that both heartbeats play an important role in the process of task resource allocation. When both heartbeats are shortened (Cases 2 and 4), the largest performance improvement is obtained, as expected (shown in **Fig. 8**).

4.6. Heartbeat Signal Cycle Limitations and Analysis on System Performance

In this section, we theoretically approximate loads on the NameNode depending on the heartbeat period by formula, and we analyze the lowest heartbeat period limit. The definition of parameters for determining loads on the NameNode is as follows.

- Network Bandwidth (Bps) : B_n

- Average heartbeat size (Bytes) : PS_{HB}
- Average heartbeat transmission and processing time(seconds) : T_{HB}
- The number of heartbeats that can be processed per second : N_{HB}
- The number of DataNodes in the HDFS cluster : N_D
- HDFS heartbeat period (seconds) : TP_{HB}

The average heartbeat transmission and processing time can be computed as $T_{HB} = (10 \cdot PS_{HB}) / B_n$. The coefficient value of 10 in the equation is from the byte-to-bit conversion and CRC packet header overhead. The number of heartbeats that can be processed per second by a NameNode is an inverse of the average processing time.

$$N_{HB} = \frac{1}{T_{HB}} = \frac{B_n}{10 \cdot PS_{HB}}$$

When The number of heartbeat signals that the NameNode receives per second is defined as N_D / TP_{HB}. Thus, it should be $N_{HB} \geq N_D / TP_{HB}$.

$$N_{HB} \geq \frac{N_D}{TP_{HB}}$$

$$\frac{B_n}{10 \cdot PS_{HB}} \geq \frac{N_D}{TP_{HB}}$$

$$TP_{HB} \geq \frac{10 \cdot N_D \cdot PS_{HB}}{B_n} \quad \cdots \quad (1)$$

Let us assume a typical HDFS cluster consists of a hundred nodes with 1GBps Ethernet. We use Tcpdump to capture the packet to measure the heartbeat message length. The average transmission size for the heartbeats is approximately 2KBytes when executing DFSIO write. If this value is substituted into Equation (1), TP_{HB} should be longer than two seconds. However, DFSIO write does not incur an excessive overhead to reduce the heartbeat period to less than 2 seconds. This is because the above DFSIO write is executed in a severe environment and the heartbeat packet size is 396 bytes when there is no change in the HDFS. Equation (1) is a satisfactory guideline for determining the heartbeat period.

5. Hive Performance

5.1. TPC-H-ON-HIVE

The Transaction Processing Performance Council (TPC) is a non-profit corporation established in 1988 to regulate the standards of performance criteria for measuring system processing performance. The TPC-*alphabet* notation enables the benchmark test model presented in TPC. This allows for an evaluation of system throughput. The TPC-H benchmark [11] used in this experiment consists of 1 to 22 complex queries with a large amount of test data. By executing these queries, the database processing performance can be measured. In TPC-H-on-Hive [12], these 22 queries using HiveQL are implemented to measure the performance of Hive on Hadoop.

In this experiment, we deal with three types of heartbeats: (A) the HDFS heartbeat, (B) the MRAppMaster → RM heartbeat, and (C) the NM → RM heartbeat. As in Section 4, we examine six different heartbeat configuration cases, as follows.

<Heartbeat Configuration Cases for Experiments>

- Case 1. HDFS HB:3s, AM→RM HB:1s, NM→RM HB:1s (default)
- Case 2. HDFS HB:3s, AM→RM HB:0.2s, NM→RM HB:0.2s
- Case 3. HDFS HB:1s, AM→RM HB:1s, NM→RM HB:1s
- Case 4. HDFS HB:1s, AM→RM HB:1s, NM→RM HB:0.2s
- Case 5. HDFS HB:1s, AM→RM HB:0.2s, NM→RM HB:1s
- Case 6. HDFS HB:1s, AM→RM HB:0.2s, NM→RM HB:0.2s

In these experiments, we generate data using dbgen, which is provided by TPC-H, and we store the generated *.tbl files on the HDFS. After setting the configuration for each experiment, we execute TPC-H-on-Hive queries. We generate 1GB and 10GB files using degen and represent each query execution time to compare the change in performance according to heartbeat period. **Fig. 9** presents a normalized graph that shows the execution time of the TPC-H-on-Hive queries for the 1GB dataset with the default configuration (Case 1). **Fig. 10** presents a graph for the same experiment for the 10GB dataset. The queries without a value on each graph are failed queries.

When only the HDFS heartbeat is shortened from 3s to 1s, only a very small performance improvement occurs. For example, when comparing Cases 1 and 3 for the 1GB dataset, the performance increases by 0.09%, and for the 10GB dataset, the performance increases by 4.2%.

When two heartbeats related with Yarn are shortened to the same value, a greater improvement in performance is possible than with the reduction in the HDFS heartbeat. When comparing Cases 1 and 4 for the 1GB dataset, the performance increases by 18.0%. With the 10GB dataset, the performance increases by 5.7%. This is because the shortened heartbeats lead to latency reductions via task scheduling.

When the AM → RM heartbeat is shortened, performance improves. When comparing Cases 1 and 5 for the 1GB dataset, the performance increases by 15.3%. With the 10GB dataset, the performance increases by 3.5%.

In the experiment with the NM → RM heartbeat, there is either a small increase or no difference in performance depending on queries. When comparing Cases 1 and 6 for the 1GB dataset, the performance increases by 0.19%. With the 10GB dataset, the performance decreases by 10.5%. As before, most of the performance gains come from the reductions in the process time required for RM to allocate resources to AM, and for AM to report the results to RM.

Shortening all three kinds of heartbeat values achieves the largest improvement in performance. When comparing Cases 1 and 2 for the 1GB dataset, the performance increases by 18.9%. With the 10GB dataset, the performance increases by 11.7%.

5.2. Analysis on the execution time difference between 1GB and 10GB data set

For the TPC-H benchmark test, we generate 1GB and 10GB files and store them on the HDFS. We expect that the execution time for the 10GB files would be ten times longer than the execution time for the 1GB files because the amount of data increases 10 times. However, the experiments show otherwise. Two of the queries show the largest and smallest execution time gaps between the 1GB and 10GB datasets. These are TPC-H-on-Hive Queries 8 and 13, respectively. The execution of Query 8 takes 103.47s for the 1GB dataset and 428.99s for the 10GB dataset. These are the largest gaps. The execution of Query 13 it takes 65.62s for the

1GB dataset and 101.29s for the 10GB dataset. These are the smallest gaps.

We analyze the TPC-H benchmark.log file, which shows the execution results of each query. With Query 8, there are 15 MapReduce jobs for the 1GB dataset and 18 jobs for the 10GB dataset. Furthermore, the number of mappers and reducers increases in the query execution for the 10GB dataset (mapper: 9 → 44; reducer: 2 → 35). With Query 13, there are four MapReduce jobs for both the 1GB and 10GB datasets. However, the number of mappers and reducers increases in the query execution for the 10GB dataset at a lower rate than with Query 8 (mapper: 5 → 11; reducer: 4 → 11). When comparing the amount of data that the queries actually deal with, Query 8 deals with four times as much data (790MB for 1GB dataset, 7.9GB for 10GB dataset) than Query 13. As the actual data that each query processes increases, the execution time of the query increases, as does the gap in the execution times of the query for the 1GB and 10GB datasets.

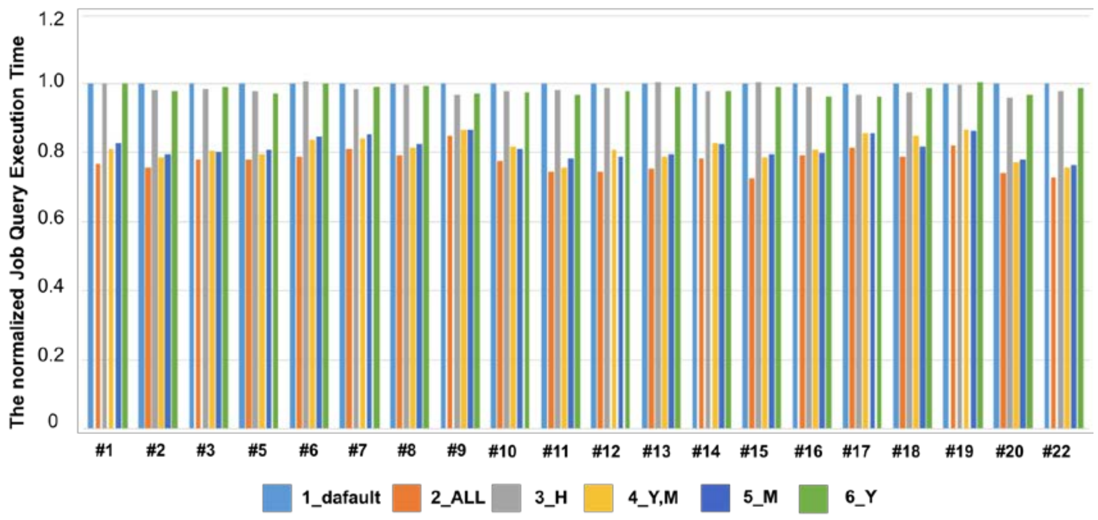


Fig. 9. A normalized TPC-H-on-Hvie Query execution time for 1GB data set.

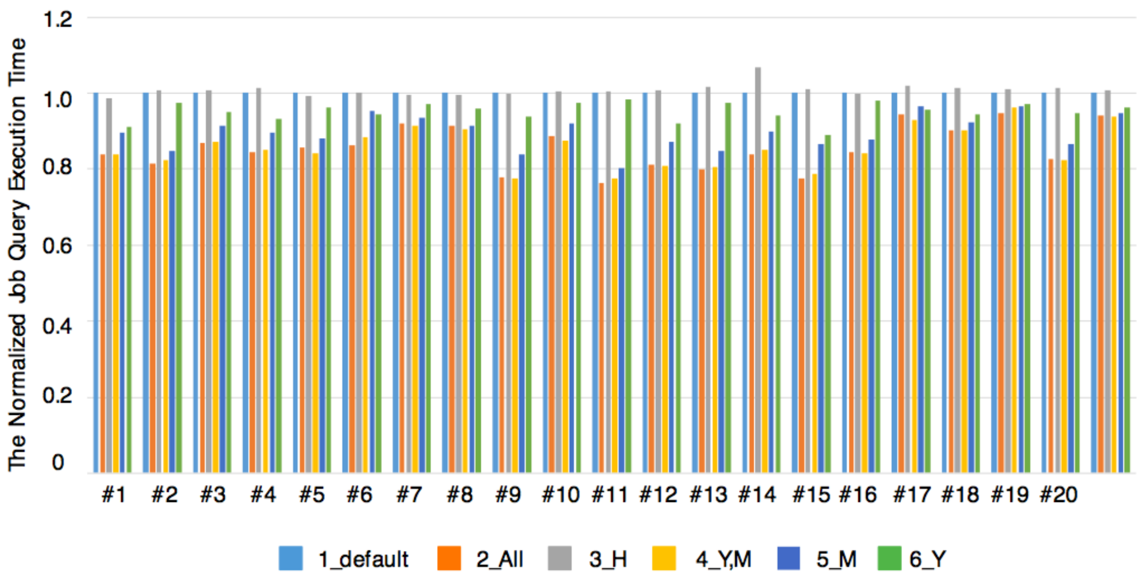


Fig. 10. A normalized TPC-H-on-Hive Query execution time for 10GB data set.

5.3. Analysis on query performance improvements

When all of the Hadoop heartbeat values are shortened from the default values (HDFS HB: 3s → 1s, AM→RM HB: 1s → 0.2s, NM→RM HB: 1s → 0.2s), the performance in the execution of queries increases. This performance increase applies to all of the queries, and it can be seen as the result of performance increases in Hadoop. For analysis, we compare each of the two queries that show the highest or lowest increases in performance among all queries.

The most improved performance queries are Queries 11 and 15, in that order. The least improved queries are Queries 19 and 6, in that order. In general, the execution time of each job is reduced when the heartbeat period is shortened. However, the amount of performance improvement differs by query type. In particular, as the number of nested select statements in a query increases, the number of read and write executions in the HDFS increases. As seen in Section 4, shortening the heartbeat periods is more effective when there are more HDFS read and write executions.

Queries 11 and 15 have nested select statements and result in the largest performance increases. Queries 6 and 19, which have the smallest increases in performance, do not have nested select statements. Queries 11 and 15, which have nested select statements, have more HDFS read/write executions and show larger performance improvements than Queries 6 and 19, which do not have nested select statements.

6. Conclusion

This paper suggests a novel method of improving Hadoop system performance by optimizing Hadoop heartbeat periods without incurring additional significant overhead. In Hadoop 2.0 (YARN), Hadoop has three kinds of heartbeats. We evaluate system performance by changing the three types of heartbeats and we analyze the resulting system performance. With the HDFS heartbeat, the changes affect the file write performance but they do not affect file reading performance. The heartbeat sent by Application Master to Resource Manager affects application performance as well as the file write and read operations. Because the heartbeat transmits during job execution, shortening the heartbeat period can significantly improve performance.

We propose a guideline for determining the minimum value of the heartbeat period via Equation (1). This equation takes into consideration the NameNode load, the number of DataNodes, and the network bandwidth in the HDFS clusters. We expect these guidelines to be useful for operating clusters. Lastly, the performance of the entire system is measured and analyzed via a TPC-H-on-Hive query using Hive in various heartbeat configuration sets. The results indicate that when reducing only the HDFS heartbeat, significant performance improvements should not be expected. On the other hand, the largest performance improvements are achieved when reducing three heartbeats. In particular, the query composed of multiple nested select statements receives more benefits by reducing the heartbeat periods. Thus, as long as this approach does not overload computing resources, it will elicit the best performance improvements. In the future, we plan to apply this scheme to medium and large-scale Hadoop clusters and evaluate the system performance.

References

[1] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O. Malley, S. Radia, B. Reed, and E. Baldeschwieler, “Apache Hadoop YARN: yet another resource negotiator,” in *Proc. of G. M Lohman, editor, ACM Symposium on Cloud Computing ’13*, no.5, 2013. Article (CrossRefLink)

[2] K. V Shavachko, Hairong Kuang and Sanjay Radia, “The Hadoop Distributed File System,” in *Proc. of MSST, 2010 IEEE 26th Symposium*, 2010. Article (CrossRefLink)

[3] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, “Hive - A Warehousing Solution over a Map-reduce Framework,” in *Proc. of Very Large Data Bases, August ’09*, vol. 2, issue.2, p. 1626-1629, 2009. Article (CrossRefLink)

[4] D. Heger, “Hadoop Performance Tuning - A Pragmatic & Iterative Approach.”

[5] Jung Kyu Park, “Improving the performance of HDFS by reducing I/O using adaptable I/O,” in *Proc. of International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016. Article (CrossRefLink)

[6] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin, “Flexible io and integration for scientific codes through the adaptable io system(adios),” in *Proc. of CLADE '08 Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, p. 15-24, 2008. Article (CrossRefLink)

[7] H. Herodotou et al., “Starfish: A Self-tuning System for Big Data Analytics,” in *Proc. of 5th Biennial Conference on Innovative Data Systems Research (CIDR 11)*, pp. 261-272, Jan. 2011. Article (CrossRefLink)

[8] K. Wang, X. Lin and W. Tang, “Predator – An experience guided configuration optimizer for Hadoop MapReduce,” in *Proc. of 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2012. Article (CrossRefLink)

[9] J. Yan, X. Yang, C. Yuan, and Y. Huang, "Performance Optimization for Short MapReduce Job Execution in Hadoop," *IEEE*, 2012. Article (CrossRefLink)

[10] H. Zhu, H. Chen, "Adaptive failure detection via heartbeat under Hadoop," *IEEE*, 2011. Article (CrossRefLink)

[11] Transaction Processing Performance Council. TPC-H Benchmark Specification.
Available: Article (CrossRefLink)

[12] TPC-H-Hive. [Online] Available : Article (CrossRefLink)

Jaehwan Lee is an Assistant Professor at the Department of Electronics and Information Engineering of Korea Aerospace University. He received his B.S. and M.S. in Electrical Engineering from Seoul National University, and Ph.D. in Computer Science from University of Maryland at College Park. He has several industry research experiences; Korea Telecom (KT) as a senior researcher, NEC labs in America and Bell labs, Alcatel-lucent as a research intern, and Samsung System Architecture lab in US as a Research Staff Engineer. His research interests include distributed computing, high-performance computing, and Big-data infrastructures to support data intelligence. He was a recipient of the General Electric (GE) Scholarship and the Korean Government Scholarship for Electric Power Industry.

June Choi received his B.S and M.S. in Electronics and Information Engineering from Korea Aerospace University in Korea. He is currently working in a company that develops Big Data platform and services Cloud infrastructures for a variety of industries. His primary research interests are in distributed-computing and high-performance computing.

Hongchan Roh received the BS degree in 2006, the MS degree in 2008, and the Ph.D. degree in 2014, all from Yonsei University, Seoul, Korea. He is currently a research fellow for SK Telecom. His current research interests include distributed deep learning, network processors, database systems, flash memory, and SSD.

Ji Sun Shin received her B.Sc. degree in Computer Science from Seoul National University, Seoul, Korea in 2001 and her Ph.D. degree from the University of Maryland, College Park, USA in 2009. From 2009 to 2012, she was a Senior Engineer at Samsung SDS, Seoul, Korea, where she was involved in the development of network access control systems. From 2012, she is working as an Assistant Professor of Computer and Information Security at Sejong University. Her research interests are computer network security, cryptographic protocols and applied cryptography.