An Efficient Design and Implementation of an MdbULPS in a Cloud-Computing Environment

Myoungjin Kim[1], Yun Cui[1] and Hanku Lee[1,2,*]
[1] Department of Internet and Multimedia Engineering, Konkuk University
Gwangjin-gu, Seoul 143-701 – Republic of Korea
[e-mail: tough105, shhan87, ilycy@konkuk.ac.kr]
[2] Center for Social Media Cloud Computing, Konkuk University
Gwangjin-gu, Seoul 143-701 – Republic of Korea
[e-mail: hlee@konkuk.ac.kr]
*Corresponding author: Hanku Lee

Received March 5, 2015; revised July 7, 2015; accepted August 3, 2015; published August 31, 2015

Abstract

Flexibly expanding the storage capacity required to process a large amount of rapidly increasing unstructured log data is difficult in a conventional computing environment. In addition, implementing a log processing system providing features that categorize and analyze unstructured log data is extremely difficult. To overcome such limitations, we propose and design a MongoDB-based unstructured log processing system (MdbULPS) for collecting, categorizing, and analyzing log data generated from banks. The proposed system includes a Hadoop-based analysis module for reliable parallel-distributed processing of massive log data. Furthermore, because the Hadoop distributed file system (HDFS) stores data by generating replicas of collected log data in block units, the proposed system offers automatic system recovery against system failures and data loss. Finally, by establishing a distributed database using the NoSQL-based MongoDB, the proposed system provides methods of effectively processing unstructured log data. To evaluate the proposed system, we conducted three different performance tests on a local test bed including twelve nodes: comparing our system with a MySQL-based approach, comparing it with an Hbase-based approach, and changing the chunk size option. From the experiments, we found that our system showed better performance in processing unstructured log data.

Keywords: NoSQL, MongoDB, Cloud Computing, Hadoop, Big data processing, Banking System

A preliminary version of this paper was presented at ICONI 2014, and was selected as an outstanding paper. In addtion, this research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-H8501-15-1004) supervised by the IITP(Institute for Information & communications Technology Promotion)

http://dx.doi.org/10.3837/tiis.2015.08.026 **ISSN : 1976-7277**

1. Introduction

Log data generated in various fields record a substantial amount of information generated during computer system management, with the accumulated log data being used in a wide variety of processes, including computer system management, system optimization, and user-customized optimization for various services [1, 2].

In this paper, we propose a MongoDB-based unstructured log processing system (MdbULPS) in a cloud-computing environment that processes large amounts of log data. Our system focuses on handling the log data generated in a bank, among various types of log data. In general, most log data generated in a banking process arise during customer transactions, and a separate log processing system is required to collect, store, categorize, and analyze the data according to the customer's transaction processes.

With a conventional computing environment, it is difficult to expand flexibly the storage capacity required to process a considerable amount of rapidly increasing unstructured log data and to implement a log processing system providing features that categorize and analyze unstructured log data. Therefore, in this study, we propose and implement a cloud environment-based log data processing system by introducing cloud computing technology [17, 30] for handling the unstructured log data that were difficult to control in a conventional computing infrastructure and system environment, such as distributed computing.

Recently, "big data" [18, 19, 20] processing and analysis, which were difficult to handle in the existing computing environment, have become uncomplicated because of the appearance of cloud computing [4]. In fact, cloud-computing technology has been actively studied in various fields, because it can collect, categorize, and analyze rapidly growing real-time data as well as mass data accumulated over a long period of time [5].

The proposed system introduces infrastructure as a service (IaaS) in a cloud environment providing flexible expandability of computing resources, thereby flexibly expanding resources such as storage, memory, and CPU in situations where log data are accumulated for long time or are increasing exponentially. Furthermore, to overcome the limitations of the existing analysis tool in cases where a batch analysis for accumulated unstructured log data is required, Hadoop-based analysis module is introduced to the proposed system, with a rapid and accurate function for parallel and distributed log-data processing. Our system distributes collected log data in a Hadoop distributed file system (HDFS) for batch processing-based analysis, and MySQL and MongoDB for real-time processing-based analysis, and it alternates between these depending on the type of analysis. In particular, our system provides an automatic recovery function in the event of data loss and system failures by introducing Hadoop distributed file system (HDFS), which creates and stores replicas of accumulated log data in blocks.

Further, from the perspective of a banking-related log processing system, our system sends log data with information generated throughout banking, such as a customer's transaction waiting time and banking processing time to our cloud server to collect, categorize, and analyze the unstructured log data. Then, our system provides a function that monitors and displays the analyzed results to users and administrators via the web monitoring system.

In particular, to process and analyze unstructured data, the proposed system utilizes MongoDB, a non-relational database, in place of the existing relational database. Thus far, relational databases, such as MySQL, have been used to store and manage data. Relational

databases use predefined schemata to store and manage structured data. However, relational databases with strict schema structures are difficult to merge with other databases with different schemata. In addition, such databases have an inappropriate structure for processing unstructured data such as log data and have the problem of expanding a database through adding system nodes. To overcome such structural limitations, numerous researchers have been conducting studies on distributed databases, such as NoSQL, with flexible schema structures [3, 6, 7].

MongoDB, representing NoSQL, is a document-oriented data model with flexible schemata; therefore, it is suitable for systems that process and analysis unstructured data. Node distribution is a key element in distributed databases for dealing with problems related to the exceedance of storage capacity resulting from rapid log data increase. Although numerous databases provide the sharding [21, 22] function to distribute and split data through node distribution, existing relational databases such as MySQL have difficulty distributing and splitting data owing to their strict schemata. In addition, for node distribution, such databases have difficulty in providing maintenance, because they add and distribute nodes manually. In contrast, MongoDB provides ease of management through the AutoSharding function, which allows users to divide data into small chunks and store them in numerous shards.

The proposed system consists of a log collector, a log graph generator, MongoDB, Hadoop-based analysis, and MySQL modules. The log collector module distributes log data generated from banks into the MongoDB or MySQL module of our cloud server according to log type. Using log data analyzed by the MongoDB, Hadoop-based analysis, and MySQL modules, the log graph generator module provides the analyzed results to the user in the form of a web interface that includes graphs. Log data required for real-time analysis are stored in the MySQL module, and the analyzed information is displayed to users by the log graph generator module in real time. In addition, log data accumulated on an hourly basis are stored on the MongoDB module, and that information is also displayed. Log data accumulated in the MongoDB module are analyzed and processed through the parallel and distributed processing methods of the Hadoop-based analysis module, and the results are displayed by the log generator module.

To evaluate the proposed system, we conducted three different performance tests on a local test bed including twelve nodes: comparing our system with a MySQL-based approach, comparing with a relational database-based approach, and changing the chunk size option. From the experiments, we found that our system showed better performance in processing unstructured log data.

The remainder of this paper is organized as follows. Section 2 discusses MongoDB and relational databases. Section 3 describes our model and the core architecture of MdbULPS, as well as presenting the workflow for processing a large amount of log data. Section 4 explains a prototype of the proposed system and its configuration, including implementation issues. Section 5 provides a description of the performance evaluation conducted to verify the effectiveness of the proposed system. Section 6 presents our concluding remarks and plans for future work.

2. Related Work

NoSQL (often interpreted as Not Only SQL) is a non-relational database. Relational databases with pre-determined schemata have been used in various fields; however, they have limitations in processing unstructured data. Therefore, NoSQL databases have received

remarkable attention from developers and researchers because of its flexible schemata structure and its ability to expand the structure of a database through node distribution [6, 7, 8, 9, 10]. The following section compares and describes the NoSQL and relational database data models, presenting the key concept and structural features of MongoDB.

2.1 Comparison of NoSQL with Relational Databases

A relational database, with features such as atomicity, consistency, isolation, and durability, maintains the integrity of a database through the transaction function. A relational database with a structure of predefined schemata can minimize data redundancy, because normalization is conducted when data are stored in the database. Furthermore, it has an advantage with processing complex query operations such as join. However, although relational databases are used in various fields, such a database is inappropriate for processing and analyzing unstructured data, because the data are stored and normalized in the database with a predefined schema structure. In addition, it has a limitation with regard to expanding exponentially increasing data through node distribution [6].

A NoSQL database provides a mechanism for storage and retrieval of data modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability [13]. Although NoSQL does not support complex operations, such as join, that are provided only in existing relational databases, it has a suitable structure for processing unstructured data owing to the fact that it can support ease of expanding and splitting data in a distributed environment through node distribution. Therefore, NoSQL, with flexible schemata, has received remarkable attention from developers and researchers as an alternative to overcome the limitations of relational databases [6, 7, 8, 9, 10, 11]. There are various approaches for classifying NoSQL databases. We describe only three models.

2.1.1 Key-value-oriented Model

The key-value-oriented model has a simple structure of storing data in the form of key and value, using the key to search for the value. The data model is divided into four models, the key-value cache, key-value store, key-value store (eventually consistent), and key-value store (ordered) models. Representative databases are Dynamo, MemcacheDB, TokyoTyrant, Voldemort, Aerospike, Memchched, Coherence, and GigaSpaces.

2.1.2 Column-oriented Model

The column-oriented data model stores data tables with sections of columns of data rather than rows of data. In fact, the model serializes all of the values of a column together, then the values of the next column, and so on. In contrast, most relational databases store data in rows. Because the model stores data separately in columns, it has an advantage with significantly fast processing time for storing and querying data when the data increases exponentially. Representative databases are Accumulo, Druid, Hbase [24, 25], and Vertica.

2.1.3 Document-oriented Model

The document-oriented model with flexible schemata is designed for storing, retrieving, and managing document-oriented information, also known as semi-structured or unstructured data. Therefore, the model does not require pre-determined schemata [5, 6, 7]. The model can extract and store data and metadata using the XML, JSON, and BSON formats. Representative databases are MongoDB [31], CouchDB [26], SimpleDB, Cassandra, and ArangoDB.

2.2 MongoDB

MongoDB, an open source C++-based NoSQL database project released in 2009, leads NoSQL database. It is a document-oriented data model that has a flexible schema structure and is therefore suitable for processing unstructured log data. The central concept of MongoDB is that documents are the basic units for storing data. A document supports and provides a variety of types of data structure consisting of key-value structures, such as numbers, dates, arrays, and inner documents. A document stores data in key-value structures in JavaScript object notation (JSON) form using binary JSON (BSON).

MongoDB provides two replication policies, master–slave and replica-set, to be prepared for system failures and data loss. The master-slave approach improves the system's reliability and performance by storing replicas generated in a master node in slave nodes. However, this policy has a severe single-point-of-failure (SPOF) disadvantage. The entire system halts when system failures occur in the master node. The replica-set policy is used to overcome this problem. MongoDB provides high availability through use of replica sets. A replica set consists of two or more copies of the data. Each replica is either a primary or a secondary replica. When a master node fails, MongoDB automatically selects an alternative master among the remaining nodes and then generates and distributes new replicas for missing replicas [14].

Although numerous databases provide the sharding [21, 22, 23] function to distribute and split data through node distribution, existing relational databases such as MySQL have a problem with distributing data because of their strict schemata. In addition, such databases have difficulty providing maintenance for node distribution, because they add and distribute nodes manually. In contrast, MongoDB provides ease of management via the AutoSharding function, which allows users to divide data into chunks and store them in shards.

3. MdbULPS in a Cloud Computing Environment

3.1 Background for Introducing the Cloud-Computing System

We designed and developed our system in the IaaS cloud environment. Therefore, our system can support ease of system management by providing flexible expandability of system resources such as storage, RAM, and CPU to process or store more massive amounts of unstructured log data. With the analysis system of the existing computing infrastructure, it is difficult to analyze accumulated unstructured data or massive log data. In these environments, cloud-computing technology is the most promising alternative for solving such problems. Thus, the technology is receiving considerable attention in the field of "big data" processing.

Cloud and distributed processing techniques based on Hadoop [27, 29] have become the core platform for processing large-scale unstructured data. Our system can analyze massive log data in a short period of time in a parallel and distributed manner by adopting MapReduce [15, 28] and HDFS [16, 32] into the Hadoop-based analysis module in MdbULPS. In fact, we use HDFS in the module for automatic system recovery against system failures and data loss. After being distributed, log data are replicated at three data nodes according to the Hadoop distribution policy, thus complying with the entire distributed processing procedure and enabling recovery from data loss. The NameNode controls mechanisms for selecting management nodes, copying replicas, and conducting automatic recovery. However, our system focuses on MongoDB-based analysis, rather than Hadoop-based analysis. Recently, most analysis systems utilize Hadoop to provide high availability along with open-source

software packages from the Hadoop Federation. Yet, Hadoop is exclusively focused on batch processing, rather than real-time processing. The proposed system, insofar as it deals with banking-related log processing, demands non-batch processes, such as iterative jobs, real-time queries, and log-data streams. Although analysis systems for non-batch processes can be implemented by combining open-source software from Hadoop ecosystem (e.g., Storm [33], Spark [34], Kafa [35], etc.), this approach is difficult to implement, and it results in a Hadoop dependency between open-source and legacy systems. Therefore, our system uses Hadoop and the HDFS exclusively for storing log data and as a journaling file system, and it uses MapReduce for analyzing log data based on batch processing.

3.2 Architecture of MdbULPS

The proposed MdbULPS is designed to run on a Hadoop cluster in a distributed manner. The overall system architecture is shown in **Fig. 1**. The proposed system has five components: the log collector module (LCM), the log-graph generator module (LGGM), the MySQL module (MSM), the MongoDB module (MdbM), and the Hadoop-based analysis module (HadoopAM).

3.2.1 LCM

The primary role of LCM is to collect the log data generated during the entire process of all customer transactions from each bank in the cloud server. In addition, it distributes the collected log data in the cloud server to MdbM or MSM according to the type of data.

Log data requiring real-time data analysis and log data requiring an hourly analysis are sent to MSM and MdbM, respectively. LCM transfers log data accumulated over a long period of time to HadoopAM, and the log data transferred from LCM are processed and analyzed by HadoopAM in a distributed and parallel manner.

3.2.2 LGGM

The primary role of LGGM is to visualize analyzed data through a web interface, providing graphs with analyzed results. Log data collected by LCM are analyzed by MSM, MdbM, and HadoopAM. LGGM generates pie, line, and bar graphs and a table based on the analyzed results and displays them in a web dashboard.

3.2.3 MSM

MSM serves to store and process the log data that manage the transaction number being handled by the teller and the number of customers waiting in line at each bank. Thus, MySQL processes the log data generated throughout the entire process of the customer's transaction in real time.

3.2.4 MdbM

MongoDB, which consists of multiple nodes, creates and stores replicas for the log data by using the replica-set policy to ensure redundancy and facilitate load balancing. In fact, MongoDB performs sharding in a manner that addresses the challenge of scaling to support high throughput and data consistency when the log data are stored. By default, there are three replicas, but that number can be changed using a MongoDB configuration. In our MdbULPS, MdbM manages and stores log information in MongoDB, including the transaction number, waiting and processing time for banking, number of bank tellers, and the like, once the

banking process is completed. In addition, depending on the requirements of a user's query, it extracts the relevant log data and then transfers those data for the required analysis.

3.2.5 HadoopAM

The main roles of HadoopAM are to perform quick and accurate parallel and distributed processing on the large-scale log data accumulated in MongoDB of MdbM, according to the user's analysis command, and then send the analyzed results to the graph generator module to create graphs. Through the MapReduce component provided by HadoopAM, users and administrators can perform an analysis to utilize statistical results, such as customer transaction processing time and waiting time for conducting banking business by extracting the log data by hour, date, month, year, and branch, using various queries. Furthermore, HadoopAM creates and stores replicas of the log data in blocks in a distributed manner using the HDFS, thereby ensuring data consistency and maintaining fault tolerance. HDFS is used in HadoopAM, as well as over the entire system. Using NameNode in HDFS, which systematically manages generated replicas, our system conducts an automatic system recovery using the replicas stored in other nodes, when unexpected system failures occur [32]. Thus, based on these systematic features, the proposed module and the entire system maintain the stability of the entire system and ensure reliability for safe log data analysis.

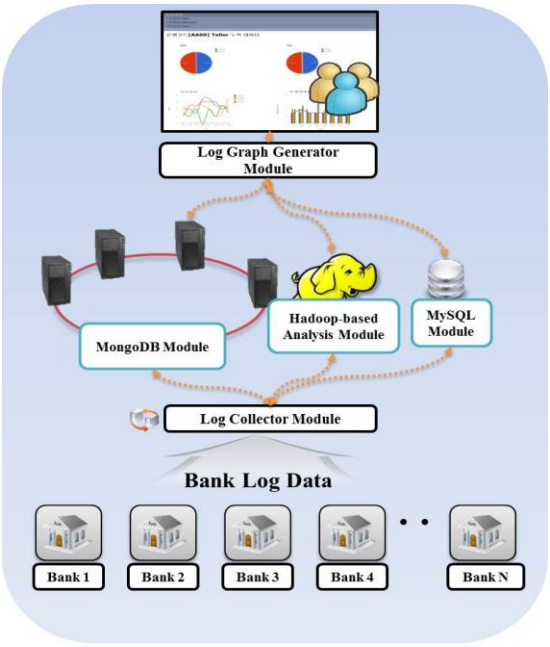


Fig. 1. Overall architecture of MdbULPS over a cloud-computing environment

3.3 Log Data Parameters

The purpose of the proposed system is to collect, categorize, analyze, and visualize log data generated during each stage of customers' banking processes. To secure accuracy and consistency of log data information used in data communication between modules, the parameters of the log data must be pre-defined. **Table 1** lists these pre-defined log data parameters. **Fig. 2** shows real data stored in MongoDB using our log data parameters.

```
mongos> db.BankingLog.findOne()
{
        "_id" : ObjectId("54eff4bc1f73691266d2da9b"),
        "bank_code" : "AA24",
        "job" : "N",
        "number" : "130",
        "generator_time" : "14:51",
        "teller" : "AA24B",
        "generator_wait_time" : "5",
        "teller_start_time" : "14:56",
        "teller_end_time" : "15:00"
}
```

Fig. 2. Example of real log data stored in MongoDB

The log data parameters are defined as follows: id, bank_code, teller, job, number, generator_time, generator_wait_time, teller_start_time, and teller_end_time. The id parameter automatically generated by MongoDB provides a unique identification number for each log datum. The bank_code parameter provides the unique code for the bank that generated the log data, with each bank having one unique bank_code. The teller parameter shows the unique identification number of the bank teller currently conducting the customer's banking activity. The job parameter is used for distinguishing types of banking transaction. In particular, the N value of the job parameter is assigned for general banking business, such as deposit, withdrawal, or remittance, while F is assigned for the others. The Number parameter indicates the number generated by the guest paging system. For example, the value 130 for the number parameter indicates that 129 people are lined up in front of the person who has the number 130. The generator_time parameter represents the time at which the number parameter was issued, and the generator_wait_time parameter shows the time until the beginning of the customer's banking activity after the number is generated. The teller_start_time parameter records the start time of the customer's banking transaction, and the teller_end_time parameter records the end time of that transaction.

Table 1. Definition of log data parameters

Type	Example
id	54eff4bc1f7385843833d3da9b
bank_code	AA01
Teller	t001
Job	N, F
Number	43
generator_time	09:10
generator_wait_time	20
teller_start_time	09:30
teller_end_time	09:40

3.4 Service Model and Workflow of MdbULPS

In this section, we introduce the MongoDB-based unstructured log processing service model as our motivation for designing and implementing MdbULPS, presenting the workflows of the sequential tasks in our system during the entire banking process. Workflows for real-time and accumulated log information are presented. The log information is divided into two types, real-time and accumulated.

3.4.1 MdbULPS Model in a Cloud-Computing Environment

Fig. 3 shows the proposed MdbULPS model. The model consists primarily of the data source, data collection, categorization and analysis, and output layers for the analyzed results. In the traditional system for processing log data, the analyzed results are sent to a central server after processing and analyzing the log data in local servers within bank branches generating log data. In this computing environment, with exponentially increasing log data, it is difficult to implement the realization of flexible storage expansion functions for processing a massive amount of unstructured log data and executing a considerable number of functions to categorize and analyze the stored unstructured log data. To overcome these limitations, the proposed service model can support flexible scalability for deployed computing resources such as storage, memory, and CPU, when the volume of log data increases exponentially by incorporating cloud-computing technology and various cloud-based open sources.

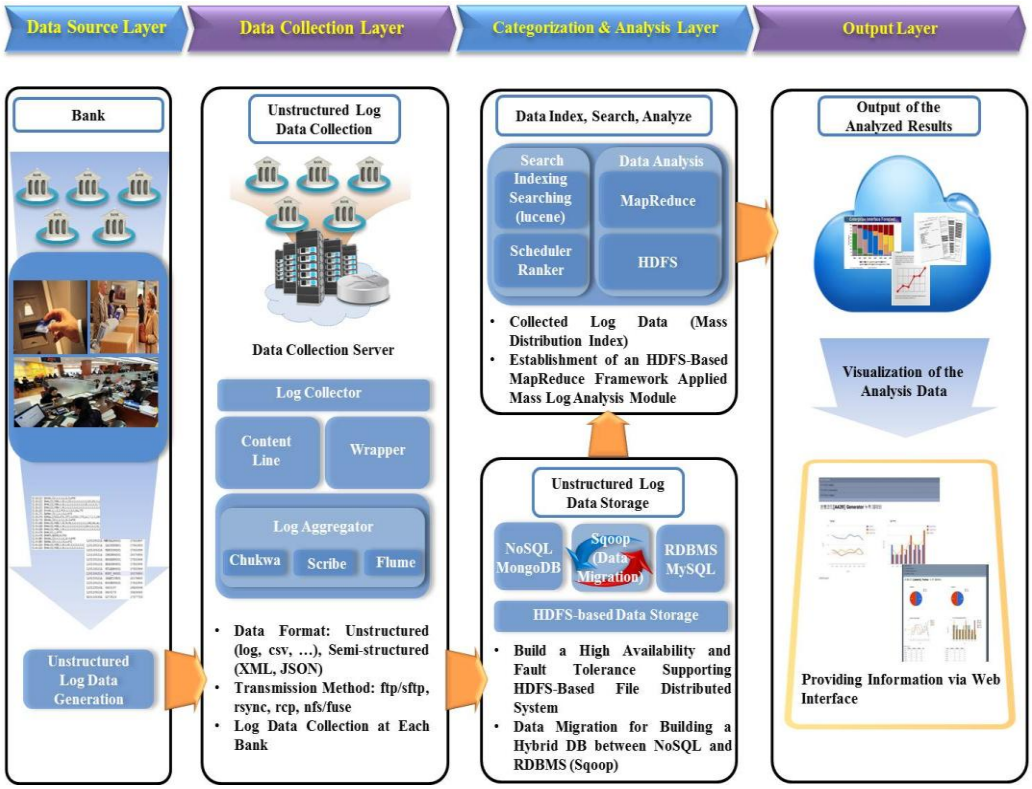


Fig. 3. MdbULPS model

The primary target of the proposed service model is unstructured log data resulting from transactions occurring in numerous branches. Log data generated in the data source layer that are not to be handled by local servers in each corresponding branch are sent directly to a central server in the cloud or to a virtual machine generated in the IaaS cloud environment. To maintain data collection scalability and efficiency, once log data are sent, the data collection layer uses Hadoop ecosystem-based open sources, such as Chukwa, Scribe, and flume, as well as various public APIs provided by vendors with raw data to collect the data. The categorization and analysis layer stores the data collected by the data collection layer in a

distributed manner in HDFS supporting automatic system recovery and maintaining data consistency. Furthermore, Sqoop is used in this layer for effective data migration, when collecting and processing unstructured data and structured data simultaneously; consequently, the user can easily transfer the data from the MongoDB to MySQL, and vice versa, when needed. The most valuable service of this layer is to analyze the collected data using Hadoop's MapReduce according to user requirements. Using this layer, developers and analysts can obtain an environment for simply developing analysis program code for the MapReduce algorithm and automatically conducting input and output processing for the log data and distributed and parallel processing by the MapReduce framework. Finally, the output layer for the analysis results visualizes the analyzed data in graphs and tables and provides the analyzed log data information to users through a web interface.

3.4.2 Workflow for Providing Real-Time Log Information

Fig. 4 shows a workflow diagram for providing real-time log information. The log data for real-time analysis with current banking transactions in progress and waiting is generated in real time, because a customer's banking transaction during business time already begins. The generated log data are collected by LCM and stored in MySQL of MSM for query and analysis. The analyzed real-time log data are transferred to LGGM, followed by display and monitoring of real-time information of a corresponding bank using graphs and tables via the LGGM web interface.

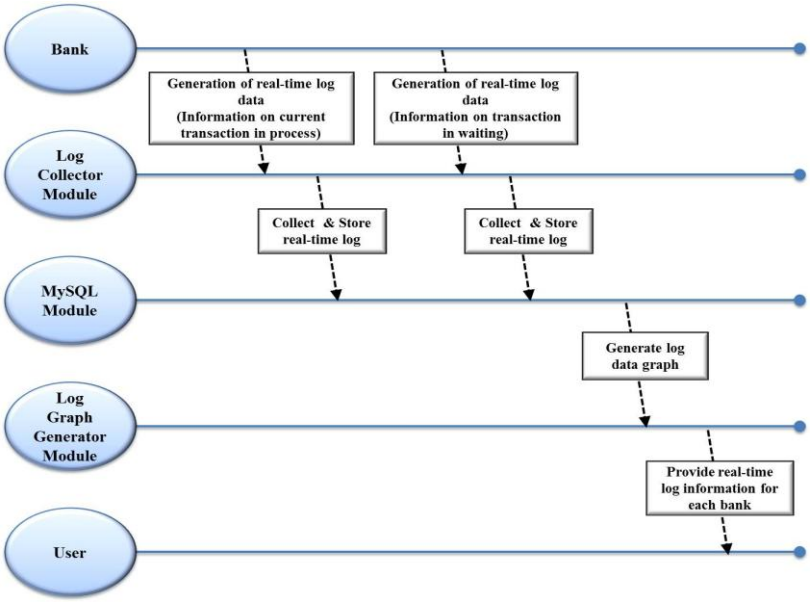


Fig. 4. Diagram of real-time log data processing workflow

3.4.3 Workflow for Providing Accumulated Log Information

Fig. 5 shows a workflow diagram for providing accumulated log information. The accumulated log data are generated at the completion of each customer's banking transaction. The elements of log data are listed and presented in **Table 1**. The generated log data are stored in document form in MongoDB of MdbM without processing data formalization. The accumulated log data become useful log information through analysis of the data in a

distributed and parallel manner using HadoopAM by hour, date, month, year, and branch. The information is provided to users through the web interface.

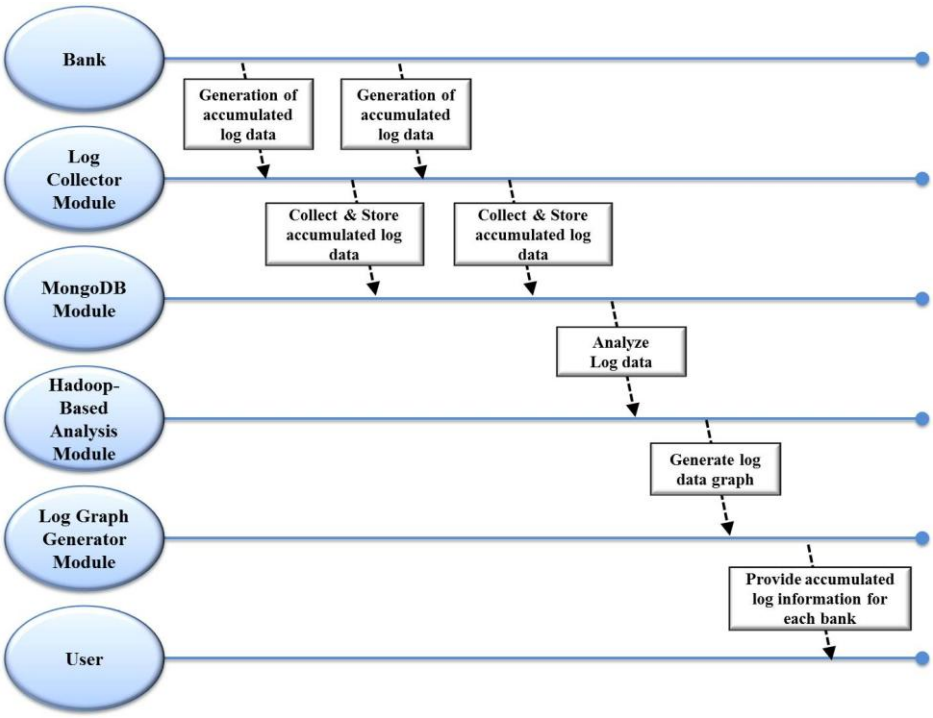


Fig. 5. Diagram of accumulated log data processing workflow

4. Implementation and prototype

4.1 MdbULPS Configuration and Implementation

For our prototype implementation, we constructed our own cluster servers in a cloud-computing environment consisting of 12 nodes. Each node consisted of Linux OS (Ubuntu 12.04 LTS 64-bit) running on an Intel core i7-4790 processor (3.60 GHz) with 8 GB registered DIMM DDR memory and 2 TB SATA-3 disk storage. All of the nodes were interconnected via 100 Mbps Ethernet adaptors. To implement LCM, we designated one node as our log collector server and used Java (Oracle JDK 64-bit 1.7.0_72) for LCM. To implement LGGM, we also designated one node as our graph generator server running Tomcat 7.0, with the server using JSP to implement the web-based dashboard and Google chart APIs to generate the graphs and tables showing the analyzed log information. MSM was implemented in one node, using MySQL 5.5.41 for real-time log information. In MSM, we used Oracle MySQL driver to connect MySQL with each module. To implement MdbM, we installed MongoDB 2.6.7 – 64-bit for accumulated log information on all the nodes, and we used a 10gen Mongo driver to connect our MongoDB with each module. In addition, to create a simulation environment for a customer's banking transaction, we generated log data including the eight parameters listed in **Table 1** using Java libraries. HadoopAM comprised one name node, with eleven data nodes running on HDFS with Hadoop 2.6.0.

4.2 MdbULPS Prototype

Fig. 6 shows a diagram of a web-based prototype of MdbULPS. Our web-based interface was developed with six jsp files. RealTimeView.jsp is used for LGGM to create graphs on real-time log information stored in MSM, sending the created graphs in MySqlView.jsp and showing them to users through a web interface. Moreover, GeneratorLog.jsp creates graphs on the average waiting time for processing one banking transaction and the number of waiting customers per time unit. Through use of CustomerProc.jsp, graphs for information on the number of completed banking business transactions by bank teller per hour, the average time spent on the banking business by the teller, and the teller's daily efficiency on the business are generated. The graphs generated by GeneratorLog.jsp and CustomerProc.jsp are sent to MongoView.jsp to be presented to users on the web interface.

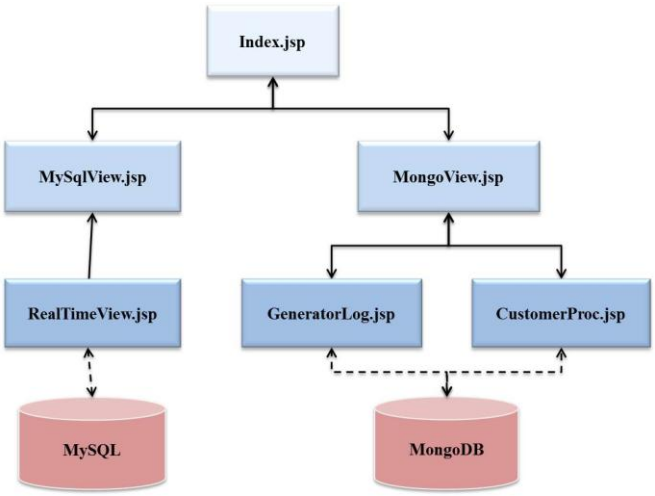


Fig. 6. Structure of the user web interface

4.2.1 Web page for Providing Real-time Log Information

Fig. 7 shows real-time log information generated by a customer's transaction at each corresponding bank in real time.

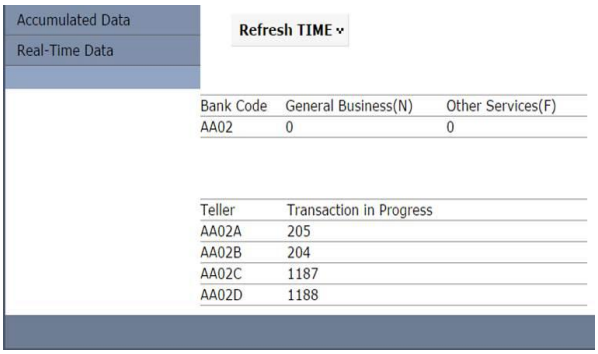


Fig. 7. Web-based dashboard for real-time log information

When the user accesses MySqlView.jsp, MSM provides real-time information regarding the current transaction number and the number of transactions waiting in the bank selected by users and administrators by extracting the information from log data stored in MySQL.

4.2.2 Web Page for Providing Accumulated Log Information on the Number of Waiting Customers

Fig. 8 illustrates the web page showing graphs and tables for accumulated log data generated after completion of a customer's transaction at each bank. Users and administrators can confirm large amounts of information, including the number of waiting customers per hour and the average waiting time for a custom's banking transaction in the specific bank they want to check. ① in **Fig. 8** shows in graph and table form the number of waiting customers for both general banking and other services during general banking hours (9 am to 4 pm) in a specific bank. The results, with the average customer waiting time for both services during the same time as ①, are shown in ② of **Fig. 8**.

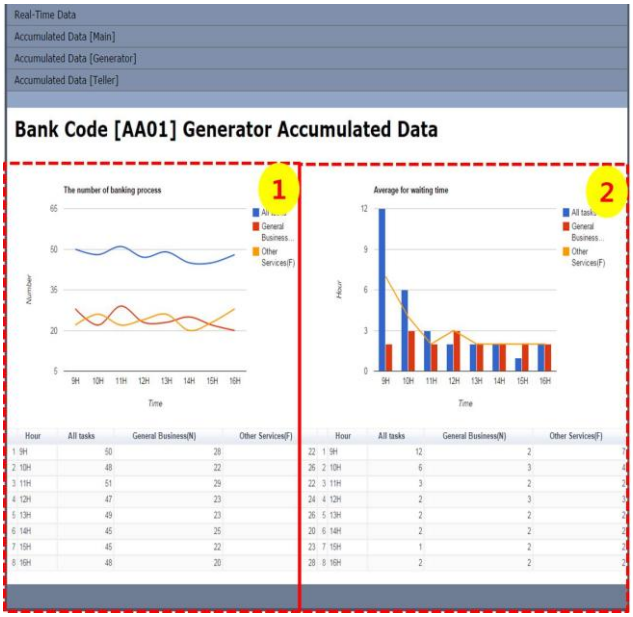


Fig. 8. Web-based dashboard for accumulated log data information of waiting time

4.2.3 Web Page for Providing Accumulated Log Information per Unit of Time

By accessing CustomerProc.jsp, users and administrators can monitor a large amount of information, including the banking business efficiency of bank tellers. The information is as follows: the number of banking transactions conducted by bank tellers per day and hour, respectively, and the average time spent by tellers to complete one baking transaction. The business efficiency of bank tellers is shown in **Fig. 9** in table and graph form. ① in **Fig. 9** shows a graph of the ratio of general business transactions and other service transactions conducted by a teller. ② in **Fig. 9** refers to a graph indicating an average throughput conducted by a teller per hour, and ③ shows the average processing time to complete one transaction.

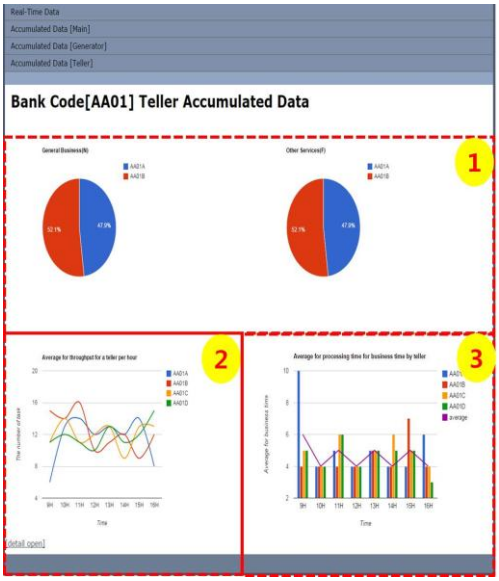


Fig. 9. Web-based dashboard for accumulated log information per unit of time

5. MdbULPS Performance Evaluation

5.1 Experimental Environment Description

We designed and implemented MdbULPS on a local test bed consisting of twelve nodes. Section 4.1 describes the cluster configuration and software specifications in more detail. To validate and verify the effectiveness and performance of our system, we conducted three different performance tests based on simulated log data generated by customer transactions at a bank. In fact, we created six types of log datasets including the nine parameters listed in **Table 1**. **Table 2** shows the datasets used in the performance evaluations. The size of one log for one customer transaction is 239 bytes.

Table 2. Log datasets used in the performance evaluations

Log dataset						
Size of dataset (GB)	1	2	4	8	16	32
Number of logs	429,280	8,585,360	17,170,720	34,341,440	68,682,880	137,365,760

Our three different performance tests are as follows. First, to compare our MongoDB-based approach with relational database-based approaches using MySQL, we measured insertion and query processing time for each dataset in the MongoDB- and MySQL-based approaches, respectively. Second, for comparing NoSQL-based approaches, we conducted a comparative performance that measured the total processing time for inserting and querying each dataset in the MongoDB- and Hbase-based approaches, respectively. Finally, we conducted a performance test measuring the total execution time for inserting datasets according to chunk size to confirm the optimized chunk size options of MongoDB.

5.2 Comparison of MdbULPS with a Relational Database-based Approach

In the first experiment, we measured the total processing time for inserting datasets and the time to complete 1,000 queries in the MongoDB- and MySQL-based approaches, respectively. To deploy the same performance environment of the MySQL-based approach, the tests for both approaches are conducted in one node because MySQL can run on a single node. We used MySQL 5.5.41 and MongoDB 2.6.7. To insert the datasets, we used the 'INSERT' command for MySQL, and the put( ) method for MongoDB. For querying the datasets, we used 'select * from bank, where bank_code=AA24B' for MySQL, and the command line shown in **Fig. 10**.

Command lines for querying each dataset in MongoDB-based approach

```
1: BasicDBObject  query = new BasicDBObject ( ) ;
2: Query.put("bank_code", "AA24B"));
3: dbCollection.find(query);
```

Fig. 10. Example of command lines for query in MongoDB-based approach

Table 3 lists the results of the performance tests for inserting and querying each dataset, and **Fig. 11** shows the total insertion processing time in both approaches. As can be seen from the table and figure, our MongoDB-based system delivered better performance than the MySQL-based relational database-based approach in terms of insertion and query processing time. For example, in the test for inserting each log dataset, we found that our approach performed approximately two times faster in inserting each dataset than the MySQL-based approach. In fact, for a 32 GB dataset, our approach required 9,465 s (approximately 157 min) to complete inserting a dataset into our system, while the MySQL-based approach required 18,231 s (approximately 304 min). In the querying performance test, our system delivered strikingly better performance than the MySQL-based approach. For example, for a 32 GB dataset, our system required 0.225 s to complete 1,000 queries, while the MySQL-based approach required 69.542. Our approach's performance was approximately 393 times faster than that of the MySQL-based approach.

Table 3. Total processing time(s) for query and insertion in both database-based approaches

Dataset	MongoDB-based Approach		MySQL-based Approach	
	Insert	Query	Insert	Query
1 GB	256	0.127	508	59.612
2 GB	544	0.129	940	60.492
4 GB	1091	0.132	2054	64.282
8 GB	2582	0.159	4288	65.433
16 GB	4835	0.213	8623	67.844
32 GB	9465	0.225	18231	69.542

The reason the proposed system showed such a strikingly better performance than the MySQL-based system is as follows. Our system reduced the processing time for query and insertion exponentially by storing and processing unstructured log data without requiring standardization procedures to be performed every time, as a relational database would require, because our system applies the flexible and expandable schemata featured by MongoDB. Excellent querying performance can be achieved owing to the fact that our system can use the MongoDB memory mapping technique. Finally, such a performance was delivered because, in contrast to a relational database, MongoDB does not perform a transaction procedure.

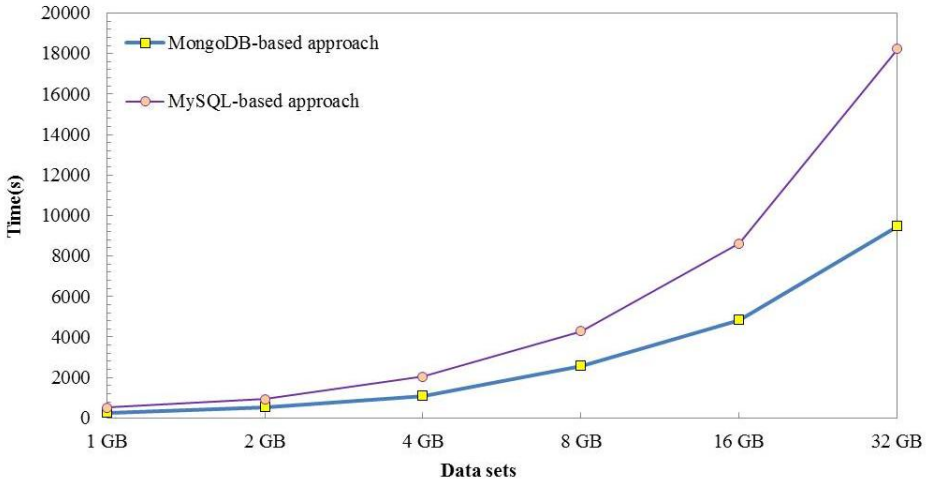


Fig. 11. Performance results of both database-based approaches for inserting each dataset

5.3 Comparison of MdbULPS with NoSQL (Hbase)

In the second experiment, to compare our approach with a different NoSQL-based approach, we tested and measured the total processing times for inserting and querying each dataset. This performance was conducted on our own twelve-node cloud cluster. Other aspects of the environment, such as software specification, manner of measuring total processing time, and datasets, were the same as for the test conducted in section 5.2. To conduct the NoSQL-based approach, we used Hbase [24, 25], version 0.98.9 provided by Apache Hadoop. Furthermore, we used the Hbase Filter class and the command lines shown in **Fig. 12** for querying each dataset.

Command lines for querying each dataset in Hbase-based approach

```
1: Filter filter = new SingleColumnValueFilter(Bytes.toBytes("cf:teller"), null,
   CompreOP.EQUAL.Bytes.toBytes("AA24B"));
2: Scan s = new Scan ( );
3: s.setFilter(filter);
4: ResultScanner rs = table.getScanner(s);
```

Fig. 12. Example of command lines for querying in Hbase-based approach

Table 4. Total processing time(s) for query and insertion in both database-based approaches

Dataset	MongoDB-based approach		Hbase-based approach	
	Insert	Query	Insert	Query
1 GB	231	0.124	564	3.358
2 GB	531	0.126	1281	3.577
4 GB	1029	0.130	2475	3.688
8 GB	2038	0.128	4899	3.944
16 GB	4287	0.123	10237	4.018
32 GB	8950	0.122	21495	4.563

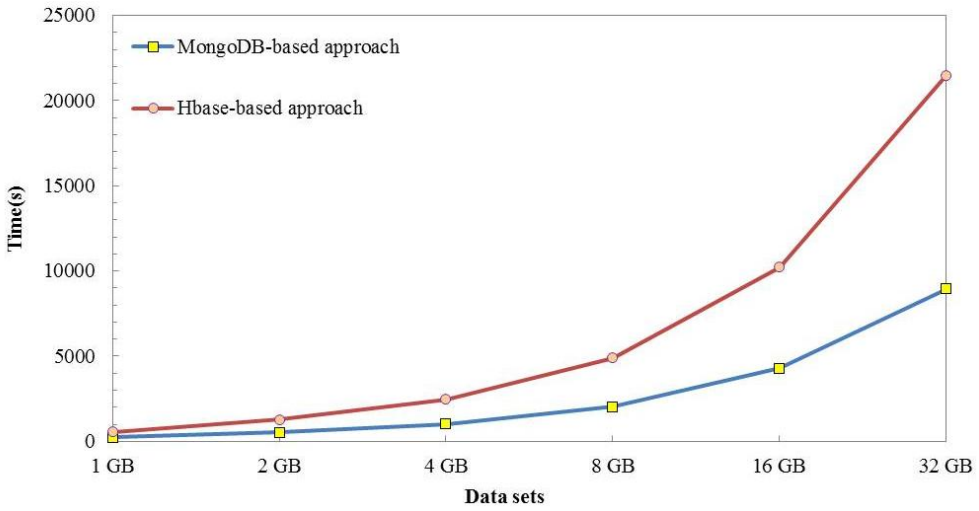


Fig. 13. Performance results of both database-based approaches for inserting each dataset

Table 4 shows processing time performance for our approach and the Hbase-based approach for the insert and query functions. **Fig. 13** illustrates the results for inserting each dataset in both approaches. The results of the performance are similar to those of the test in Section 5.2. In fact, the performance in inserting each dataset was approximately 2.4 times faster than that of the Hbase-based approach. Although both database-based approaches are a kind of NoSQL database, a better performance result was shown with our approach, because a column-based database such as Hbase is inappropriate for processing log data of the document type. In addition, for measuring the querying of each dataset, our approach also showed better performance than the Hbase-based approach. In fact, our system was 30 times faster than that approach, in terms of completing the querying of each dataset. The reason why the MongoDB-based approach outperformed the Hbase-based approach is that the Hbase conducts a full table scan each time users require queries. From the results, we found that our system clearly exhibited a better performance than the Hbase-based approach provided by Apache Hadoop in terms of system reliability, with inserts and queries for analyzing a large volume of log data**.** On the other hand, even though the Hbase-based approach performed less well for querying than our approach, it showed better performance than the MySQL-based approach, because Hbase running on shard servers or master and slaves uses distributed and parallel processing for querying and analyzing stored data.

5.4 Changing the Chunk Size Factor

In the third experiment, we measured the total elapsed inserting time using different MongoDB options for the chunk size. MongoDB processes large portions of datasets in a parallel and distributed manner, after the datasets are split into chunks of size 64 MB. However, by inputting the following command line on the MongoDB shell, `db.settings.save( {_id:"chunksize", value: <sizeInMB>} )`, users and administrators can change the chunk size options to improve data processing performance, depending on the size and type of unstructured data. Thus, to verify that chunk size options affected performance, we measured the time required to complete inserting for a 1 GB dataset. Seven chunk size options were used in the experiment, viz., 32, 64, 128, 256, 512, and 1024 MB. **Fig. 14** shows the total processing times required for inserting datasets with different chunk size values. According to **Fig. 14**, we found clearly that the performance of our system was best when the chunk size was set to 64 MB.

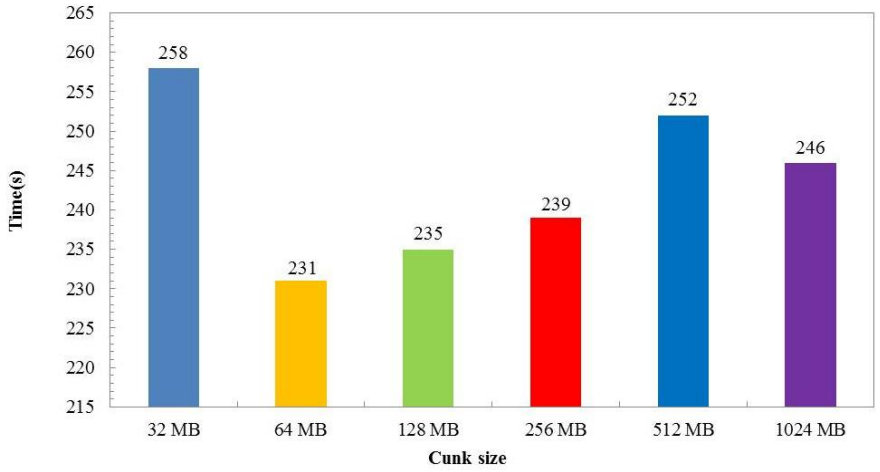


Fig. 14. Total processing time for inserting 1 GB dataset with various chunk size factors

Sharding is a method for storing data across multiple machines in order to avoid exceeding the storage capacity of a single machine, and to conduct distributed queries. MongoDB in the proposed system utilizes an auto-sharding function that distributes and balances log data from banks in distributed shard servers or nodes. It supports the auto-sharding function through the configuration of a sharded cluster that consists of three components: shard servers, mogos routers, and configuration servers.

When the log data grows beyond the specified chunk size (default: 64 MB) configured by administrators and users, MongoDB in the proposed system attempts to split the log data into chunks of this size (e.g., 64 MB) and then stores these chunks in shard servers. Our system has four shard servers, and each server includes three replica sets. The auto-sharding function involves the following sharded cluster processes: sharded collection balancing, chunk migration across shard servers, chunk splits in a sharded cluster, sharded key indexes, and sharded cluster metadata.

The reason why this factor affects performance in inserts when changing the chunk size is that I/O overhead is generated during the chunk migration process. If the chuck size factor is set to less than 64 MB, performance is degraded because I/O overhead results from more

frequent migrations between each shard server. The other reason is the increased management cost generated in the sharded cluster metadata process. In fact, after the chunks migrate to the sharded servers, the configuration servers update and store the metadata, including the list of chunks on each shard server and the ranges that define these chunks. That is, because the number of chunks increases, there is more overhead in the configuration servers. In addition, although there are fewer migrations in the opposite case, overhead results from the perspective of networking, because existing chunks of 64 MB remain larger until they reach the newly configured size. Therefore, performance is degraded when the chunk size is set to more than 64 MB.

6. Conclusion

After identifying limitations to the processing and analyzing of unstructured log data through relational databases such as MySQL, we focused on designing MdbULPS in a cloud-computing environment including MapReduce, Hadoop, and MongoDB to collect, categorize, and analyze unstructured log data generated from banks. The proposed system uses a cloud environment to provide a flexible expansion of computing resources and includes the ability to expand resources such as storage space and memory in a flexible manner, under conditions such as extended storage or rapid log data increase. Moreover, to overcome the processing limits of the existing analysis tool, when a batch analysis of the aggregated unstructured log data is required, the proposed system includes a Hadoop-based analysis module for quick and reliable parallel-distributed processing of a massive amount of log data. Furthermore, because the HDFS stores data by generating copies of the block units of the aggregated log data, the proposed system offers automatic restore functions for the system to operate continually after it recovers from a malfunction. Finally, through a distributed database using the NoSQL-based MongoDB, the proposed system provides methods of effectively processing unstructured log data.

To validate the performance of the proposed MdbULPS, we conducted three different tests on our local test bed consisting of twelve nodes to compare our system with relational database-based and Hbase-based approaches with regard to inserting and querying each dataset and identifying the optimal chunk size options. From the experiments, we confirmed that our system exhibited better performance with inserting and querying each dataset compared with the others. In addition, we found that our system performed best when the chunk size was set to 64 MB. The reason the performance is affected by the chunk size is the I/O overhead generated during the chunk migration process.

In future research, we plan to implement our system in a real cloud-computing environment, such as Amazon EC2, Rackspace, or KT cloud, using real log data generated for banks, rather than simulated log data. In addition, we will handle various log data such as traffic, stock, online shopping mall, and security log data by using our system.

References

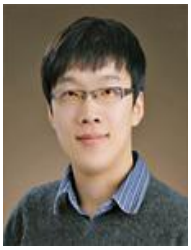
[1] H. Lee, “Design and implementation of web attack detection system based on integrated web audit data,” *Review of Korean Society for Internet Information*, vol. 11, no. 6, pp. 73-86, 2010.

[2] T. Yoon, S. Lee, K. Yoonm, and J. Lee, “Design and application of multiconcept keyword model based on web-using information,” *Review of Korean Society for Internet Information*, vol. 10, no. 5, pp. 95-1105, 2009.

[3] U. Park, "A database schema integration method using XML schema," *Review of Korean Society for Internet Information*, vol. 3, no. 2, pp. 39-56, 2002.
[4] D. Agrawal, S. Das, and A. EI Abbadi, "Big data and cloud computing: current state and future opportunities," *EDBT/ICDT '11* in *Proc. of the 14th International Conference on Extending*, pp. 503-533, 2011. Article (CrossRef Link)
[5] H. Yu, and D. Wang, "Mass log data processing and mining based on Hadoop and cloud computing," in *Proc. of Computer Science & Education (ICCSE) 2012 7th International Conference on*, pp. 197-202, 2012. Article (CrossRef Link)
[6] N. Leavitt, "Will NoSQL databases live up to their promise?," *Computer*, vol. 43, no. 2, pp. 12-14, 2010. Article (CrossRef Link)
[7] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proc. of Pervasive Computing and Applications (ICPCA) 2011 6th International Conference on*, pp. 363-366, 2011. Article (CrossRef Link)
[8] R. Hecht, and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *Proc. of Cloud and Service Computing(CSC) 2011 International Conference on*, pp. 336-341, 2011. Article (CrossRef Link)
[9] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *iiWAS '11* in *Proc. of the 13th International Conference on Information Integration and Web-based Applications and Services*, pp. 278-283, 2011. Article (CrossRef Link)
[10] M. Stonebraker, "SQL databases v. NoSQL databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10-11, 2010. Article (CrossRef Link)
[11] Z. Wei-Ping, L. Ming-Xin, and C. Huan, "Using MongoDB to implement textbook management system instead of MySQL," in *Proc. of Communication Software and Networks (ICCSN) 2011 IEEE 3rd International Conference on*, pp. 303-305, 2011. Article (CrossRef Link)
[12] S. Lombardo, E. Di Nitto, and D. Ardagna, "Issues in handling complex data structures with NoSQL databases," *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) 2012 14th International Symposium on*, pp. 443-448, 2012. Article (CrossRef Link)
[13] NoSQL, http://en.wikipedia.org/wiki/NoSQL
[14] MongoDB, http://www.MongoDB.org/
[15] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM - 50th Anniversary Issue: 1958 – 2008*, vol. 51, no. 1, pp. 107-113, 2008. Article (CrossRef Link)
[16] K. Shavchko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," *Mass Storage Systems and Technologies (MSST) 2010 IEEE 26th Symposium on*, pp. 1-10, 2010. Article (CrossRef Link)
[17] J. Shi, H. Li, Y. Hu, and E. Huang, "Research on key technologies of cloud computing," *International Journal of Digital Content Technology and its Applications*, vol. 6, no. 20, pp. 438-445, 2012. Article (CrossRef Link)
[18] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no 2, pp. 171-209, 2014. Article (CrossRef Link)
[19] X. Wu, X. Zhu, G. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97-107, 2014. Article (CrossRef Link)
[20] C. Ji, Y. Li, W. Qiu, Y. Jin, Y. Xu, U. Awada, K. Li, and W. Qu, "Big data processing: Big challenges and opportunities," *Journal of Interconnection Networks*, vol. 13, no 3-4, article no. 125009, 2012. Article (CrossRef Link)
[21] Y. Liu, Y. Wang, and Y, Jin, "Research on the improvement of MongoDB Auto-Sharding in cloud environment," *in Proc. of International Conf. on Computer Science and Education*, pp. 851-854, 2012. Article (CrossRef Link)
[22] X. Wang, H. Chen, and Z. Wang, "Research on improvement of dynamic load balancing in MongoDB," *in Proc. of 11th International Conf. on Dependable, Automatic and Secure Computing*, pp. 124-130, 2013. Article (CrossRef Link)
[23] K. N. Duan, "Migration of stored procedure to distributed cloud database," *in Proc. on Advances in Materials Science and Information Technologies in Industry*, pp. 513-517, 2014.

Article (CrossRef Link)
[24] D. Borthakur, et al., "Apache Hadoop goes realtime at Facebook," *in Proc. of the ACM SIGMOD*, pp. 1071-1080, 2011. Article (CrossRef Link)
[25] M. N. Vora, "Hadoop-Hbase for large-scale data," *in Proc. of 2011 International Conference on Computer Science and Network Technology*, pp. 601-604, 2011. Article (CrossRef Link)
[26] Z. Liu, Y. Wang, and R. Lin, "A novel development and analysis solution to PaaS log by using CouchDB," *in Proc. of IC-NIDC 2012,* pp. 251-255, 2012. Article (CrossRef Link)
[27] Apache Hadoop, http://hadoop.apache.org/
[28] J. Dean, and S. Ghemawat, "MapReduce: A flexible data processing tool," *Communication of the ACM*, vol. 53, no. 1, pp. 72-77, 2010. Article (CrossRef Link)
[29] M. Y. Eltabakh, et al., "CoHadoop: Flexible data placement and its exploitation in Hadoop," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, pp. 575-585, 2011. Article (CrossRef Link)
[30] Y. Zhang, "Web data mining technology on cloud computing," *Applied Mechanics and Materials*, vol. 543-547, pp. 3490-3493, 2014. Article (CrossRef Link)
[31] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle – Database comparison, *in Proc. of 3rd International Conference on Emerging Intelligent Data and Web Technologies,* pp. 330-335, 2012. Article (CrossRef Link)
[32] M. Kim, S. Han, Y. Cui, H. Lee, H. Cho, S. Hwang, "CloudDMSS: Robust Hadoop-based multimedia streaming service architecture for a cloud computing environment," *Cluster Computing*, vol. 17, no. 3, pp. 605-628, 2014. Article (CrossRef Link)
[33] Apache Storm, https://storm.apache.org/
[34] Apache Spark, https://spark.apache.org/
[35] Apache Kafka, http://kafka.apache.org/

Myoungjin Kim received M.S. and Ph.D. degrees from Konkuk University, Seoul, Korea, in 2009 and 2015. Currently, he is a postdoctoral researcher at the Social Media Cloud Computing Research Center. His research interests include distributed computing, distributed real-time programming, MapReduce, media transcoding and cloud computing.



Yun Cui received an M.S degree from the department of Internet and Multimedia Engineering at Konkuk University, Korea,. At present, he is a Ph.D. student and an assistant researcher at the Social Media Cloud Computing Research Center. His current research interests include cloud computing, social network services, home network services, and distributed computing.

Hanku Lee is the director of the Social Media Cloud Computing Research Center and a full professor at the division of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea. He received a Ph.D. degree in Computer Science from Florida State University, USA. His recent research interests include cloud computing, distributed real-time systems, distributed computing, and compilers.