NoSQL Database Modeling for End-of-Life Vehicle Monitoring System

Joohyoung Jeon, Minjeong An, Hongchul Lee
Lab of Information Network, School of Industrial and Management Engineering, Korea University, Seoul, Korea.

* Corresponding author. Email: hclee@korea.ac.kr
Manuscript submitted April 4, 2015; accepted August 25, 2015.
doi: 10.17706/jsw.10.10.1160-1169

Abstract: Korea was legally set to car recycling rate to 95% from 2015. Also EU has enacted the law for car recycling. Accordingly, the monitoring system that stores and manages the data generated by the End of Life Vehicle (ELV) are needed. The parts information of the ELV is quite a lot, the information contained sub-components are very many. When building a monitoring system database on an RDBMS, there are some limitations such as constraint of the storage capacity, wasting space, and complex operations. The aim of this paper is to present and implement a NoSQL Database system using MongoDB in order to deal and manage huge data and information. At the end of the paper, we analyze the performance difference between the proposed database system and the existing database system using the statistical analysis.

Key words: Big data, NoSQL database, MongoDB, paired t-test.

1. Introduction

The automobile industry is called the parable of flowers of the machinery industry. It is an important factor of national economic growth. In 2014, the number of vehicle registration in Korea has reached 20 million. Also the number of ELVs is increasing. With the continued growth of the automobile industry, the international discussions on the automobile recycle are increase. For example, EU is recommended to recycle 95% of the cars. Also Korea enacted the Resource Recycling Act.

The ELV monitoring system was made by these backgrounds [1]. It collects parts information generated by the dismantling process. The ELV monitoring system is made up of Relational Database System. RDBMS is considered the ACID properties of data; it is an advantage to maintain a data consistent and integrity. However, if part information generated by the single vehicle has a sub-information again, to maintain data consistency and integrity, many redundant data are stored in the RDBMS. Considering this duplicate data problem of RDBMS, the database of the ELV monitoring system is to be improved. NoSQL has emerged as an alternative to solve the limitations of the RDBMS. It allows easy and efficient storage of large amounts of data. In this paper, we present the ELV monitoring system using a NoSQL database.

The remainder of the paper is organized as follows. Section 2 discusses the types and characteristics of NoSQL Databases. It allows selecting the suitable NoSQL database for ELV Monitoring system. Section 3 describes system overview and data architecture. In section 4, we implement MongoDB database. Section 5 shows the experimental results both database systems using the *Paired t-test*. Section 5 concludes the paper.

2. Backgrounds

No SQL (Not Only Structured Query Language) is no need pre-defined schema and will not need the join

operation between data. It also provides easy horizontal-scale up and the read and write performance are fast compared to the RDBMS. In particular, it is suitable for distributed system environment.

2.1. Theoretical Classification of NoSQL

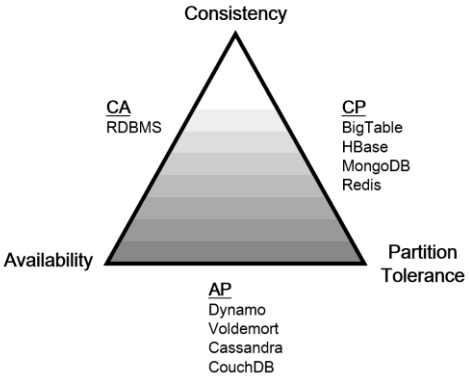

Fig. 1. CAP theory.

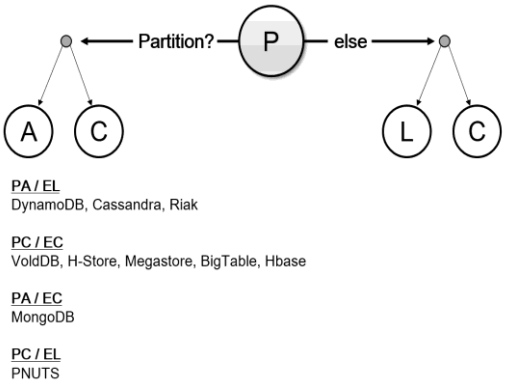

Fig. 2. PACELC theory.

Fig. 1 is the classification of NoSQL under the CAP theory [2]. CAP theory was classified according to the functions of a distributed system that is selected in the event of a failure in the distributed system. Consistency and Availability are a feature of the distributed system, Partition-tolerance means a network characteristic of a distributed system. In other words, Partition-tolerance refers to when a failure in the distributed system, the network is allowed to do any message losses. Thus, if the user does not select P which will not recognize the network failure. That means a loss of a message will not occur, in any case. Since the implementation of this distributed system is impractical, Partition-tolerance is always selected. The limitation of CAP theory is that is not classified the NoSQL in normal network conditions [3].

PACELC theory pointed out the limitations with the CAP theory. Fig. 2 shows a PACELC theory. It was classified according to the function of selecting a distributed system network when the network is normal and abnormal conditions [4]. In PACELC theory classifies the presence of network failure (Partition) a top priority. Network failure (Partition) occurs, Availability and Consistency are a trade-off, and network will choose the one or the other. Whereas if Partition is not occurred (in normal conditions), Latency and Consistency are trade-off.

For example, Fig. 2 PA/EL System, in the partition situation, gives up Consistency in order not to lose Availability. Otherwise, in the normal condition, also give up Consistency in order not to lose Latency. According to PACELC theory, we chose MongoDB classified as PA/EC database system. Users find the data in real time using the ELV monitoring system. Therefore, in Partition situations, database selects the Availability, to avoid interference with some failure node and other nodes. Whereas in a normal network condition it selects the Consistency to improve the data reliability of the monitoring system.

2.2. Functional Classification of NoSQL

Table 1. Functional Classification of NoSQL

Function	Name
Column/ Column Families	Hadoop/HBase, Cassandra, Hypertable, Accumulo, SimpleDB, Cloudata, MonetDB, Flink, Informix *et al.*
Document-oriented Store	Elasticsearch, MongoDB, Couchbase, RethinkDB *et al.*
Key Value/Tuple Store	DynamoDB, Azure Table Storage, Riak, Redis, Aerospike *et al.*
Graphic Database	Neo4J, Infinite Graph, Sparksee, TITAN, InfoGrid *et al.*

Table 1 is a functional classification of NoSQL. First, Column/Column Families will query the data in column basis. Thus, it is useful for frequent updates column unit date database system. Second, Document-oriented database system is useful for storing data that do not require a certain type. It do not need a pre-defined schema. Third, Key Value/Tuple NoSQL Database that is manages the date using a pair of key-value. You can see the value by using the key. In particular, it is easy to apply the Key Value Type DB in the ability to save the results from the RDBMS. However, it is vulnerable to a range search. Fourth, Graphic Database is NoSQL Database for storing graphic information [5].

Using the reference function of MongoDB, we can store multiple collections in single collection. These characteristics are well suited for the data structure of the tree type. For example, consider the actual dismantling process. The number of dismantled parts is generated in a single ELV. And it has a number of sub-property information. Hence, MongoDB is suitable for storing and querying this tree type data structure.

2.3. Database Requirement

The requirements of the database are as follows: in the End of Life Vehicle monitoring system.

(1) Expressivity

(2) Processing

Expressivity. Part information of the ELV has a sub-information again. Sub-information includes a lot information. Such as the official parts name, commonly used parts name, the number of child components, dismantling station, material weight information, *et al*. Such information should be easily expressed for user.

Processing. It is necessary to make it easier to store and analyze large amounts of data. In case of storage space constraints, by utilizing the Auto-Sharding function, the database stores data automatically in distribution system. In addition, it is support the data analysis using Map/Reduce. Also, in the future, it should support the cloud system. As a result, we chose MongoDB. Because it meets two requirements.

3. System Modeling

3.1. A Brief Introduction to MongoDB

MongoDB is a document-oriented NoSQL database, was written by the C++ language. It manages collections of BSON documents. It does not need a pre-defined schema. Further, without having a pre-defined schema, it is possible to search for the Key value through the function. Therefore, the response speed is fast compared to the relational database and it is useful to search the necessary information at the same time [6]. Since pre-defined schema is not required, the application establishes a data structure. It is an advantage to shorten the initial development time in the data structure changes frequently [7].

Also MongoDB can represent a variable data. If the variable data such as raw material content is included in the collection, it can be changed easily. Replication is a strong advantage of MongoDB. It stores the data automatically in distributed system in case of network failure conditions. And it is possible to store more data and handle more load without requiring large or powerful machines [8].

3.2. System Outline

Fig. 3 is a system outline according to a dismantling process system. Dismantling procedure is shown by solid line, the flow of data is shown by a dotted line. Rectangular refers to the working space or a data space. Ellipse shape means the work.

Markets are divided into the car market and used parts market. The ELV occurs in the car market. Generated ELV are stocked a junkyard in certain areas. Stocked ELV are classified according to vehicle conditions. The vehicle is dismantled according to the legal process [1]. The part from the dismantled vehicle is re-classified according to their characteristics. Compressed body is recycled by shredding. And

reusable part is released to the used part market through the cleaning and repacking depending on their state.

Approaching to the system perspective, the initial data are stored with the stocked ELV at the same time. The initial data is mainly general information for the ELV. According to the dismantling procedure, the system stores the dismantled part information in the database. In this process, sub-part information, such as state of dismantled part, weight, manufacturer, and the details of the configuration information are stored to the sub-dataset. The user must be able to query the data via the monitoring system. In this paper, we converted the database of the monitoring system as NoSQL database.

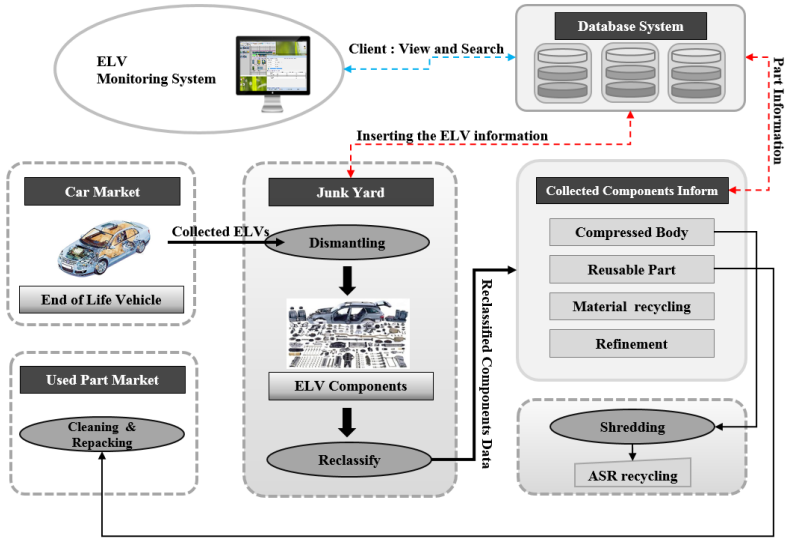

Fig. 3. System outline.

3.3. Data Architecture

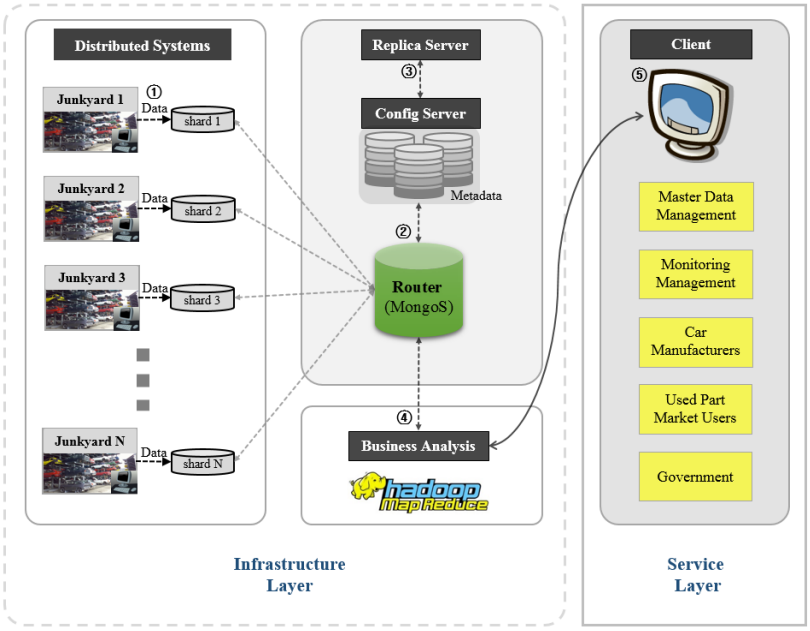

Fig. 4. Data architecture.

Fig. 4 shows the data architecture. The monitoring system is divided into Infrastructure Layer and

Service Layer. The Infrastructure layer mainly use a Sharding (distributed storage) and Replica Set (replication) of the functions of MongoDB. The distributed systems area stores the information saved from each junkyard.

The Configuration Server set the each junkyard as a Shard server. It is set the properties of the Shard Server and manage their data. The Configuration Server automatically generate a Replica Server which is a kind of backup server. Replica Server automatically recovers data when any data is arbitrary deleted and an error occurs in the Shard Server or Configuration Server.

Service layer provides the information of stored and managed in monitoring system to the client. It can be divided into five areas. First, *Master Data Management* manages all server areas consisting of MongoDB. Second, *Monitoring Management* is responsible for the web application. Third, *Car Manufacturers* are provided with information in a monitoring system. Fourth, *Used Part Market Users* are composed of individual users, they viewed the price information of used parts, part status, and merchant information using a web application in monitoring system. Finally, *Government* is provided with a variety of information of the ELV, such as environmental regulations materials generated by ELV processing and car recycling rate.

4. Implementation

4.1. Design of MongoDB Collections

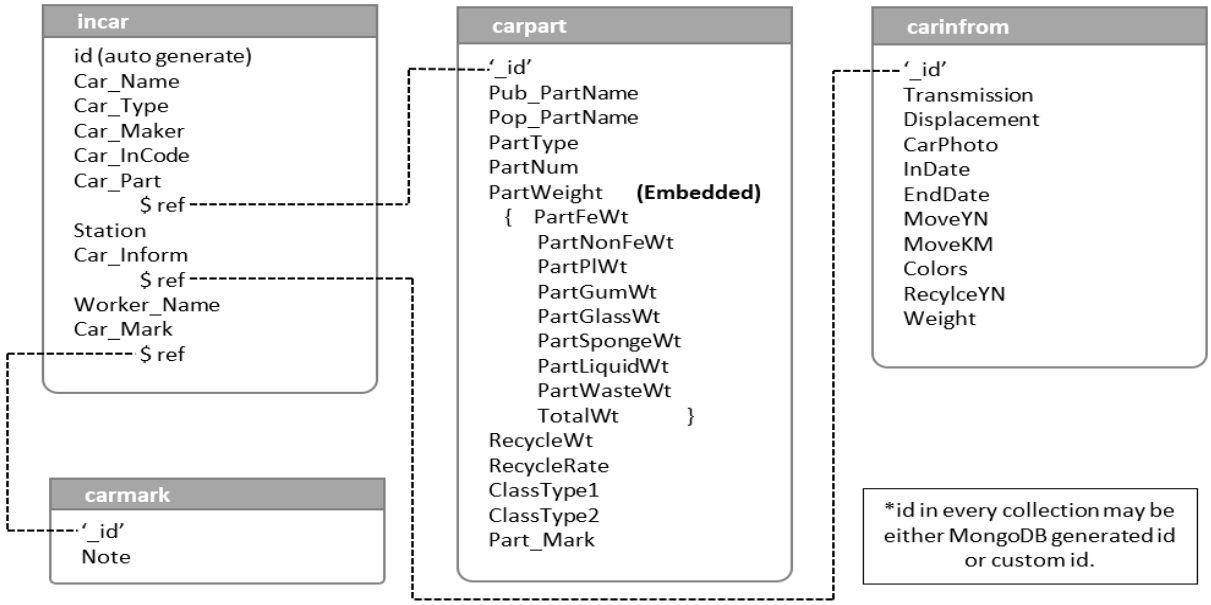

Fig. 5. The design of MongoDB collections.

"Fig. 5" is the design of MongoDB collections [9]. And the following is a description of the collections.

Incar collection stores the information of the stocked ELV. The '_id' is a unique string that is automatically generated by MongoDB. *"Car_Part"* is connected with *"carpart collection"* using the reference function. The *"Car_Inform"* is connected with *"carinform collection"* that stores basic information about the ELV. In addition, the *"Car_Mark"* is linked with *"carmark collection"* that stores the workers handwritten records.

Carpart collection contains all parts information generated by dismantled ELVs. It was storing data in an array. *"PartWeight"* stores weight information in accordance with the raw material properties (Fe, Pl, Gum, Glass, Sponge, and Liquid *et al.*) in an embedded array.

Carinfrom collection stores default information of the stocked car. These are transmission information,

vehicle photo, stocked date, end day of dismantled, mileage, year running such as basic information.

Carmark collection stores the Text information recorded directly to the ELV dismantling workers.

4.2. Data Insert

The development environment is as follows.

CPU: Intel® i3-4150, RAM: 8G, OS: windows 7-64bit.
MongoDB version is 3.0.1 (released in 3/17/2015)

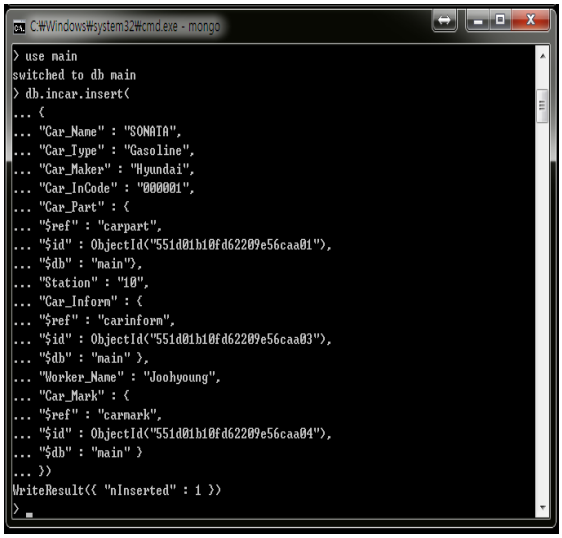

Fig. 6. Insert incar collection.

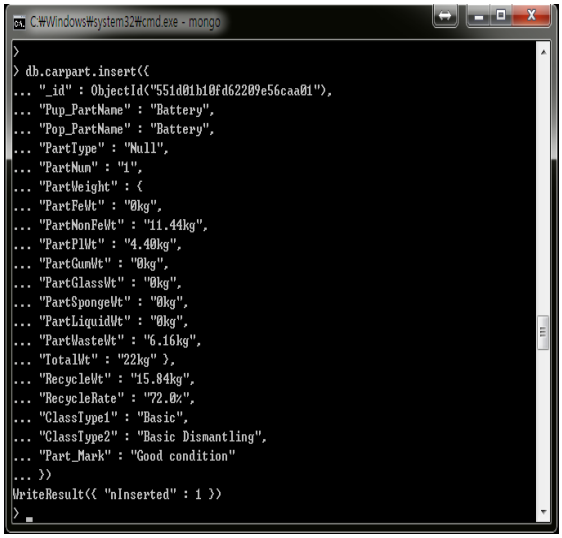

Fig. 7. Insert carpart collection.

Fig. 6 shows a creating the "*Incar Collection*". *"Car_Part"*, *"Car_Infrom"*, and *"Car_Mark"* are the name that defines the collection for reference. MongoDB's collection is connected to reference using the $ref, $id, and $db. In particular, the '_id' of *"Incar collection"* did not enter separately because it is automatically generated by the MongoDB. If the user wants to directly give the '_id', you can use the *createCollection* instructions.

On the other hand, each *ObjectId* in {"$id"} will be defined in advance by user. This is one of the main characteristics of MongoDB, user can pre-define the *ObjectId* of the collection for reference before it generated.

Fig. 7 shows a *"Carpart Collection"*. It stores the dismantled part information from the ELV, as explained above. *"Pup_PartName"* means the officially part name, *"Pop_PartName"* means the commonly used parts name. Feature of the *"Carpart collection"* that is to store part material information in the embedded array form. For the part material information may be changed time to time. At this time, it is possible to replace only the changed information through the atomically update.

4.3. Data Query

MongoDB supports simple query. Basically, the join operation is not supported in MongoDB. But user can easily find specific information from the other collections. In addition, they can modify only the information that need to be modified without querying the total collection's information. In this section, we only use the car name; we get the information from the other collection which is connected with it.

Fig. 8 is a query that finds the *"Car_Part"* information through *"Car_Name"* (*SONATA*) in *"Incar Collection"*. Use this query when you want to find the dismantled part information from the *"Carpart collection"*.

```
var sonata = db.incar.findOne({"Car_Name":"SONATA"})
```

This variable is to find the car name from the *"Incar collection"* called *"SONATA"*.

```
var dbRef = sonata.Car_Part
```

It defines a variable called *dbRef*, which defines a reference collection. As we see in Fig. 6. When we create the *"Incar collection"*, *"Car_Part"*, $ref, and $id were pre-defined to reference collection name and *ObjectId*. By adding the *"Car_Part"* name to the sonata variable, we find the *SONATA's* dismantled part information.

```
db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

Finally, by using the above query, we find the dismantled part information of the *SONATA*.

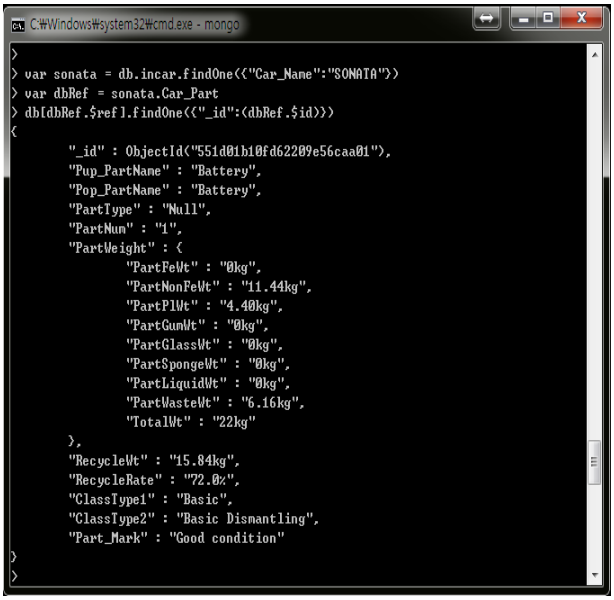

Fig. 8. Query result of finding SONATA from "Carpart collection".

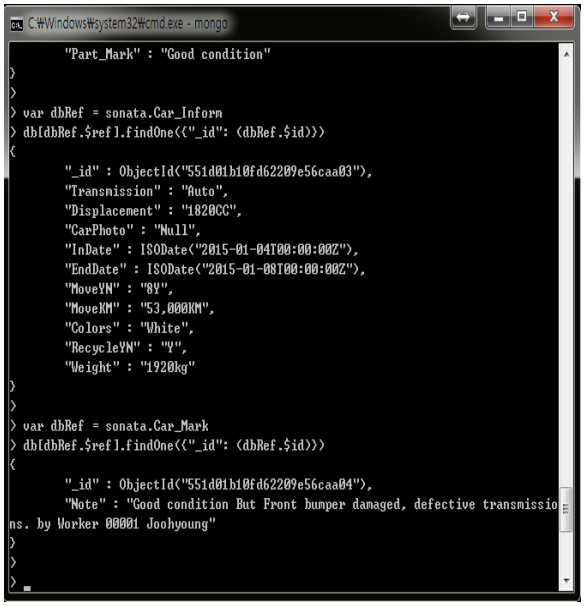

Fig. 9. Query result of find SONATA from "Carinform" and "Carmark" collections.

Fig. 9 is the query result of finding the car information and worker's written record from *"Carinform collection"* and *"Carmark collection"*.

5. Experimental Analysis

The experiment was performed in conditions of Windows 7 64Bit, 8GB Memory, and we use MySQL (5.6.20 Community Server), MongoDB (3.0.1), and Eclipse (java version 1.8.0_25). Performance difference of the both systems was measured by query time in milliseconds based on the Java environment.

Table 2 shows the query statement used as the measurement basis. Insert, Find, Join, and Update are mainly used query in database system. Therefore, we set these operations on the basis of the measurement.

We measured the start and end processing time of each query statements in both database systems. In order to confirm the performance difference between the two database systems, in accordance with the data increase, we experimented with increasing the same query from 1 time, 100 times, and 10000 times.

Based on the processing time, the *Paired t-test* was used to compare the performance of the both systems. For Paired t-testing, the experimental results from the both systems meet the *I.I.D* (Independent and Identically Distributed random variables) conditions. The experimental conditions for satisfying *I.I.D* are as follows.

First, Each DB system is implemented on the same computing environment, query time was measured at a different time. Second, the experiment was replicated 21 times, the result of the first test data was excluded from the measurement, taking into account that the data after the server is running. Third, measuring the query time was performed independently without restarting the server and any other programs. Fourth, the experiments were carried out after a certain period of time after the server is operating.

Table 2. Query Statement used to Measure the Performance

Query Statement	MySQL	MongoDB
Insert Data	INSERT INTO '(column Name)' VALUES '(Attribute)'	db.incar.insert({"names":"attributes"})
Find Data	SELECT all FROM 'incar'(table name)'	db.incar.find( )
Join Operation	SELECT all FROM incar's id and carpart's id	db.findOne( ) *Use *DBRef* method in java
Update Data	UPDATA incar table's Car_Type	db.update(searchQuery, updateQuery)

Table 3. Testing Result

Query type	Document(s) found	Query Time in milliseconds*	
		A Paired-t Confidence Interval (Significant difference at the 0.05 level)	A Paired-t Confidence Interval (Significant difference at the 0.01 level)
Insert Query	1 time	[28.533 41.767]	[26.105 44.196]
	100 times	[-2821.076 -2673.725]	[-2848.110 -2646.690]
	10000 times	[-330016.379 -312070.621]	[-333308.864 -308778.136]
Find Query	1 time	[66.150 74.251]	[64.663 75.737]
	100 times	[152.731 179.669]	[147.788 184.612]
	10000 times	[2303.689 2566.711]	[2255.433 2614.967]
Join Query	1 time	[69.585 75.415]	[68.516 76.484]
	100 times	[79.464 113.636]	[73.195 119.905]
	10000 times	[2270.068 2798.032]	[2173.203 2894.897]
Update Query	1 time	[88.952 97.348]	[87.412 98.888]
	100 times	[52.374 68.326]	[49.447 71.253]
	10000 times	[-876.978 -693.722]	[-910.599 -660.101]

*1 seconds = 1000 milliseconds
Performance difference = MongoDB – MySQL
The positive number means that MongoDB's query time is taking longer than MySQL's query time.

Table 3 shows the experiment result. According to the result, MongoDB system shows a remarkable performance in the Insert and Update Query. The data to be inserted is increased, the difference in performance of both systems is evident. On the other hand, MongoDB took longer than MySQL in the Find and Join query.

We were able to know the powerful functions of MongoDB from the experiment. It is useful for storing data that is increasing exponentially. MongoDB has a difference in performance compared to the existing

database technology. Whereas, Find and Join query in the existing database still showed a better performance.

6. Conclusion

Using the MongoDB database system proposed in this study, it is possible to easily store and manage dismantled parts information in the ELV monitoring system. In addition, this distribution database system can be stored in low-cost PC with shard functions of MongoDB. With this horizontal-scale up, it can deal with a problem that data increases explosively. The limitations of the experiment are that the query time is measured on a single node. Therefore, future studies will include a query time measured in a distributed system environment. In addition, by conducting the query optimization studies will look for ways to improve the operation speed of the Find and Join query in MongoDB.

Acknowledgment

This study was supported by the R&D Center for Valuable Recycling (Global-Top Environmental Technology Development Program) funded by the Ministry of Environment. (Project No.: GT-11-C-01-150-0).

References

[1] Jung, W. P., Hwa, C. Y., Myon, W. P., & Young, T. S. (2012). A study on monitoring system architecture for calculation of practical recycling rate of end of life vehicle. *Clean Technology*, *18(4)*, 373-378.

[2] Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent. *Partition-Tolerant Web Services*, *SIGACT News*, *33(2)*, 51–59, 2002.

[3] Coda, H. You can't sacrifice partition tolerance. Retrieved Oct 7, 2010, from http://codahale.com/you-cant-sacrifice-partition-tolerance

[4] Abadi, D. J. (2012). Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, *45(2)*, 37–42.

[5] Stefan, E. List of NOSQL databases. Retrieved 2009, from http://nosql-database.org

[6] Zhao, G., Huang, W., Liang, S., & Tang, Y. (2013). Modeling MongoDB with relational model. *Proceedings of the 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)* (pp. 115–121).

[7] Banker, K. (2012). *MongoDB in Action*. Manning Publications.

[8] Liu, Y., Wang, Y., & Jin, Y. (2012). Research on the improvement of MongoDB auto-sharding in cloud environment. *Proceedings of the 7th International Conference on Computer Science and Education.*

[9] Sanjoy, S. N., Manabendra, D. C., Kumar, S. P. *et al.* (2014). NoSQL data model for semi-automatic integration of ethnomedicinal plant data from multiple sources. *Phytochem Anal, 25(6)*, 495-507.

Joohyoung Jeon was born in Seoul, Korea in 1987. He received the BS degree in Department of Geography from Sangmyung University, Korea in 2010. He worked in Korea Military Service as an engineering officer from 2010 to 2012. He is currently a master's student in School of Industrial and Management Engineering at Korea University, Korea from 2014. Also he is an assistant researcher in Korea Institute of Science and Technology (KIST). His research interests are NoSQL database, distributed system, and discrete event system simulation.

Minjeong An was born in Seoul, Korea in 1977. She obtained the master degree in School of Industrial and Management Engineering at Korea University, Korea in 2008. She is currently a Ph.D candidate in School of Industrial and Management Engineering at Korea University, Korea. Her research interests are logistics information system and cloud computing architecture.

Hongchul Lee was born in Seoul, Korea in 1960. He obtained the Ph.D. in School of Industrial Engineering at Texas A&M University, USA in 1993. He is currently working as a professor in School of Industrial and Management Engineering at Korea University, Korea since 1996. His research interests are production information system and logistics information system and supply chain management (SCM)