Improved Information Retrieval Performance on SQL Database Using Data Adapter

M Husni*, S Djanali, H T Ciptaningtyas, and I G N A Wicaksana
Department of Informatics, Faculty of Information Technology, Institut Teknologi Sepuluh Nopember, Indonesia

*mhusni43@gmail.com

Abstract. The NoSQL databases, short for Not Only SQL, are increasingly being used as the number of big data applications increases. Most systems still use relational databases (RDBs), but as the number of data increases each year, the system handles big data with NoSQL databases to analyze and access data more quickly. NoSQL emerged as a result of the exponential growth of the internet and the development of web applications. The query syntax in the NoSQL database differs from the SQL database, therefore requiring code changes in the application. Data adapter allow applications to not change their SQL query syntax. Data adapters provide methods that can synchronize SQL databases with NoSQL databases. In addition, the data adapter provides an interface which is application can access to run SQL queries. Hence, this research applied data adapter system to synchronize data between MySQL database and Apache HBase using direct access query approach, where system allows application to accept query while synchronization process in progress. From the test performed using data adapter, the results obtained that the data adapter can synchronize between SQL databases, MySQL, and NoSQL database, Apache HBase. This system spends the percentage of memory resources in the range of 40% to 60%, and the percentage of processor moving from 10% to 90%. In addition, from this system also obtained the performance of database NoSQL better than SQL database.

1. Introduction
Currently the flow of information through the internet has been very heavy and for the next few years, it is predicted to continue. As a result, the stored data is also getting bigger. This triggers a new field of knowledge known as big data. Along with this, there is also database to manage big data, known as Not Only SQL (NoSQL). Big data became more famous, after the development of cloud computing services. The United Nations Economic Commission for Europe (UNECE), predicts if the increase in the amount of data will reach 350% by 2019 when compared to 2015 [1].

The technique for managing the most widely used database is the relational database (RDB) with the Structured Query Language (SQL) method, where MySQL is one of the well-known Database Manager Systems for web-based application. This system is reliable and easy to use, so MySQL has become the main choice database used by high profile companies like Facebook, Twitter, and YouTube [2].

But with increasing amount of data, big data management for data retrieval operation, using SQL method is felt less economical and less efficient, because the manager must provide server with high I/O bus specification. NoSQL database is emerging as a result of growing data transactions over the
internet exponentially using web applications. NoSQL itself has a schema-less specification and does not need to define statically how data should be modeled [3].

System which use NoSQL database, do not use the same syntax as retrieving data in SQL databases because the query syntax is different. Instead of changing the source code or changing the SQL database to NoSQL, this paper shows how to integrate two databases using data adapter.

NoSQL is designed to address the performance and scalability requirements of web applications that can’t be handled by relational database. Many open source and commercial NoSQL are developed by various vendors. There are currently more than 150 NoSQL databases [4]. Ying-Ti Liao compares data adapter systems to support hybrid database architectures, including relational databases and NoSQL databases, using three different query approaches [5].

Data consistency issues are often experienced for data synchronization in an application that uses more than one database. Husni uses model-view-controller method to synchronize distributed application [6]. Jue Wang and Dong-song Zhang present the use of middleware to create a synchronization system for distributed databases [7]. Slightly different from middleware, Ramesh Dharavath and Chiranjeev Kumar researched three-tier architecture with distributed middleware to support atomic transactions in heterogeneous NoSQL databases [8]. Yubiao Wang presents an incremental web-based incremental synchronization mechanism on a heterogeneous database [9].

In this paper, a research conducted to synchronize SQL databases with NoSQL databases using data adapters for the case of database usage expansion in applications. The application is connected with SQL to handle small and medium-sized data, the NoSQL database server as a back-end system for analyzing data and performing a number of read or write operation, or periodically performing data backups from SQL.

2. Methodology

2.1. General system design

The system built is to synchronize two SQL and NoSQL databases. MySQL is used for Database Management System (DBMS) in SQL database. Apache HBase is used for NoSQL database. The total number of servers used in this study is three servers, one server for MySQL, one server for Apache HBase and one server for Data Adapter. Figure 1 shows the General Data Adapter System Design.

![Figure 1. System Architecture](image)

Data adapter is the link between the application and the database. Data adapters play a role in receiving requests from applications and transforming data from MySQL to Apache Hbase. The main component of this data adapter system consists of four parts: the relational database, NoSQL database, DB Adapter and DB Converter. DB Adapter serves to receive requests from applications such as insert data, update data, delete data and retrieve data. The data adapter provides an interfaces that can be
accessed by the application to be able make changes and access the data in database. The interface provided by the data adapter return value in Application Programming Interface (API). This interface value result is JSON data type which made using Flask. Flask is a micro framework that runs in Python programming language. Application can accessed through port 5000. With this interface, applications can make changes and access data in the database without change the SQL query syntax.

Meanwhile, DB Converter is responsible for transforming data and reporting the transformation results recorded in the database. Data transformation is done from the RDB database to the NoSQL database. To receive requests from DB Adapter, DB Converter runs on port 5001. Apache Phoenix is used to help the transformation process. In DB Converter is also installed a database using MySQL to store the synchronization log. The main components of this data adapter are built using Python programming language.

2.2. Data adapter design
Data adapter has the most important role in the system. Any request made by the application will be processed first in the data adapter, which will then be forwarded to the database. There are several examples of interfaces used by the application, in this research the interface used is API (JSON) [10].

Data adapter consists of two main components: DB Adapter and DB Converter. DB Adapter serves to receive all requests from applications such as query submission, data retrieval, data updates, and data deletion. In order to do this, the data adapter provides an interface in API form. API generates output data in JSON data type. All data communication performed by the application is through this interface. Flask is a micro framework used to create interface features [11].

DB Converter transform data from the SQL database to the NoSQL database. In this case, the transformation is from MySQL to Apache HBase. By default, requests for changes to data from the application will be redirected to MySQL, while for data access, the requests from applications will be directed to HBase. The data transformation process is using Apache Phoenix. Apache Phoenix allows system to translate SQL queries, compiles them into a series of HBase commands, and executes them to HBase. Python [12] is used to implement all existing processes in the data adapter.

2.3. SQL database design
The SQL database is used to store all relational application data. In the design and development of an application demanded the ability of the application to be accessible by many users. Many developers separate the application server with database server on the basis of this need. In this paper, a separate server from the application server is used as the SQL database. We use MySQL for SQL database. In order to use the application, it is necessary to create user accounts that have the authority to make data changes on the database server. All requests from applications related to data changes such as insert, update, delete, and retrieval of data can be executed in MySQL.

2.4. NoSQL database design
The NoSQL Database stores data in a different format with the SQL database. This database is an additional database because of the growing amount of data. We use Apache HBase for NoSQL database. To be able to use Apache HBase, Apache Hadoop, a file system that specifically handles big data problems, must install first on the server because Apache Hbase runs on top of Apache Hadoop layer [13] [14].

The installation of the NoSQL database is performed on one separate server from the SQL application and database. HBase will accept the transformed data from MySQL that is controlled through the data adapter. This transformation process needs help from Apache Phoenix as a SQL query translator.
2.5. Transformation and synchronization process

The transformation process is done by running the application through the terminal manually. The first thing to do when synchronizing is check the synchronization log database, whether system ever do synchronize before. If it has not been done before, then system will do the initialization process. Flow diagram of the general synchronization process can be seen in Figure 2.

The initialization process is the process of copying all data in the SQL database to the NoSQL database. This process begins by dumping all data in a MySQL table into a .csv file format. Then the system will create a new table similar to the existing table in MySQL. Once completed, the next process is to transform all data to HBase. This data transformation process is handled using Apache Phoenix.

If it has been synchronized before, then the next process is the system retrieves all queries from the logs in the MySQL database. Remote MySQL database system, then read the log list in /var/log/mysql/log_query.log file. Logs that are read are based on the last synchronization time. From the log, system checks all queries which changes the data such as insert, update and delete query syntax. The query is taken and then converted to Apache Phoenix query syntax. All new syntax then stored in a single file with the .sql format. With Apache Phoenix, all queries in the .sql file will be transformed to HBase, so all the data changes that occur in the SQL database will occur also in the NoSQL database. If the synchronization process succeeds, the system will record the log and sync status to the synchronization log database. The database used for logging uses MySQL and is separate from SQL database server, NoSQL database server or application server.

2.6. Interfaces design

The application accesses the database by running SQL queries through the interface or routes provided by the data adapter. With query direct access approach, the data adapter will directly forward the query syntax to the intended database and return the value of the database output. From the route accessed by the application, the system will first check whether the query is changing data in the database or retrieving data. If the query is insert, update or delete, then the query command will be executed to the MySQL database. If not, or in this case the query is select, then the system will make sure whether the synchronization process is in progress or not. By default, the query select will be directed to the HBase database, but if the synchronization process is in progress then the query will be redirected to the MySQL database. The system then prints the output obtained from the database in the form of JSON.
3. Implementation
In this research, we use data set from openflight.org which contains data about aviation around the world [15]. This data consists of three tables, there are routes, airlines and airports. The number of rows and tables is not fully used but changed according to the scenario to be performed.

3.1. Database initialization
Initialization of the database will create a new table in the NoSQL database. If the table in HBase is already created, then the system will export data in the SQL database into .csv file format. This file then executed to the NoSQL database. This test will ensure that this process runs when the synchronization process in the SQL and NoSQL databases has never been done before, or in other words the system run the synchronization process for the first time. Figure 3 shows the print screen when the initialization process run.

3.2. Database transformation
The transformation process takes place when there are data changes in the SQL database. Changes occur from the query of applications such as insert, update and delete. If the previous system has already done the transformation, then the initialization process will not take place. The system will take a list of queries that are executed in the SQL database, then run into the NoSQL database so that the data in both databases is same. Figure 4 shows the print screen when the transformation process is in progress.

Figure 3. Initialization process.

Figure 4. Transformation process.
3.3. Data adapter interfaces
The data adapter interface provides a route that can be accessed by the application to perform a process to the database. This feature also assigns to select from database where application request will be processed. This test is expected each request from the application in the form of data changes such as insert query, update and delete, will be directed to the SQL database and for data retrieval requests will be directed to the database NoSQL.

In addition to find out whether or not the interface is running, it is necessary to record the response time required by the system until the process ends in the table.

4. Experimental Result
The response time is recorded as many as five experiments, then the average value is taken. Interface details can be seen in Table 1 (column 2) and the experimental result is shown in Table 1 (column 3).

| Interface                                      | Explanation                                      | Result  |
|------------------------------------------------|--------------------------------------------------|---------|
| /sinkron                                       | Synchronize database                              | Success |
| /insert_routes                                 | Add new line to routes table                      | Success |
| /delete_routes_by_id                          | Delete a row data in routes table                 | Success |
| /update_routes_by_id                          | Edit data in routes table                         | Success |
| /select_all_routes                             | Retrieve all data rows in routes table            | Success |
| /select_route_by_id/<id>                       | Retrieve one row data in routes table             | Success |
| /insert_airline                                | Add new line to airlines table                    | Success |
| /delete_airline_by_id                          | Delete one row data in airline table              | Success |
| /update_airline_by_id                          | Edit data in airline table                        | Success |
| /select_all_airline                            | Retrieve all data rows in airline table           | Success |
| /select_airline_by_id/<id>                     | Retrieve one row data in airline table            | Success |
| /insert_airport                                | Add new line into airport table                   | Success |
| /delete_airport_by_id                          | Delete one row data in airport table              | Success |
| /update_airport_by_id                          | Edit data in airport table                        | Success |
| /select_all_airport                            | Retrieve all data rows in airport table           | Success |
| /select_airport_by_id/<id>                     | Retrieve one row data in airport table            | Success |
| /select_routes_country/<id>                    | Retrieve one row data from route table with its country detail | Success |
| /select_all_number_routes_by_airline           | Retrieve all airline’s data and routes number     | Success |
| /select_all_number_routes_more_than/<number>   | Retrieve all flight number as per the option      | Success |
| /select_all_airline_in_routes                  | Retrieve all available airline’s data on routes table | Success |
| /select_all_airline_source                     | Retrieve all airlines data and its source flight  | Success |

4.1. Comparison number of rows in table
This test is performed to find out how many resources are used and the response time from the MySQL database to HBase during the initialization process takes place by comparing the number of lines. Transformation times can be seen on figure 5, figure 6 and figure 7. The ratio of the number of rows used in the table for each experiment is as follows:

- Airplane: 6.000, 12.000, 18.000, 24.000, 30.000
- Airport: 7.000, 14.000, 21.000, 28.000, 35.000
- Route: 60.000, 120.000, 180.000, 240.000, 300.000
Figure 5. Transformation time chart based on initialization time.

Figure 6. Transformation times chart based on number of tables.

Figure 7. Transformation time chart based on number of query.
4.2. **Comparison number of table in databases**

This test’s aim is to find out how much resources used and the response time of the data adapter system during the initialization of data from the MySQL database to Hbase. The comparison number of tables used for each experiment is 3, 6, 9, 12, and 15 tables. Transformation times can be seen on Fig. 4.

4.3. **Comparison query number insert, update, and delete**

This test’s aim is to find out how much resources and time spent by the system to execute three different query operations such as insert, update and delete from MySQL to HBase on the data transformation process. In each query operation, the comparison of query numbers used is starting at 1,000, 2,000, 3,000, 4,000 and 5,000 queries. Transformation times can be seen on Fig. 5.

4.4. **Evaluation**

From functionality testing, all functions can run as expected. However, for the data retrieval process on MySQL, there is one interface with a response time that exceeds the limit. MySQL is not good enough to handle queries that require retrieval data to more than one table in tables that have large amounts of data. On the other hand, HBase is able to handle all query processing requests from any tested interface.

In terms of capacity and performance, for all test scenarios, the percentage of RAM memory resources used showed a linear increase, no changes soaring or decreasing significantly, ranging from 1.736,6 MB to 2.431,8 MB. While the processor resources used show a decrease in percentage for each increase in the number of transformed queries. This decrease is also shown in the test results for the ratio of the number of rows and the ratio of the number of tables at the time of the initialization process. However, the difference occurs at fourth attempt. This is because Apache Phoenix only requires a high percentage of processors when the process begins or when Apache Phoenix starts executing queries. For the rest, during the synchronization process takes place, the percentage value of use tends to be lower and moves at a consistent rate. Apache Phoenix requires more RAM memory resources to perform the transformation process than the percentage of processor usage.

The average percentage of the processor used for the initialization process is 37.389%. Query transformation process spent less processor resource average than initialization process, that is 26,546% for insert, 16,208% for update and 24,943% for delete. The update query transformation process has the smallest percentage of processor usage, but in contrast to its longest transformation time between other operations. The average transformation time required for each operation is 23.4 seconds for insert, 116.7 seconds for update and 29.0 seconds for delete.

5. **Conclusion**

From the experimental results, we can conclude:

1. Data adapters can be used as a system to synchronize between SQL databases and NoSQL databases. By using direct access query approach, the application can still access the database while the synchronization process is in progress.
2. Based on the test results, for each increase in the amount of data or query, the time required to initialize is linear, both in the initialization process and the transformation.
3. NoSQL database, i.e. Apache HBase, has a better capability when compared with relational database, which is MySQL, in terms of query analysis for data retrieval of multiple tables with a large number of rows.
4. Data synchronization between SQL databases with NoSQL databases using data adapter improves information retrieval performance on SQL database.

Suggestions given for further development of this system are as follows:

1. Data adapter mechanism needs to be developed by comparing other NoSQL database type.
2. Further research is needed to synchronize the database with more number of nodes, and how the system is capable of handling failures which occur on the database server or data adapter server.
Acknowledgments
This research was supported by Lab Research Grant (Non PNBP Fund) 2017, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, Research ID 869/PKS/ITS/2017.

References
[1] G Audin 2017 No Jitter (Online) available at http://www.nojitter.com/post/240170228/the-network-impact-of-big-data
[2] Oracle 2017 Oracle (Online) available at https://www.oracle.com/mysql/index.html
[3] Couchbase 2017 Couchbase (Online) available at https://www.couchbase.com/nosql-resources/why-nosql
[4] D G Chandra 2015 BASE analysis of NoSQL database Future Generation Computer Systems 52 13-21
[5] C Liao and C Ying-Ti 2016 Data adapter for querying and transformation between SQL and NoSQL database Elsevier 65 111-121
[6] Muhammad H, Henning T C, and I G N A Wicaksana 2016 Model View Controller for Distributed Application The International Conference on Engineering, Technology, and Industrial Application (ICETIA 2016) Solo
[7] D Jue and S Z Wang 2015 Research and Design of Distributed Database Synchronization System 8th International Conference on Intelligent Computation Technology and Automation, China
[8] C Ramesh and K Davarth 2015 A Scalable Generic Transaction Model Scenario for Distributed NoSQL Databases The Journal of Systems and Software 101 43-58
[9] X Junhao, R Wen, W Yubiao, and F Wang 2015 Research on Incremental Heterogeneous database Synchronization Update Based on Web Service International Conference on Computational Intelligence and Communication Networks, India
[10] Oracle 2016 MySQL (Online) available at https://www.mysql.com/about/
[11] A Ronacher 2015 Flask (Online) available at http://flask.pocoo.org/
[12] Python Software Foundation 2017 Python (Online) available at https://www.python.org/
[13] Apache HBase Team 2017 The Apache Software Foundation (Online) available at http://hbase.apache.org/book.html#_read_hbase_shell_commands_from_a_command_file
[14] Apache Software Foundation 2017 Apache Software Foundation (Online) available at https://phoenix.apache.org/who_is_using.html
[15] OpenFlights 2017 OpenFlights (Online) available at https://openflights.org/data.html