ISSN 1975-8359(Print) / ISSN 2287-4364(Online)
The Transactions of the Korean Institute of Electrical Engineers Vol. 65, No. 3, pp. 469~476, 2016
http://dx.doi.org/10.5370/KIEE.2016.65.3.469

통신 가입자 데이터 관리를 위한 MSSQL Server와 NoSQL MongoDB의 성능 비교

A Comparison of Performance Between MSSQL Server and MongoDB for Telco Subscriber Data Management

Aaron Nichie* · 구 흥 서†
(Aaron Nichie · Heung-Seo Koo)

Abstract - Relational Database Management Systems have become de facto database model among most developers and users since the inception of Data Science. From IoT devices, sensors, social media and other sources, data is generated in structured, semi-structured and unstructured formats, in huge volumes, thereby the difficulty of data management greatly increases. Organizations that collect large amounts of data are increasingly turning to non relational databases - NoSQL databases. In this paper, through experiments with real field data, we demonstrate that MongoDB, a document-based NoSQL database, is a better alternative for building a Telco Subscriber Data Management System which hitherto is mainly built with Relational Database Management Systems. We compare the existing system in various phases of data flow with our proposed system powered by MongoDB. We show how various workloads at some phases of the existing system were either completely removed or significantly simplified on the new system. Based on experiment results, using MongoDB for managing telco subscriber data turned out to offer performance better than the existing system built with MSSQL Server.

Key Words : Subscriber data management, Single view, Big data, NoSQL, JSON

1. Introduction

Relational Database Management Systems (RDBMS) have dominated the data management industry since the inception of data science. They have become, to most data scientists, the de-facto database option for all data management projects even though other databases with different models, other than the RDBMS model, are available namely NoSQL databases. This huge popularity of RDBMS translates into developers and users having much confidence in RDBMS. Perhaps, that explains why RDBMS are the obvious choice of database systems among most companies for data management projects[1]. Most organizations are unfamiliar with NoSQL Databases and thus may not feel knowledgeable enough to choose or even determine that the approach might be better for their purpose[2].

† Corresponding Author : Dept. of Computer and Information Engineering, Cheong-Ju Univ., Korea
E-mail : hskoo@cju.ac.kr
* Dept. of Computer and Information Engineering, Cheong-Ju Univ., Korea
Received : December 28, 2015; Accepted : February 6, 2016

RDBMS are well-known for their ability to perform transactions (ACID property). However, building a Subscriber Data Management System (SDMS), also known as Subscriber Single View, does not strictly require the use of RDBMS. Yet RDBMS are hugely preferred, in most cases, among developers and users. Thus, data have to be modelled or reshaped to conform to the strict principles of RDBMS. In a sense, not using the right tool for the right job is tantamount to misapplication or misuse of resources because inefficiencies are bound to occur. In the era of IoT and Big Data such resource misapplications cannot be ignored because their impact will cause system inefficiencies if not total system failure.

NoSQL databases will not replace relational databases, but instead will become a better option for certain type of projects, such as those that are distributed, that involve large amount of data or must scale[3]. One of such projects is managing subscriber data. Subscriber data management involves managing millions of subscriber data with records of each subscriber distributed over several different telco subsystems in the form of Call Data Records (CDRs) and Internet Protocol Detail Records (IPDRs).

With increasing number of “over the top” services delivered directly to subscribers, data generated by single subscriber could, predictably, be in large amount. Management would obviously be a nightmare, if right systems and tools were not used. It is important that people learn to look at their data and select many databases for many needs[4]. Additionally, as huge number of subscribers are ported in and out daily the management system must be easily scalable to cater for this fluidity[5].

Despite the importance of SDMS, the reality on the field is that telcos struggle to achieve a single view of the subscriber. Subscriber data is managed by desperate data stores and accessed by numerous incompatible APIs[6]. With arrival of data from IoT device sources such as smart meters, Cloud, Social, Location and Sensor-based apps, the business and technical requirements significantly change. A new paradigm to how data is stored and managed alongside traditional subscriber profile in order to build a unified platform for subscriber data management is required. If telcos are to realize the goal of SDMS, they must integrate these new sources of data into their SDMS solutions[7]. The data are often loosely structured and highly variable arriving at high velocity and volume. The lack of structure does not typically lend itself to being stored in normalized relational tables instead fitting better with NoSQL models[8]. Furthermore, RDBMS were neither designed to cope with scale and agile challenges that face modern applications nor were they built to take advantage of the cheap storage and processing available today[9].

In most SDMS, transaction processes do not occur that makes NoSQL the optimal option of database model for building SDMS. Document-level atomicity which is offered by MongoDB, as we will show subsequently, is enough to build and manage a successful SDMS[10]. MongoDB, for that matter NoSQL databases, is the answer to the call for a new paradigm in building an SDMS.

In this paper, through experiments with real field data, we demonstrate that MongoDB is a better alternative for building a Telco Subscriber Data Management System which hitherto is mainly built with RDBMS. We compare the various phases of data flow on the existing system with our proposed system powered by MongoDB. We show how various workloads at some phases of the existing system were either completely removed or significantly simplified on the new system.

2. System Design and Implementation

Bringing data from several operational subsystem into a single database schema is an uphill task. Consider Fig. 1 which depicts a fraction of a typical telco ecosystem. Subscriber Data are siloed across multiple network, business and application services. Data is then managed by multiple discrete elements in the telco ecosystem.

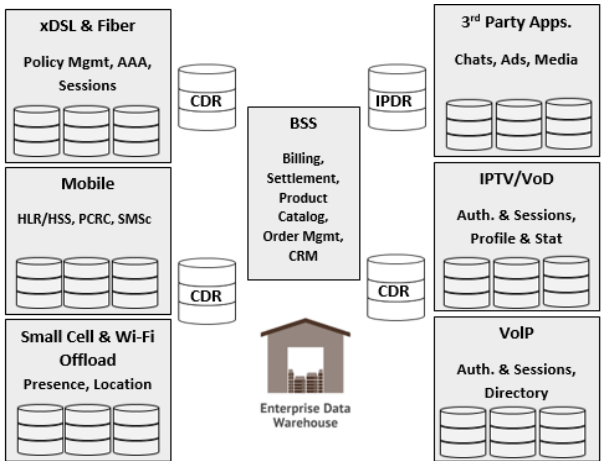


Fig. 1 Subscriber Data Siloed across Multiple Network, Business and App. Services (from MogoDB White Paper, May 2014)

2.1 Design

The subscriber data follows a simple workforce hierarchy where mobile subscribers belong to a Freelancer (FL) and Freelancers belong to a Team Leader (TL) on many to one relationship. Using RDBMS to build SDMS requires the definition of an upfront schema. In other words, it requires that the future of the data is predicted right at the start of the project. Coming out with such an upfront design, which takes into consideration all future eventualities in the life of the subscriber data, is very difficult before, during and even after the project has been deployed. Developers may keep altering schemas several times and, in worse cases, keep dropping entire schemas to build them from scratch several times over just to cope with the diversity in data formats.

The existing SDMS was implemented with MSSQL server. Fig. 2 shows schema design of the existing system in third normalized form. Fig. 3 shows MongoDB schema version.

The data models were designed based on principles which are widely accepted as rule of thumb in both database systems on the real field of operation. This serves as the foundation for our research. For MSSQL server, an RDBMS, it is a rule of thumb to design schema in the third normal form. Which is premised on the principle that RDBMS avoid being bias towards any particular data access

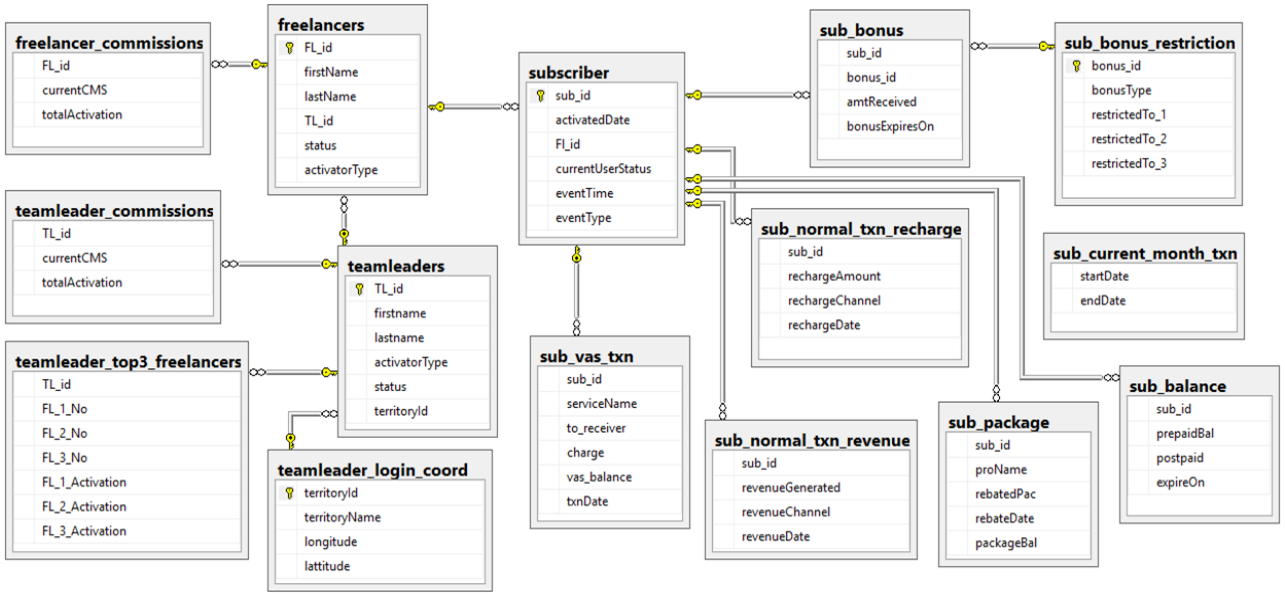


Fig. 2 MSSQL Server Schema

pattern. On the other hand, with MongoDB the rule of thumb is data is modelled according to how they are accessed. Thus, the schema was designed as embedded data model with manual inter-collection references. Document embedding, referred to as pre-join in some text, resulted in 15 MSSQL Server tables being translated into 3 MongoDB collections as shown in Fig. 2 and 3.

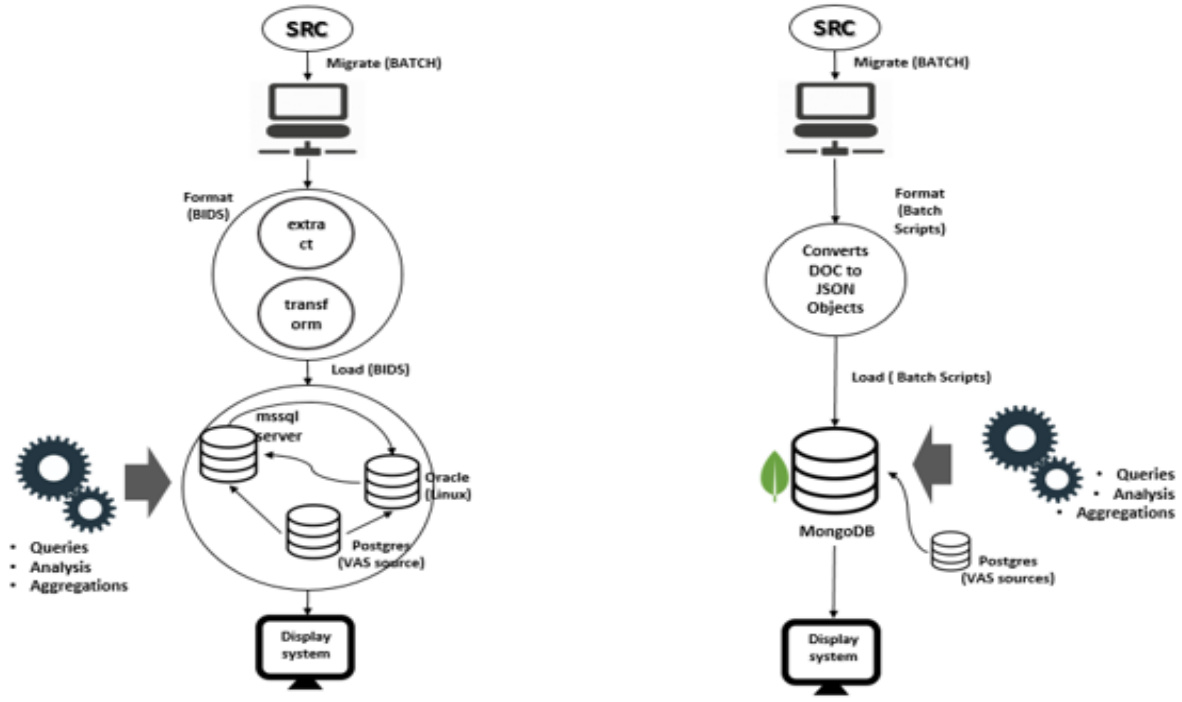
Fig. 3 MongoDB schema

2.2 Implementation

Our approach to the system implementation was motivated by the principle of plug and play. Due to the fact that other subsystems may access data from the same CDR sources, causing changes in their formats would have had cascading effects on other systems. In other words, we wanted a seamless replacement of the existing system with the new solution which is implemented with MongoDB. To this end, throughout the various stages of data flow in the existing system, the stage was maintained in the new system but needless operations at a stage were either completely removed or lumped up into a single operation as shown in Fig. 4.

On the existing system, the "format stage" was implemented using Business Intelligence Development Studio (BIDS) which is integrated into SQL Server Data Tool for Visual Studio. On the new system, we implemented that using batch scripts to extract and convert the files into JSON documents and objects appropriately. At the "storage stage", MSSQL Server and Oracle running on Linux have been replaced with MongoDB on the new system. MSSQL and Oracle were both deployed on the old system just to complement each other in the sense that applications which are easy to build on individual platforms are built and maintained on that platform, as and when needed data is exchanged among the two via Linked Server. However, MSSQL Server is the main database serving all requests on the existing system. Thus, in this project Oracle was not involved in the implementation.

It can be noticed that an RDBMS, Postgres, which manages Valued Added Services (VAS) was allowed in the new system. The reason is that VAS operations are transactional processes which RDBMS are extremely good at managing. However, we could have incorporated the VAS processes into MongoDB by utilizing atomicity at document-level together with implementation of locks to achieve transactions. As best practice, RDBMS and NoSQL databases are co-deployed to process different data flow to achieve the optimal combined features from both families[10]. Additionally, most professionals are of the view that, database administrators today must be polyglot-



(a) Existing SDM System (b) New Proposed SDM System With MongoDB

Fig. 4 Subscriber Data Management System Data Flow

friendly, meaning they must know how to work with many different RDBMS and NoSQL databases.

3. Test Criteria

Table 1 presents the technical environment on which the test was conducted. The two systems were tested in the fashion presented in Table 2. Apart from BULK INSERT and ETL categories, the other category of queries were executed twice for each SDMS solution. The first being a case where no extra indexes existed apart from the primary key index and the second being a case where all necessary secondary indexes existed.

Table 1 Test Environment

Computer System	
Operating System	Windows 10 Home 64-bits (10.0, Build 10586)
Processor	Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz (4 CPUs), ~3.3GHz
Memory	8192MB RAM
Drive Type	Solid State Drive (SSD)
Database System and Add-Ons	
Microsoft SQL Server Express 2014 (64-bit) Version: 12.0.2269.0	MongoDB for Windows 64-bit Version 3.0.7 Default Engine: MMAPv1
Visual Studio 2013 Business Intelligence - SSIS	

The test data, which is an actual field data, ranges from one million records to 10 million records of telecom subscriber data. The ETL portion of the test was to measure the performance of our new solution proposed in Fig. 4. We increased the test data and ran the query sets to measure execution time. The results are summarized into charts which are presented in the Results and Analysis chapter.

4. Results and Analysis

In this section, experiment results are explained. The results are compared with known theoretical facts to confirm the validity or otherwise of the results.

4.1 Bulk Insert

The first segment of this test case was that which

Table 2 Explanation of test criteria

Test criteria	Description
BULK INSERT	
• Denormalized • 3rd Norm & Embed Doc.	Single table/collection insert Normalized tables/embedded JSON docs
EXTRACT TRANSFORM LOAD	
• Data extracted from CALL, SMS, Data Package & Voucher .UNL CDR Files	
Simple Queries	Day to day normal operation queries
• SELECT • UPDATE • DELETE	Select / find range query Update / Upsert tables / documents when subscribers are ported in / activated Remove subscribers when ported out
Aggregation Queries	Queries made out of business policies
• FL_tot_activations • TLs_Top3_FLs • Realign_FLs_to_TLs • Check_TLs_No_FLs	Total activations made by freelancers for current business period Team leader's top 3 freelancers with most number of activations Realignment of freelancers to team leaders Total number and details of freelancers per team leader

involved denormalized data. On MSSQL Server a single table for subscribers with single record for each subscriber. A single collection for subscribers with single unembedded document for each subscriber on MongoDB. The result is shown in Fig. 5.

The second segment of this test case was that which involved normalized data on MSSQL Sever and embedded documents on MongoDB. It is important to stress that this is the actual implementation in the real field of operation supported by the various rule of thumbs. Thus, subsequent test cases followed the rule of thumbs aforementioned in the design section of this paper. The result is shown in Fig. 6.

Fig. 5 shows a linear growth in elapsed time as the amount of data increased in both databases. However, MSSQL Server was faster in bulk inserting the denormalized data while MongoDB took almost double the elapsed time of

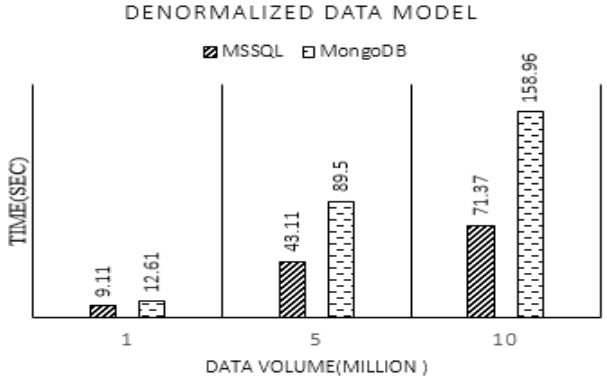


Fig. 5 Test Result for Denormalized Data Model

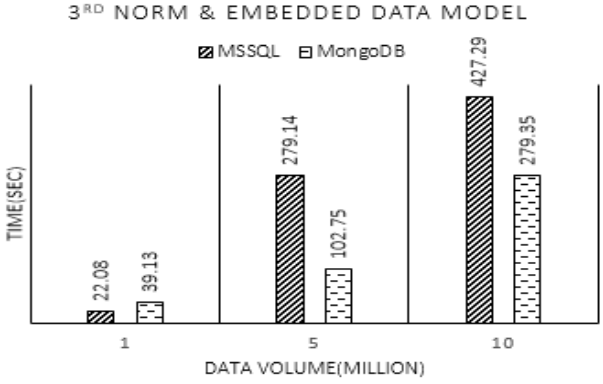


Fig. 6 Test Result for Normalized & Embedded Data Model

MSSQL Server to bulk insert the same amount of data. Taking a closer look at Fig. 6, at the beginning, MSSQL Server took less time to bulk insert one million records. However, that was not the case as data volume increased up to ten million. It took much longer time than MongoDB. This result shows that it was faster for MongoDB to bulk insert embedded documents than it was for MSSQL Server to bulk insert normalized data.

The theoretical explanation is that MSSQL Server had to repeat certain data, example subscriber ID, in the foreign keyed tables to comply with referential integrity constraint.

4.2 Extract Transform Load - ETL

This test case was made up of five different ETL processes. Each process extracting data from .unl source files which are generated by various telco subsystems. Fig.

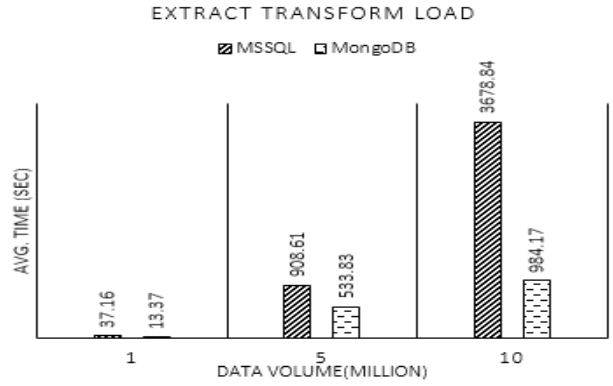


Fig. 7 Test Result for ETL Processes

7 presents the average elapsed time of the five processes.

From the results, ETL process on MongoDB using our proposed approach with batch scripts, as explained in details in the implementation section, was a faster approach. As the data got big, the ETL process on MSSQL Server, implemented in BIDS, took an exponential amount of time. The performance of our new solution suggests that collecting data in operational systems and then relying on nightly ETL processes to update the subscriber records, which is the case in the existing system, is no longer necessary but just an option. Our new approach can offer record updates in reasonable amount of time thereby eliminating the need to rely on nightly updates.

4.3 Simple Queries

Simple query test case was made of two segments. The first segment was when the tables and collections, respectively, did not have any secondary indexes. That is, the only index available was primary key index. Fig. 8

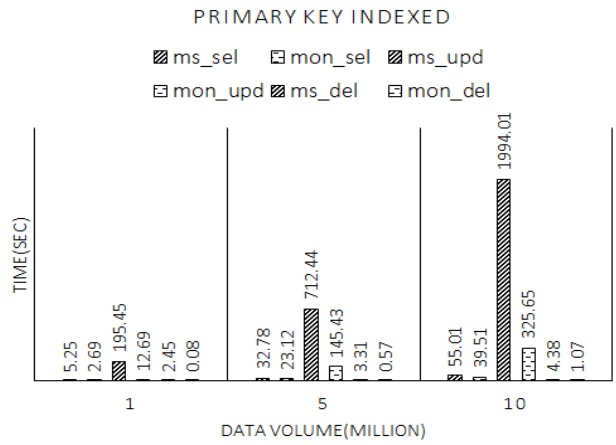


Fig. 8 Test Result for Simple Queries – Primary Key Indexed

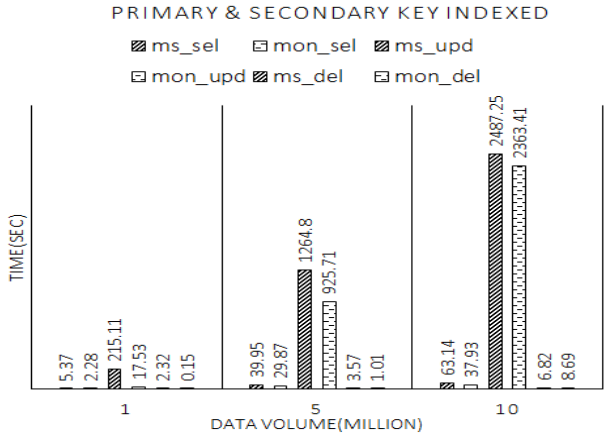


Fig. 9 Test Result for Simple Queries – Primary Key & Secondary Key Indexed

presents the results. The second segment was when the tables and collections, respectively, had secondary indexes. Fig. 9 presents the results. From the result, ms and mon mean MSSQL Server and MongoDB respectively. Also, sel, upd, and del mean select, update and delete respectively. For instance, ms_sel implies SELECT query on MSSQL Server and mon_upd means UPDATE query on MongoDB.

In this test MongoDB was faster. Particularly consider the UPDATE query. MSSQL took exponential amount of time to perform the updates. The reason is that to update a single record, MSSQL Server had to check for referential integrity before performing update with possibility of cascading the updates across multiple tables. MongoDB did not suffer from those bottlenecks. It was faster for MongoDB to navigate to the innermost embedded documents than it was for MSSQL Server to perform updates on multiple tables. From Fig. 9, MongoDB was faster in all queries but not as we expected in performing updates. Particularly, considering the update query, the performance of MongoDB degraded drastically from the case when there were no secondary indexes. MongoDB almost took the same amount of time to perform the update as it took MSSQL Server. Theoretically, prior to an update if indexes are available on affected table columns or document fields respectively, index rebuilding occurs as and when updates are performed. The cost of which is pretty expensive in MongoDB than MSSQL Server as our experiment showed.

4.4 Aggregation Queries

This test case also had two segments. First with only Primary Key index and then with secondary indexes. From

the result, ms and mon mean MSSQL Server and MongoDB respectively. Also, FTA, RA, CA and top3 mean FL_tot_activations, Realign_FLs_to_TLs, Check_ TLs_No_FLs and TLs_Top3_FLs queries respectively. For instance, ms_FTA implies FL_tot_ activations aggregation query on MSSQL Server and mon_Top3 means TLs_Top3_FLs aggregation query on MongoDB.

In all aggregation queries in both segments MSSQL Server was faster as shown in Fig. 10 and 11. Expectedly, MSSQL Server, as an RDBMS, is built to perform complex aggregations by joining multiple tables efficiently. However, with MongoDB the concept of join is not supported but possible to implement using manual links and DBRefs.

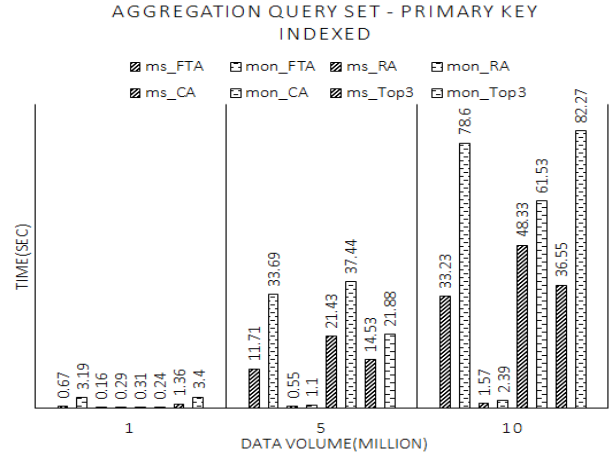


Fig. 10 Test Result for Aggregation Queries – Primary Key Indexed

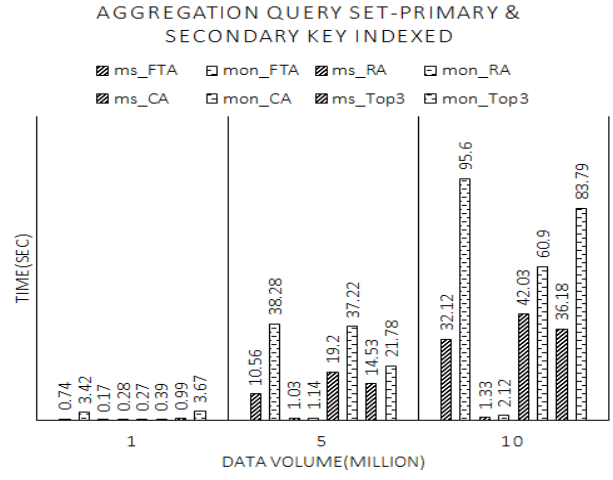


Fig. 11 Test Result for Aggregation Queries – Primary & Secondary Key Indexed

5. Conclusion

Based on our results we can make conclusion that our new ETL solution, which has MongoDB as its engine, can reliably replace the old system to achieve better overall system performance. Our new ETL approach can offer record updates in reasonable amount of time thereby eliminating the need to rely on nightly updates. Also as other test results have shown, SDMS with MongoDB yields superior performance over MSSQL Server. Its showed better performance in the simple query test case, bulk insert (3rd Norm. & embedded) test cases. MSSQL Server showed better performance with bulk insert (denormalized) and in the aggregation query test.

In reality, aggregation queries are less frequently executed, usually once in a fortnight. Thus, their faster executions may not be advantageous in broad sense. Simple queries, on the other hand, are executed several times in a day to serve business purposes both by end users and developers. Thus, simple queries have high tendency of impacting the system performance rather than aggregation queries.

References

[1] S. Nyati, S. Pawar, R. Ingle, “Performance Evaluation of Unstructured NoSQL Data Over Distributed Framework”, Advances in Computing, Communications and Informatics (ICACCI), pp. 1623-1627, Oct. 21, 2013.

[2] L. Neal, “Will NoSQL Databases Live Up to Their Promise?”, Technology News, IEEE Computer Society, pp. 12-14, Oct. 2010.

[3] A MongoDB White Paper: RDBMS to MongoDB Migration Guide, Considerations and Best Practices, pp. 2-14, Aug. 2015.

[4] W. Ming-Shih Dr. “Current Data Security Issues of NoSQL Databases”, Fidelis Cybersecurity, pp. 2-14, Jan. 2014.

[5] P. Enfedaque, From Oracle to MongoDB: A Real Use Case at Telefonica PDI, Oct. 06, 2012. [Online]. Available: www.slideshare.net/pablito56/nosql-lmatters fromoracletomongodbv3/1(Downloaded: Nov. 09, 2015)

[6] Telecommunications Datasheet, [Online]. Available: http://s3.amazonaws.com/info-mongodb-com/MongoDB_Telco_Datasheet.pdf (Downloaded: Oct. 04, 2015)

[7] Subscriber Data Management: Unlocking New Customer Insight with Big Data, A MongoDB White Paper, pp. 2-9, May 2014.

[8] A. Hafiz, O. Lukumon, B. Muhammad, A. Olugbenga, O. Hakeem, A. Saheed, "Bankruptcy Prediction of Construction Businesses: Towards a Big Data Analytics Approach", IEEE Conf. Pub., pp.1-5, Mar. 09, 2015.
[9] Huawei Technologies Company Ltd: CBS Call Detail Record Reference V100R002C02LG0372_09 Millicom Ghana LTD - Tigo, pp. 17-206, May 31, 2014.
[10] L. Tuan Dinh, S. H. Kim, N. Minh Hoang, D. Kim, S. Y. Shin, L. Kyung Eun, R. Da Rosa, "EPC Information Services With NoSQL Datastore for Internet of Things", IEEE Conf. Pub., pp. 47-54, May 08, 2014.

저 자 소 개

Aaron Nichie graduated from University of Ghana, Legon with B.S. degree in Computer Engineering in 2012. He is currently pursuing his M.S. degree at Cheong-Ju University, Korea. His research interests include Internet of Things (IoT), Data Mining, Big Data, Data Analytics and Data Engineering.
E-mail : nichieaaron@gmail.com

Heung-Seo Koo received his B.S., M.S., and Ph.D degrees from Inha University, Incheon, Korea, in 1985, 1989, and 1993, respectively. Since 1994, he has been with Cheong-Ju University, Korea, where he is currently a professor. His research interests include Big Data, DITA.
Tel : 043-229-8492
E-mail : hskoo@cju.ac.kr