Uniformly Integrated Database Approach for Heterogenous Databases

Hlaing Phyu Phyu Mon, Thin Thin San, Zinmar Naing, Thandar Swe
University of Computer Studies (Meiktila), Meiktila, Myanmar
hlaingphyuphyumon16@gmail.com

Abstract

The demands of more storage, scalability, commodity of heterogenous data for storing, analyzing and retrieving data are rapidly increasing in today data-centric area such as cloud computing, big data analytics, etc. These demands cannot be solely handled by relational database system (RDBMS) due to its strict relational model for scalability and adaptability. Therefore, NoSQL (Not only SQL) database called non-relational database is recently introduced to extend RDBMS, and now it is widely used in some software developments. As a result, it becomes challenges regarding how to transform relational to non-relational database or how to integrate them to achieve business purposes regarding storage and adaptability. This paper therefore proposes an approach for uniformly integrated database to integrate data separately extracted from individual database schema from relational and NoSQL database systems. We firstly try to map the data elements in terms of their semantic meaning and structures with the help of ontological semantic mapping and metamodeling from the extracted data. We then cover structural, semantical and syntactical diversity of each database schema and produce integrated database results. To prove efficiency and usefulness of our proposed system, we test our developed system with popular datasets in BSON and traditional sql format using MongoDB and MySQL database. According to the results compared with other proficient contemporary approaches, we have achieved significant results in mapping similarity results although running time and retrieval time are competitive with the others.

Keywords- Relational database, NoSQL database, ontological semantic mapping, database schema

1. Introduction

In today's IT software development, every developing process needs to use database for storage, analyzing and retrieval of various kinds of data depending on their goals. As IT technologies advances with evolution of cloud computing, big data, etc, it needs to store tremendous amount of data and information in different kinds of development platforms. Consequently, the role of relational database for management and storage purpose becomes insufficient due to lack of large capacity, scalability and heterogenous capability to work with advanced database products and needs. Therefore, new innovation called NoSQL database system evolves so that the needs of current technology demands can be supplied. Meanwhile, the usage of relational DBMS cannot be discarded because many software products are still using RDMBS due to its rich features and usefulness. Therefore, there is a need to build a bridge for those two types of databases so that they can be integrated for simultaneously logical needs of data from physically distributed databases over heterogenous data sources [1].

In integrating the data from separated relational systems into a new one, there exit a lot of solutions [1]. However, to the best of our knowledge, there is only few researches [1,2,3] to integrate distributed relational and nonrelational database. There are many challenges to combine different complex database structures and schemas so as to migrate all required information of each into a new database one.

While relational database management system depends on relational data model such as MySQL, Oracle, PostgreSQL, etc, non-relational NoSQL management systems are using various kinds of semi-structured data model such as key-value stores, column stores, graph stores, etc [4,5,6]. Therefore, many scholarly works are being demanded to address the issues of structural mapping upon different syntactic and semantic structure, understanding the semantic meaning of database elements and relationships. Our paper therefore takes these challenges as research opportunities to figure out how data elements, relations and structures are semantically mapped with the use of ontological semantic definitions and how to transform them to well organized new database.

The contribution of this paper is introducing how to semantically map database definitions among different database elements, relations and structures without consideration of schema mapping, database aggregation and joins among different data sources. For our purpose, we particularly use MySQL (sql[1] file extension) for relational database and MongoDB (bson[2] file types) for

[1] sql-structured query language
[2] bson-binary json

ISBN 978-99971-0-381-9 © 2017 ICAIT

non-relational database. MongoDB which acts like database as service over cloud network using rich storage structures and query languages [7].

The remainder of the paper is organized as follows. A brief note for background theory of NoSQL and relational database, and data integration problems and solutions are described in Section 2. In Section 3, we explain the structure and solution of proposed system and the analysis are described in section 4. We finally conclude the paper in Section 5 by exploring our intended future works.

2. Preliminary study

2.1. Background theory

2.1.1. NoSQL Database vs Relational DBMS. The term NoSQL was first introduced in 1998 for relational database to skip the use of SQL [10]. The term was used again in 2009 at the conferences of advocates of non-relational databases NoSQL meetup in San Francisco [11]. It is designed for rapidly iterated changing environment especially in agile software development process so that a significantly higher data throughput is produced, horizontal scalability is supported for huge volume of data storage and commodity hardware for more cost-effective alternatives.

Relational RDBMS database systems were developed in 70's to store structured data in the form of table with their own query language model called structured query language (SQL) [12].

In contrast to RDBMS, NoSQL uses structural, semi-structure and unstructured documents to store the data, and enables to scale the storage volume well in the horizontal direction for very large amount of data which are desperately demanded in cloud computing and big data storage. Moreover, the design of NoSQL does not rely on highly available hardware, and it challenges the shortcomings of RDMBS such as rigid schema design, performance of single servers and limited storage data (eg. 50 GB for inbox search at Facebook or 2PB in total at eBay).

2.1.2. Data Integration Problems and Solutions. The advent of NoSQL gains a great attention of research scholars and have been evolving many achievements and proposals to enhance NoSQL techniques. Among them, data integration from different databases involves with specific problems and solutions.

A logical integration of data separately stored in different databases reduces time-consuming, cost and human made errors for the processes which are using manual integration. Furthermore, a semantic based logical integration can handle complex structures and meanings of data elements which are going to combine as new one. Although there are many popular database drivers such as JDBC, OLE DB, etc which use generalized query languages. They are also able to handle different database management systems but they lack of capabilities to work on structural, semi-structural and semantic differences of data sources [1]. Therefore, we need to develop a systematic integrated approach that can understand semantical and syntactical meanings of data elements, relationships and structures of different data sources so as to integrate different structures and schema types of relational and non-relational databases.

2.2. Literature review

The popularity of NoSQL becomes heated since very recent years. As it is, many scholarly works studies and proposes some advanced features and methods to interoperate NoSQL. However, only a few studies empathize on integration of data stored in NoSQL systems [13,14,15]. The research works [1, 16] propose uniform interface and platform to integrate databases. Whereas the work [1] presents uniform access platform to collect data from different separated database management system, the work [16] proposes a uniform interface that allows to access the data stored in different NoSQL systems (HBaase, Redis, and MongoDB). The paper work [17] presents a framework to seamlessly fill the gap of SQL deficits with the help of document stores structure of NoSQL.

As explained above, data integration among different databases plays a key role in migration process. Therefore, in our paper, we propose an approach to integrate data from different sources without a need of concept of both relational and non-relational databases for the user and programming skill. They just need to load the databases they want and our system will map the required process and deliver the merged database in non-relational format (JSON[3]) to the users.

3. Problem architecture and solutions

3.1. Problem architecture

The architecture of our proposed approach integrates the idea of HybridDB [1] and our novel idea in integration of heterogenous databases. The workflow of our architecture initiates when a user request is received. The user request will be importing the databases they want to merge (Mysql and MonoDB files in our paper).

The user inputted databases files are accepted by database controller and query the database views, table views and dataset results with the aid of particular native driver of each different database: MySQL and

[3] json-javascript object notation

MongoDB. The resulted query results are relayed to database modular that extracts particular connection and specification parameters for data records contained in each database. In this case, each database file may contain more than one table. The user is allowed to use any number of table for each database. We regard that those databases are already normalized. After database modular separates each table of each database definitions, the database manipulator organizes them into similar semantic concepts and maps each element (name, value types, relationships, structures, etc) of different data sources with the help of DB ontology. The sample scenario can be seen in Figure 1.

The core part of this architecture is database controller that accepts inputs, executes database operations on the source system with the aid of database manager which can access and control native drivers of all database types allowed by this system.

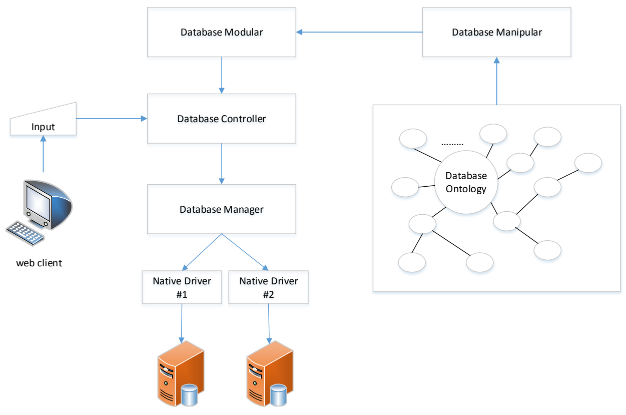

Figure 1. Architecture of proposed system

3.2. Problem Solution

3.2.1 Managing database operations: This process deals with inputted database files depending on their database types. The scenario we consider in this paper is staff information list of a university.

```
    db.createCollection(“staff_profile”);
    db.staff_profile.insert([{staff_ID:1,
         name:"Thein Tun",
         position: "Lecturer",
         address:{
                 street:"PyiTawThar",
                 city:"Yangon"
                },
         contact:[
         {name:"U Myo Myint",relationship:"Father"},
         {name:"Daw Sein",relationship:"Mother"}
         ]
       }, {…}, {….}]);
```

Figure 2(a). CRUD database operation of MongoDB

The user may enter the staff information in two different files: .sql and .json file for MySQL and MongoDB integration. Our approach then uses different CRUD (create, insert, update and delete) operations for each particular database shown in Figure 2 (a) and (b) without human participation.

In the first place, after getting user inputs, the system will try to query data, value, data types, relationship and structure separately for each database type. The database controller and manger work together to get connection to native drivers and get all possible information on those inputted database script files.

```
create database staffs;
create table staff_profile {
   ID int (PK), staff_name varchar(30), rank varchar
   (30), salary varchar(10), phone_number
   varchar(15), town varchar(10)
};
create table staff_contact{
        ID int (FK),
        name varchar(20),
        relationship varchar(20)
};
insert staff_profile (1,’Thein Tun’,’Lecturer’,’200,000
MMK’,’00959-******’,’Yangon’);
```

Figure 2(b). CRUD database operation of MySQL

3.2.2 Structural, semantical and syntactical mapping: The database manipulator understands the heterogeneity of data structure, relationships and semantic meaning of data objects of both database files with the help of database ontological structure. Here, we assume that there will be some relationships between two databases. The ontology extracts a real connection between data objects of both files and translates them, removes duplicate records, attributes and sometimes transforms some data into another types and structures. For those cases, we build ontology based on database terminologies and possible relationships of the dataset. In this paper, we train our ontology structure with 35 instances of datasets and test their usefulness with 800 datasets in evaluation stage. We use Protégé for structuring ontology and use OWL-API java platform for querying ontological meanings upon Tomcat web server.

3.2.3 Organizing integrated database file: After understanding and manipulating the inputted two database files, the database controller merges them into new database one in NoSQL format, .json file format in this paper. The result file is then tested by opening the connection its native driver and perform essential CURD operations before delivering to the users so that encountered errors can be solved in this stage. The final result for example scenario is shown in Figure 3.

```
db.staff_profile.insert([{staff_ID:1,
        name:"Thein Tun",
        position: "Lecturer",
        salary:"200,000MMK",
        address:{
                     street:"PyiTawThar",
                     city:"Yangon",
                     phonenumber::0095-9-***-***"
                    },
        contact:[
        {name:"U Myo Myint",relationship:"Father"},
        {name:"Daw Sein",relationship:"Mother"}
        ]
    }, {…}, {….}]);
```

Figure 3. Integrated database result

4. Experimental Results

4.1 Implementation Setting

The proposed system is developed with laravel 5.3 MVC framework and angualrjs for front and back-end interfaces. Tomcat server is used for web server, and OWL-API is used to build semantic information of database elements and structures. For two different databases types, as mentioned earlier, sql file for MySQL and BSON (binary JSON) file for MongoDB are used. The datasets are download from the database [18,19] and tested with 35 instances of database with different 800 datasets. The result file is produced as json format to be compatible to run on any NoSQL database.

The system is implemented on a window 10 PC equipped with 3.10 GHz, Intel® Core TM of CPU and 4.0 GB of RAM.

4.2 Experimental Results

The system performance is evaluated with three main parameters: similarity rate, retrieval time and throughput time. These criteria are measured by varying database sizes and number of different datasets as illustrated below. To prove competitive results, we compare our evaluation results with other proficient works called HybridDB[1] and SOS platform [13].

a) impact of dataset size

We measure the retrieval and throughput time by varying the sizes of databases. As shown in Figure 4(a), both of retrieval times in MySQL and MongoDB become significantly low in all compared approaches when the database size increases as general theory. The retrieval and total throughput time for particular database size: small dataset (below 80 rows and below 10 columns), medium dataset (between 80 and 5000 rows, and between 10 columns and 30 columns) and large datasets (between 5000 and 10,000 rows and between 30 and 50 columns).

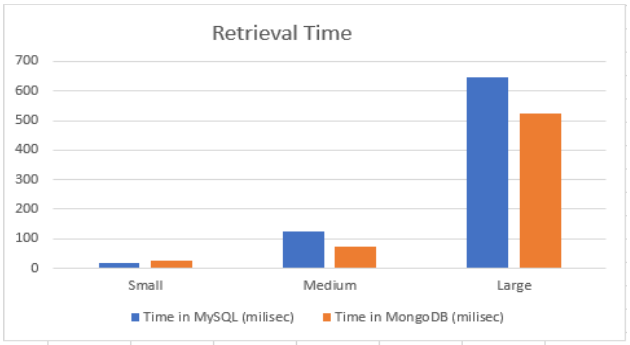

Figure 4(a). Measurement of retrieval time

The retrieval time starts when user request is sent from our system to native data source until getting the query results from them. For smaller database size, MySQL can work faster than MongoDB. For relatively increasing database size, MongoDB gets significant results in its speedy database operation.

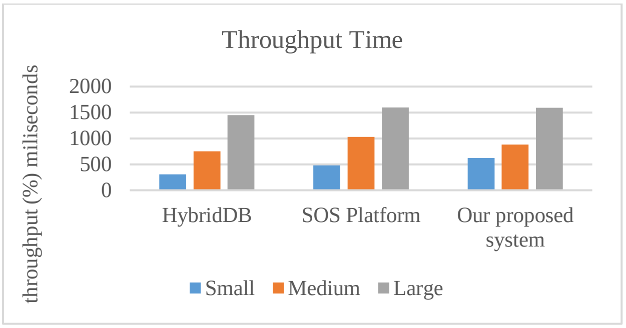

Figure 4(b). Measurement of throughput time

The throughput time means the total time since user inputs the files and until they receive the results. We got relatively similar result in these two retrieval and throughput time compared with HybridDB and SOS platform due to their significantly competitive methods.

b) impact of different dataset numbers

The similarity rate is measured how database ontology matches the elements, relationships and structures of two inputted database files. The higher value of similarity rate means exact similarity and the lower value describes higher dissimilarity. To test the similarity rate to show the efficiency and usefulness of ontological usage, we investigate our proposed system with compared works by testing different 800 datasets which are significantly different of major 35 instances of database files.

According to results described in Figure 4(c), our similarity rate is significantly higher in density of database sizes because the more classes we consider in mapping, the higher the similarity rate we can find due to semantic technology. For smaller data sets, the result is not much different with popular approach HybridDB in this field while our result is better than SOS platform.

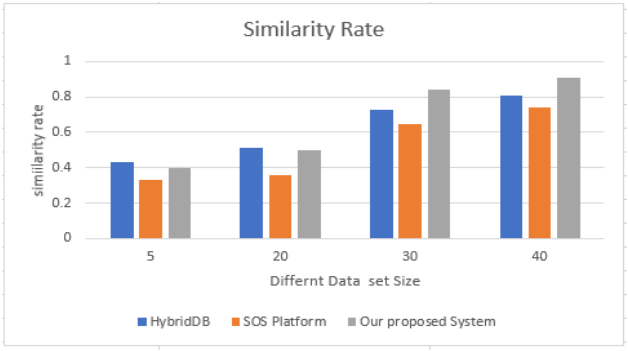

Figure 4(c). Measurement of similarity rate

5. Conclusion and future work

This paper has tried to fill the gaps of database integration problem by innovating uniformly integrated approach for different databases. We contributed an ontological mapping to understand semantic structure of data elements of relational and non-relational data sources. As the limitation of this paper, our system will be able to integrate two databases which are not zero relationships between them. The experimental proved that we have better results in similarity rate which is mostly needed in database integration area in order to minimize the complex and subtle meaning of data schema. We will extend this proposed work for further database operations such as join, aggregation, etc that are current limitations of this paper.

6. References

[1] A. V. Fogarassy, T. Hugyak, “Uniform data access platform for SQL and NoSQL database systems”, Information Systems, 69 (2017),pp.93-105.

[2] W. Allen, “Unified data modeling for relational and nosql databases”, 2016. https://www.infoq.com/articles/unified-data-modeling-for-relational-and-nosql-databases

[3] R. Sellami, S. Bhiri, B. Defude, “Odbapi: a unified rest api for relational and nosql data stores”, in: 2014 IEEE International Congress on Big Data, IEEE, 2014, pp. 653-660.

[4] V. Abramova, J. Bernardino, P. Furtado, “Experimental Evaluation of NoSQL Databases”, International Journal of Database Management Systems (IJDMS), Vol. 6, No.3, June 2014.

[5] B.G. Tudorica, C. Bucur, “A comparison between several nosql databases with comments and notes”, in: 2011 RoEduNet International Conference 10th Edition: Networking in Education and Research, 2011, pp. 1–5

[6] L. Dobos, B. Pinczel, A. Kiss, G. Racz, T. Eiler, “A comparative evaluation of nosql database systems”, Anales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae Sectio Computatorica 42 (2014), pp.173-198.

[7] https://www.mongodb.com/

[8] D. Kumawat, A. Pavate, Correlation of NOSQL & SQL Database, Journal of Computer Engineering (IOSR-JCE), volume 18, issue 5, 2016, pp. 70-74.

[9] A. B. M., Moniruzzaman and S A Hossain. "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison." arXiv preprint arXiv:1307.0191 (2013).

[10] Strozzi, Carlo: No-SQL-A relational database management system. 2007-2010. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[11] Evans, Eric: NoSQL 2009, May 2009. Blog post of 2009-05-12.

http://blog.sym-link.com/2009/05/12/nosql_2009.html

[12] Fayech, I. and Ounalli, H.: “Towards a Flexible Database Interrogation”, International Journal of Database Management Systems (IJDMS) Vol.4, No.3, June 2012 .

[13] P. Atzeni, F. Bugiotti, L. Rossi, “Uniform access to non-relational database systems: The sos platform” in: Advanced Information Systems Engineering Springer, 2012, pp. 160-174.

[14] O. Cure, F. Kerdjoudj, D. Faye, C. Le Duc, M. Lamolle, “On the potential integration of an ontology-based data access approach in nosql stores”, Int. J. Distrib. Syst. Technol. (IJDST) 4(3) (2013) 17-30.

[15] Y. Yuan, Y. Wu, X. Feng J. Li, G. Yang, W. Zheng, “Vdb-mr: mapreduce-based distributed data integration using virtual database”, Future Gener. Comput. Syt. 26 (8) (2010) 1418-1425.

[16] P. Atzeni, F. Bugiotti, L. Rossi, “Uniform access to nosql system”, Inf. Syst. 43 (201) 117-133.

[17] J.Roijackers, G. H. Fletcher, “On bridging relational and document-centirc data stores”, in: Big Data, Springer, 2013, pp. 135-148.

[18] http://jsonstudio.com/resources/

[19]http://www.mysqltutorial.org/mysql-sample-database.aspx