A Study of Normalization and Embedding in MongoDB

Mrs. Anuradha Kanade
Sinhgad Institute of Management & Computer Application Narhe, Pune
anuradha3279@gmail.com

Dr. Arpita Gopal
Sinhgad Institute of Business Administration & Research
Kondhwa, Pune
arpita.gopal@gmail.com

Mr. Shantanu Kanade
Sinhgad Institute of Management, Vadgaon, Pune
shantanukanade@gmail.com

Abstract— **With the advancement in the database technology NoSQL databases are becoming more and more popular now a days. The cloud-based NoSQL database MongoDB is one of them. As one can know the data modeling strategies for relational and non-relational databases differ drastically. Modeling of data in MongoDB database depends on the data and characteristics of MongoDB. Due to variation in data models MongoDB based application performance gets affected. In the present paper we have applied two different modeling styles as embedding of documents and normalization on collections. With the embedding feature we may face situation where documents grow in size after creation which may degrade the performance of database. The maximum document size allowed in MongoDB is limited. With references we get maximum flexibility than embedding but client-side applications must issue follow-up queries to resolve the references. The joins in this case cannot be effectively used. Hence there is need for defining the strategy of extent of normalization and embedding to get better performance in the mixed situation. The paper discussed here shows the variation in the performance along with the change in the modeling style with reference to normalization and embedding and it gives the base to find the extent of normalization and embedding for reducing query execution time.**

Keywords **—NoSQL, RDBMS, MongoDB, Zettabytes, Normalization, Denormalization, Embedding, Query Processing Time**

I. Introduction

Last three decades were ruled by the traditional relational database management systems like DB2, Oracle etc. They have the standard SQL [8]. Due to the growing web scale applications like Facebook, Mobile apps, RFID the internet has become an essential part of the world today. Everyday zettabytes of data is being generated due to these applications. Due to changing need of applications and databases the traditional relational databases are proved to be weak in distributed environment. This made NoSQL databases to get importance and preference. Being schema free, elastic and scalable, NoSQL databases appeared to be effective [2].

The term NoSQL is used first time in 1998 by Mr. Carlo Strozzi to name his lightweight open source relational database. The system did not expose the standard SQL interface [13].
There is a series of database following NoSQL standards. The term “Not Only SQL” is also used for these databases. They provide storage and retrieval mechanism with less constrained consistency models than traditional relational databases. Because of simple design, horizontal scalability and availability NoSQL databases have gained the popularity worldwide. Recently they are widely used in big data and real time web applications like Facebook, Yahoo, Google, and Amazon. With the advent of the Web 2.0 NoSQL has entered the database market [2].

Mainly there are five categories of NoSQL databases like Key-Value Store, Document-Store, Column-Oriented, Graph Database, and data structure store. Cassandra, MongoDB, BigTable, neo4j and Redis are the examples of these databases respectively. The databases like XML and Object Oriented Databases come under the NoSQL category. With the venture funding and open-source movement provided by the smaller IT companies databases like Couchbase, MongoDB and Riak have come up. Oracle also has enhanced its old Berkeley DB with some NoSQL features which is called as Oracle NoSQL. A product HBase by IBM has been targeted by the big data now [2][13].

In this paper the study on NoSQL, open source, schema free, document store database-MongoDB is presented. The MongoDB supports BSON documents and it is highly scalable. The features like map-reduce, auto-sharding and rich documents are the main attractions of MongoDB for the NoSQL users. MongoDB is a great match for cloud-based, on-premise, hybrid deployments. Simple structure of MongoDB helps to understand it easily.
In following section the work carried out on MongoDB from other researchers is summarized.

II. Background Work on MongoDB

As NoSQL trend is relatively new, there are various opportunities for research and development. Many researchers are attracted to this category of databases. Apart from other NoSQL databases we discuss here the work done in MongoDB as follows.
The main functionality and security features of two of the most popular NoSQL databases: Cassandra and MongoDB are reviewed in this paper [9]. The main problems they found common to both systems include lack of encryption support for the data files, weak authentication both between the client and the servers and between server members, very simple authorization without support for Role Based Access Control (RBAC) or fine-grained authorization, and vulnerability to SQL injection and Denial of Service attacks. This gives the opportunity to implement the security aspects in MongoDB like NoSQL databases. The basic map-reduce algorithm is studied in [8]. It is claimed that the NoSQL databases such as MongoDB and its key-value stores provide an efficient framework to aggregate large volumes of data. The comparison between Oracle and MongoDB is done by considering various issues such as theoretical, differences, features, restrictions, integrity, distribution, system requirements, architecture, query and insertion times [1]. They stated that MongoDB provides flexibility, horizontal scalability. Also it can store complex data like array, object or reference into one field. Mapping of objects is also very easy in MongoDB. The features of MongoDB like map-reduce and replications of data make the development faster than the Oracle. Being open source, plug-ins for MongoDB can be developed for easy work. [6] studied the performance difference among SQL, NoSQL databases such as one open source SQL database (PostgreSQL) and two open source NoSQL databases (Cassandra and MongoDB) with regard to the sensor data storage. It is shown that MongoDB is the best choice for a small or medium sized non-critical sensor application, especially when write performance is important.
In the reference [7] it is stated that it is necessary to detect defects in the code at early stage to assure the quality of the software which can be achieved by using Code review activity. The researchers have analyzed the code review repository of open source software, Chromium with MongoDB as the back end. Before that they addressed seven research questions for which they found interesting answers. The principles and implementation mechanism of Auto-Sharding in MongoDB along with the improved algorithm based on the frequency of data operation is considered by [14]. They claimed that this algorithm improves the concurrent read and write performance of cluster by effectively balancing the data among shards.
The study has performed by [11] in which TPC-H queries were implemented in MongoDB to see the performance difference with the open source RDBMS PostgreSQL. In his paper he found that the performance of the MongoDB was very poor as compared to the PostgreSQL.
The effort is carried out to improve the performance of web services interactions [12]. They determined the execution contexts quantitatively, that make dynamic offloading effective. Also response time is compared with other plain services which gave the expected accuracy in prediction. In case of data-intensive multiple interactions applications Attribute Loading Delegation technique is introduced which enabled the optimized data transfer.
According to [4] distributed NoSQL systems provide high availability for large volumes of data but complex queries are not supported by them. The performance of earlier solution based on inverted lists of single terns in large scale was poor. The solution as a query driven mechanism that stores query terms in combination based on query history is provided that performs well in terms of less bandwidth consumption, improved capacity of NoSQL system and response time with low additional cost of storage. Also it is mentioned that secondary indexes supported by the MongoDB like databases make them transparent to developers. MongoDB has the excellent features like map-reduce, capability to handle complex operations like adding, deleting element to or from the array of documents that make MongoDB very much popular.
Our semantic based study on one of the commercial application is presented earlier to see the performance variation in the normalized and embedded data models. The study showed that the tightly normalized data model is not producing deterministic results in many complex queries containing joins. The embedded data model could perform very well in all the queries.
Now in the current paper we are presenting our experience with another application with more than hundred thousand records.

III. Normalization in RDBMS

The concept of normalization was first introduced in the field of relational database by E.F. Codd. Well organization of data, minimizing update anomalies and maximizing data accessibility are the objectives of the Normalization process. The First Normal Form (1 NF), Second Normal Form (2 NF) and Third Normal Form was his initial contribution which are supported by many commercial case tools. The higher forms like Boyce-Codd Normal Form (BCNF), Forth Normal Form (4NF) and Fifth Normal Form (5 NF) are also invented but they are not commercially applied [5]. But it is experienced by many designers that tight normalization over the database degrades the system performance. Hence these people prefer to de-normalize the data to improve the performance. The denormalization is a process of reducing number of physical tables which are accessed more frequently to reduce the query processing time. Denormalization reduces joins required to design the query to get desired output. In [10] the cloud data management system called ‘SQLMR” is presented. It is developed for compiling SQL-like queries to a sequence of map-reduce jobs. This system is capable of handling tera to petabyte of data in cloud environment. According to [10] the systems like MongoDB, Cassandra, Dynamo and CouchDB are capable to handle large data.
The above summary helped us to decide our research direction. It is also clear from the literature review that the open source NoSQL database MongoDB is suitable in both structured as well as unstrutured data. It can handle the semistructured data very efficiently. As there is no prior

work, we take this opportunity to study semantically the extent of normalization and embedding for which the MongoDB performance for the application is better.

IV. Normalization Vs. Embedding in MongoDB

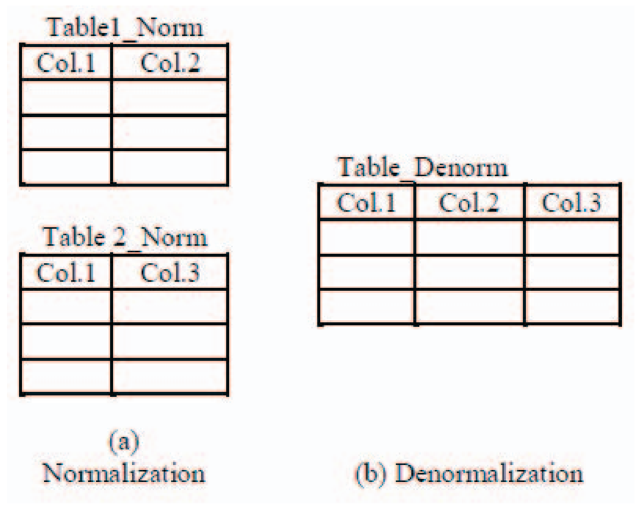
The aim of relational database design is to make group of attributes that form the relations which minimizes data redundancy and reduce the storage space. Normally the relational database should be properly normalized because it increases the data integrity, speed and efficient use of space [3]. But MongoDB does not support joins and so it requires denormalization. The study is carried out to see the performance difference between the normalized data model and embedded i.e. denormalized data model in MongoDB environment.

The basic entity in MongoDB is the document which is the collection of key-value pairs as shown here. There are three fields (columns) viz. _id, Name and Address with their respective values.

{_id:1, Name: “Anuradha”, Address: “Pune”}

The RDBMS ‘Record’ is referred as ‘Document’ in MongoDB. ‘Table’ in RDBMS is referred as the ‘Collection’ in MongoDB that contains multiple documents. The Primary key concept of RDBMS is reflected by ‘_id’ field in MongoDB. The joins can be reflected by linking and embedding. The linking can be done either using Manual References or DBRef.

In the present study the linking of the collections is done by using manual references to achieve normalization. The denormalization can be related to embedding of documents in one collection.



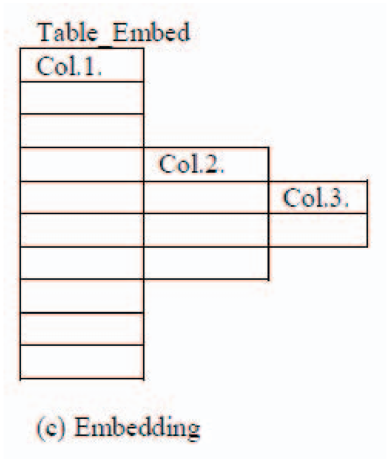


Figure 1- (a) to (c) Data Modeling in Database

Above Figure 1 shows the difference among normalized, denormalized and embedded data model, as (a), (b) and (c) respectively. In (a) common key is used to refer the related tables Table1_Norm and Table2_Norm. In case of denormalization the tables are merged together as shown in (b) whereas (c) is showing the case where the embedding is applied. This is similar to denormalization but still little variation is there. Below is the structure to embedded document.

Embedded document having one-to-one relationship is shown below.

{_id:1, Name: “Anuradha”, Address :{ City:”Pune”, Country:”India”}}

Similarly the embedded document with one- to- many relationship are shown as follows.

{_id: 1, Name: “Anuradha”, Children :[{ Name: “Soham”, Age: 6}, {Name: “Soumya”, Age: 1.9}]

Thus the array of values can also be stored very easily in MongoDB.

In following section the results obtained through the experiments are discussed.

V. Experiment and Results

In Part I the schema design for normalized collections and in Part II, schema with embedded documents is considered for the logical data model named ‘Asset Maintenance’. The documents are inserted in all the MongoDB databases responsible for UNF, 1NF, 2NF, 3NF and embedded schema. Each database contains about 103570 documents except the embedded database. It contains the multiple documents embedded in one another that are corresponding to the 103570 documents in normalized and unnormalized databases.

In Part I of the study, Open source RDBMS, MySQL, is also considered as a reference database for comparing normalized schema performance with MongoDB.

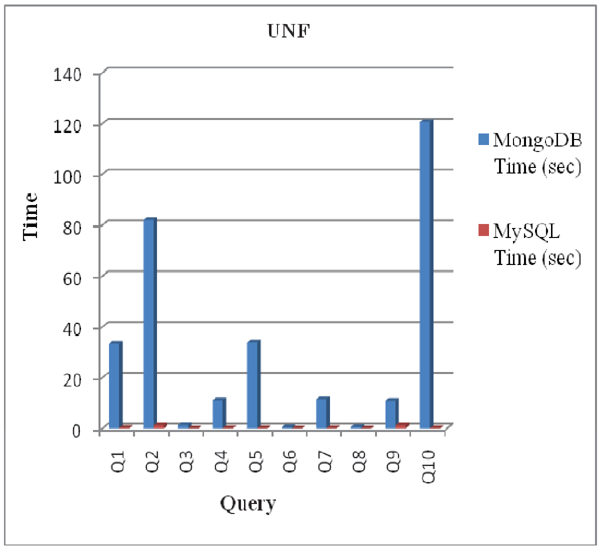
In the same way for Part II, the collection is modeled with the highest possible level of embedding. The level of embedding is six in the present model. The embedded schema results are compared with fully normalized schema results.

The set of queries are designed for normalized and embedded environment. As MongoDB does not support joins, select queries for similar output are modified for both the environments.

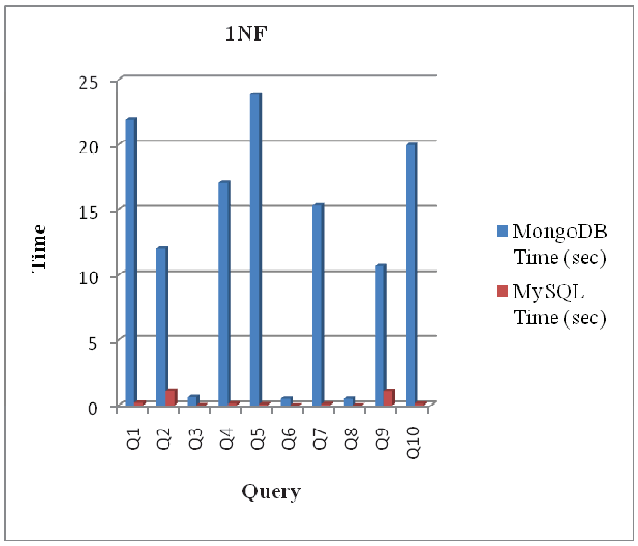
The query processing time is measured accordingly and average query execution time is calculated over 100 executions per query.

A. *Part I – Normalization*

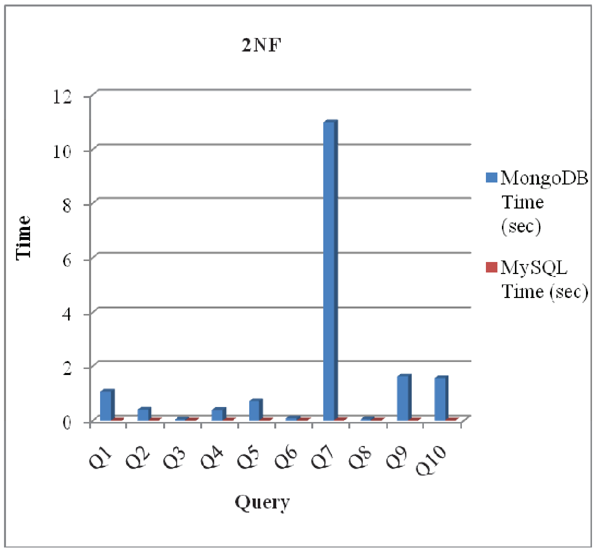
In this case queries applicable for UNF, 1 NF, 2 NF, 3 NF and embedded models are executed for MongoDB and MySQL. In this case we considered the set of common queries that can be applied for all the five databases responsible of each data model as mentioned above. Following graphs (a), (b), (c) and (d) show the query execution time in seconds in unnormalized, 1 NF, 2 NF and 3 NF physical schemas in MySQL and MongoDB respectively.



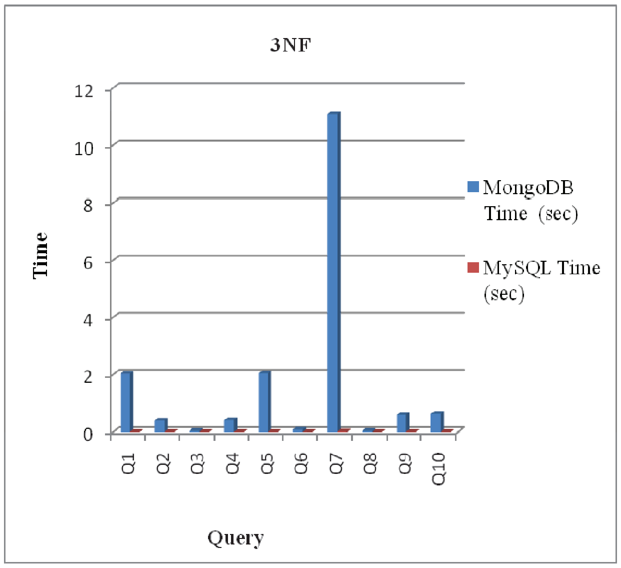
(a)



(b)



(c)



(d)

Figure 2- (a) to (d) Query Execution in Normalization Part

In above Figure 2 four different column charts are displayed. Each column chart shows the difference in query execution time in all databases viz. un-normalized UNF, normalized up to first level 1 NF, normalized up to second level 2 NF and normalized up to third level 3 NF, as (a), (b), (c) and (d) respectively. The time taken in MongoDB is represented using blue color and that of MySQL is represented using maroon color. Q1 to Q10 are the ten different queries applied on the databases that are taken on X-axis and the query execution time in seconds is taken on Y-axis. The set of queries contain ten different queries applicable for all databases. The query execution time in seconds for these queries with respect to the level of normalization is measured for MongoDB and MySQL.

B. *Part II – Embedding and Comparison with Normalization*

In Part II of study, all the data is contained in only one collection which contains embedded documents to represent the relationship between the entities. The embedding level of this data model is six. This means that there are six documents embedded in one another.

The part of embedding of the documents in present embedded database is done as shown below.

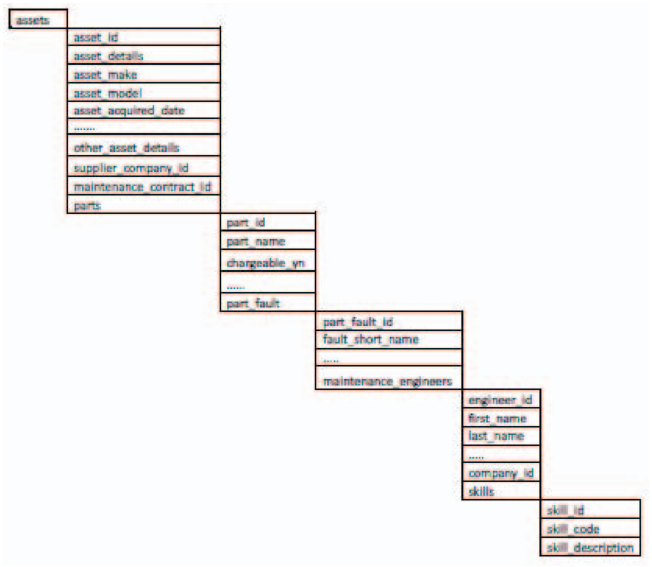


Figure 3-Embedded Document Structure

From the figure we can A single message can have multiple attachments. There is one-to-many relationship between the message and attachment entity. Hence the attachment is embedded in the message document as shown.
The set of queries in embedded environment is generated similar to the normalized part.
The chart below presented in the figure (e) shows the query execution time taken by MongoDB in Normalized and Embedded physical schema separately. The queries are represented by the letters Q1 to Q10 which are taken on x-axis and the query execution time in seconds is taken on Y-axis. The execution time for each query in embedded database is compared with the execution time for the same query in fully normalized database.

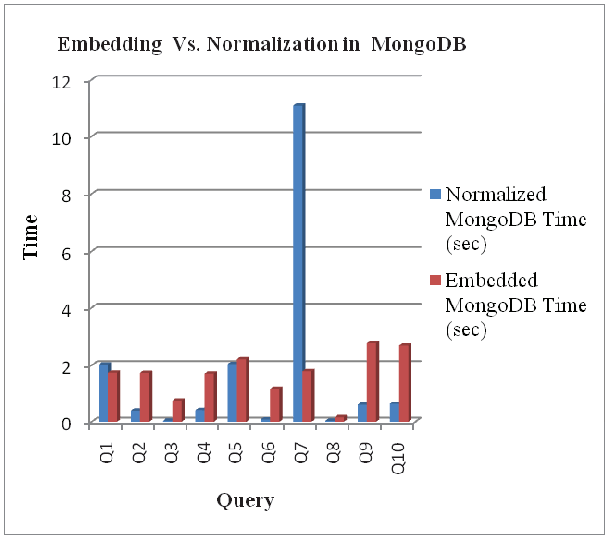


Figure 4- (e) Query Execution in Embedding Vs. Normalization Part

VI. Concluding Remarks

This is clear from the above charts that the MongoDB gives deterministic and comparatively better performance in all the queries.
The 2 NF and 3 NF data model of MongoDB gave very good performance as compared to UNF and 1NF data model for all queries. Many times in case of UNF and 1NF MongoDB databases the query execution fetches all the records. So it takes more time. Query Q7 fetches about 20580 documents in both 2 NF and 3 NF databases. So it takes little more time than other queries.The embedded MongoDB data model gives very good performance as compared to normalized MongoDB data model.
The chart 5 shows the comparative results in Normalized and Embedded databases. It is very clear that the embedded MongoDB database is performing very consistently in case of all the queries. The result for normalized MongoDB database is not much consistent. Embedded MongoDB could produce deterministic and good results for all queries. It performs more consistently. All the results are produced in one or two seconds.

VII. Future Work

The experiment conducted was on particular schema for a commercial application. The similar work can be extended to other schemas for semi structured and unstructured data with about 5,00,000 documents or more to see the performance difference. Our plan is to implement embedding and normalization for those schemas to study difference in the performance and to decide the strategy for determining the extent of normalization and or embedding to get better results with respect to the data modeling style.

ACKNOWLEDGMENT

We are very thankful to our friends and relatives who directly and indirectly helped us during our work.

REFERENCES

[1]. Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin, 2012, "MongoDB vs. Oracle - database comparison", Third International Conference on Emerging Intelligent Data and Web Technologies IEEE, pp. 330-335

[2]. Anuradha Kanade, Dr. Arpita Gopal, Shantanu Kanade , 2013, "Cloud Based Databases- A Changing Trend", International Journal of Management, IT and Engineering, Vol.3, Issue 7, pp. 273-282

[3]. Anuradha Kanade, Dr. Arpita Gopal, 2013, "An Experimental Study on Open Source RDBMS", FDI Issues & Prospects-2013, National Conference, pp. 203-206

[4]. Christian von der Weth, Anwitaman Datta, 2012, "Multiterm Keyword Search in NoSQL Systems", Published by the IEEE Computer Society in IEEE Internet Computing, pp.34-42

[5]. G. Lawrence Sanders, Seungkyoon Shin, 2001, "Denormalization Effects on Performance of RDBMS", Proceedings of the 34th Hawaii International Conference on System Sciences, IEEE, pp. 1-9

[6]. Jan Sipke van der Veen, Bram van der Waaij, Robert J. Meijer, 2012, "Sensor Data Storage Performance: SQL or NoSQL" , Physical or Virtual Fifth International Conference on Cloud Computing, IEEE pp.431-438

[7]. Jun Wei Liang and Osamu Mizuno,2011, "Analyzing Involvements of Reviewers Through Mining A Code Review Repository", Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, IEEE, pp. 126 - 132

[8]. Laurent Bonnet, Anne Laurent, Michel Sala, B´en´edicte Laurent and Nicolas Sicard, 2011, "REDUCE, YOU SAY: What NoSQL can do for Data Aggregation and BI in Large Repositories", 22nd International Workshop on Database and Expert Systems Applications, IEEE, pp. 483-488

[9]. Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes and Jenny Abramov, 2011, "Security Issues in NoSQL Databases", International Joint Conference of IEEE TrustCom-11, IEEE, pp. 541 - 547

[10]. Meng-Ju Hsieh, Chao-Rui Chang, Li-Yung Ho, Jan-Jan Wu, Pangfeng Liu , 2011, " SQLMR : A Scalable Database Management System for Cloud Computing", International Conference on Parallel Processing, IEEE, p.315-324

[11]. Nico Rutishauser, 2012, "TPC-H applied to University of Zuric, "MongoDB: How a NoSQL database performs", A Report submitted to , pp. 1-29

[12]. Quirino Zagarese, Gerardo Canfora, Eugenio Zimeo, Françoise Baude, 2012, "Enabling Advanced Loading Strategies for Data Intensive Web Services", IEEE 19th International Conference on Web Services, IEEE, pp. 480-487

[13]. Wiki, "NoSQL Database History", accessed on 21/09/2013 on Wikipedia source, <http://en.wikipedia.org/wiki/NoSQL>

[14]. Yimeng Liu, Yizhi Wang and Yi Jin, "Research on the improvement of MongoDB Auto-Sharding in cloud environment", Seventh International Conference on Computer Science & Education (ICCSE), 2012 14-17 July 2012, pp. 851 - 854