Research Journal of Applied Sciences, Engineering and Technology 11(4): 434-439, 2015
DOI: 10.19026/rjaset.11.1799
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: April 23, 2015 Accepted: May 22, 2015 Published: October 05, 2015

Research Article

From Relational Databases to NoSQL Databases: Performance Evaluation

Ahmad Abdullah and Qingfeng Zhuge
College of Computer Science, Chongqing University, Chongqing 400044, China

Abstract: In nowadays applications, the amount of data in the database grows exponentially. So, the DBMS must process these huge amounts of data as fast as possible. The main aim of this study is to prove that NoSQL databases process big data faster than relational database. The changing in applications, user and infrastructure characteristics, mostly of the Web 2.0 domain and cloud platform, led to explosion of data sources and massive workloads. These huge amounts of data have raised the problems of storage and usability of data as the usual Relational Database Management Systems (RDBMS) were unable to handle the exponentially growing data on a single in house server (vertical scalability). Furthermore, the required time to process these big data is an issue. The study implements prototype which verifies performance argument. The performance evaluation compares the insertion and retrieval speeds between MongoDB as NoSQL database and MySQL as relational database. The benchmarking performed shows that MongoDB is faster than MySQL in the most of scenarios we chose, particularly when we deal with huge amount of data.

Keywords: MongoDB, MySQL, NoSQL databases, performance evaluation, relational databases, speed comparison

INTRODUCTION

Since the 1970s, the relational databases and the associated entity relationship models have together been the standard for database development (Harrington, 2009). Choosing between databases was limited to the study of differences between available commercial and open source relational databases. They provide the users with the best mix of simplicity, robustness, flexibility, performance, scalability and compatibility (Plugge *et al*., 2010). However, the changing in applications, user and infrastructure characteristics, mostly of the Web 2.0 domain (Hecht and Jablonski, 2011) and cloud platform, led to exponential growth of Internet, the explosion of data sources and massive workloads. This kind of data is usually referred to as Big Data (Chang *et al*., 2006). Relational databases are found to be inadequate in handling Big Data applications. Also, multiplication of data sources and types has led to the problem of storing and manipulation of these unstructured data by structured data models provided by RDBMS. NoSQL databases enjoy schema-free architecture and possess the power to manage highly unstructured data. They can be easily deployed to multi-core or multi-server clusters serving modularization, scalability and incremental replication. NoSQL databases being extremely scalable, offer high availability and reliability, even while running on hardware that is typically prone to failure. In this study, the comparison focuses on time consumption. So that, DBMS which takes less time in processing huge amount of data will be more favorable. The main objective of this study is to prove that the performance of NoSQL databases is better than relational databases in the cases of Inserting and filtering data. Even though, we do not cover all scenarios available, we give a good picture of the speed differences between selected databases.

Challenges with relational database: With the continuous development of the Internet and cloud computing, various types of applications have emerged, which made database technology more demands, mainly in the following aspects (Bhat and Jadhav, 2010; Han *et al*., 2011).

High concurrent of reading and writing with low latency: Database were demand to meet the needs of high concurrent of reading and writing with low latency, at the same time, in order to greatly enhance customer satisfaction, database were demand to help applications reacting quickly enough.

Efficient big data storage and access requirements large applications need: Database to meet the efficient data storage and can respond to the needs of millions of traffic.

High scalability and high availability: With the increasing number of concurrent requests and data, the database needs to be able to support easy expansion and upgrades and ensure rapid uninterrupted service.

Corresponding Author: Ahmad Abdullah, College of Computer Science, Chongqing University, Chongqing 400044, China
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

Lower management and operational costs: With the dramatic increase in data, database costs, including hardware costs, software costs and operating costs, have increased. Therefore, need lower costs to store big data.

Although relational databases have occupied a high position in the data storage area, but when facing above requirements, it has some inherent limitations.

Slow reading and writing: A relational database itself has a certain logic complexity, with the data size increases, it is prone to bring about deadlocks and other concurrency issues, this has led to the rapid decline in the efficiency of reading and writing.

Limited capacity: Existing relational database cannot support big data in search engine or Big System.

Expansion difficult: Multi-table correlation mechanism which exists in relational database, became the major factor of database scalability.

NoSQL databases: As late as 2008, relational databases were both commercially dominant and well entrenched in the development community. However, a new group popularly known as "NoSQL databases" has emerged (mostly in large web applications) to challenge the supremacy of relational databases and address the shortcomings in them. This class of storage engine seeks to breakdown the rigidity of the relational model, in exchange for leaner models that can perform and scale at higher levels, using various models (including key/value pairs, column oriented databases and document oriented approaches) which can be created and read efficiently as the basic unit of data storage. Primarily, these new technologies have arisen in situations where traditional relational database systems would be extremely challenging to scale horizontally to the degree needed for global systems.

Even though, NoSQL databases don't have specific definition, we adopted this definition: A NoSQL database is a database that is not an SQL database. Data is not stored in relations and the main query language to retrieve data is not SQL.

NoSQL obviously have general common characteristics:

- Ability to horizontal scalability, not using the relational model (nor the SQL language)
- Ability to replication and partitioning data across multiple servers
- Access to a simple API
- A weaker concurrency model than the ACID transactions of most relational (SQL) database systems
- Flexible Scheme, allowing fields to be added to any record without controls

Introduction to mongoDB: Chodorow and Dirolf (2010) MongoDB[1] is a schema less document oriented database developed by 10 gen and an open source community. The MongoDB is intended to be scalable and fast and is written in C++. In addition to its document oriented databases features, MongoDB can be used to store and distribute large binary files like images and videos. So MongoDB could be used as a file system. MongoDB stores documents as BSON (Binary JSON) objects, which are binary encoded JSON like objects. MongoDB is a document-oriented database with no transactions and joins. So it is easier to write queries. Each document has an ID field, which is used as a primary key. To enable fast queries, the developer can create an index for each query able field in a document. MongoDB also supports indexing over embedded objects and arrays.

Documents in MongoDB can be organized in so called "collections". Each collection can contain any kind of document, but queries and indexes can only be made against one collection. But MongoDB has restriction of indexes number per collection. Relations in MongoDB can be modeled by using embedded objects and arrays.

With all these features, MongoDB has many advantages. It is extremely fast. And it is easy to adapt quickly as requirements change.

METHODOLOGY

We intended to test and compare two different types of databases, relational databases such as MySQL[2] and Document Oriented NoSQL databases such as MongoDB. We used (PHP) as a programming language. We generate a pair of test one for each DBMS and measure which of them performs the fastest. Testing should reflect real-world scenarios as possible as we can, in order to find out how much faster is MongoDB instead of MySQL in real applications. The software we choose determines the possible ways to implement our use cases depending on the available features. The choosing of these two databases is partly because MySQL is one of the most popular databases in website applications. We chose Document Oriented NoSQL database, because it has rich data model and it is considered a real alternative solutions by a lot of companies. MongoDB is relatively new and was released in 2009. However, according to previous studies which compared MongoDB with CouchDB, MongoDB was speeder than CouchDB in most tests. That makes the choosing of MongoDB more exciting to make comparisons with MySQL. This prototype is simple database model based on MySQL and MongoDB to compare the performance. We also present the results of the performance test, showing which one of the tested databases is the best suited one for the type of data that we chose to store. Even though, we do not cover all scenarios available, we give a good picture of the speed differences between selected databases.

Performance benchmarking: To test the performance of the MongoDB and MySQL we developed a test suite in PHP. Every benchmark will be run on different object sets with different sizes, because we need to see how the databases scale. The tasks are generated randomly, thus every query differs and it is very hard for the database to cache any query. The tests are always running under the same user using the default database settings. First of all, we have two scenarios.

Data insertion: This benchmark tests the speed of inserting a lot of data objects. Both MySQL and MongoDB support inserting all data in a single request (Bulk inserts[3]). Experiments will perform both type of inserts in order to see the differences between both types.

Data retrieval: This benchmark will measure the time to query objects with a specific value. In relational databases this will test the join behavior and in document oriented databases the speed of querying objects.

Every individual test case, however, has been run five times. And then the average value will be calculated. That because the times may vary between each run.
Software and Hardware:

- Microsoft Windows7 64-bit
- PHP 5.3.5
- MySQL 5.1.62
- 10 gen MongoDB 2.0.5
- php_mongo v 1.3
- Sony vaio VPCCA (laptop)
- Intel core i5, CPU 2.30 GHz (I used only 40% from CPU)
- 4 GB RAM
- **Client libraries:** During the implementation of the performance benchmark, we used php_mongo as client libraries for PHP programming language. There are much more libraries for almost every programming language, but we used the official one[4]
- **Databases setting:** With MySQL, we test using the memory engine for Member Table. And because memory engine doesn't support Blob data, I chose MyISAM engine for Image table. Even though comparing a disk engine in MongoDB to a memory engine in MySQL sounds unfair, the other MySQL engine can't compete with MongoDB at all i.e., they took a lot of time comparison with MongoDB.

RESULTS AND DISCUSSION

Data insertion scenario: The following subsections showcase the results of the write (insertion) speed benchmarks performed. They answer the questions of which DBMS, MySQL or MongoDB, is the fastest at different amount of data.

Relational data model is based on MySQL that mainly contain Member and Image tables Fig. 1.

Document oriented NoSQL data model is based on MongoDB that contain Member document. All members will insert into one document and each collection has this structure:

Member = {
"_id": object Id ("51603a46d06858501d000000"),
"firstName" : "MzdBk8",
"username" : "sjk8J3ByD",
"lastName" : "WzUKPhKRchT8",
"age": 57}

Notice that we used different types of data such as (string, integer and image) to see if it may affect the time of insertion and to find out the differences in implementation.

The cost time of insertion into MongoDB and MySQL were recorded in the tables according the different numbers of rows/documents and different test cases.

Insert to "members" object: In the first experiment, we performed "single insert" tin order to answer the question: how fast is MongoDB and MySQL, respectively when inserting 1000 documents/rows, 10000 documents/rows and 100000 documents/rows? Table 1 and Fig. 2.

Figure 2 clearly shows, MongoDB is faster than MySQL at inserting objects. Approximately, MongoDB came out two times faster that MySQL in case of insert queries.

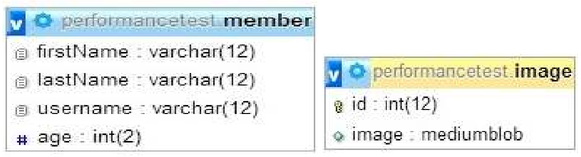

Fig. 1: Member and image tables

Table 1: The results of insertion members into databases

		MySQL	MongoDB
Number of objects	1000	0.1810	0.0762
	10000	1.8720	1.0918
	100000	18.5432	10.8424

Average processing time in second

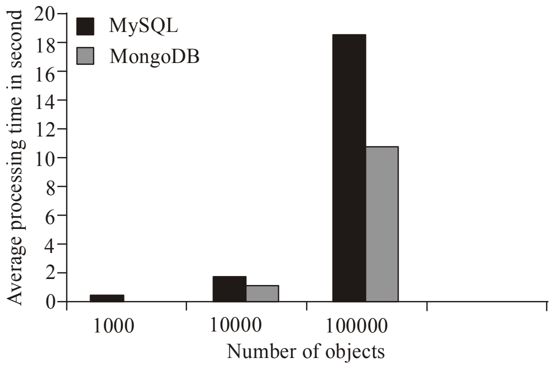

Fig. 2: Graph visualizing insert speeds of MongoDB and MySQL

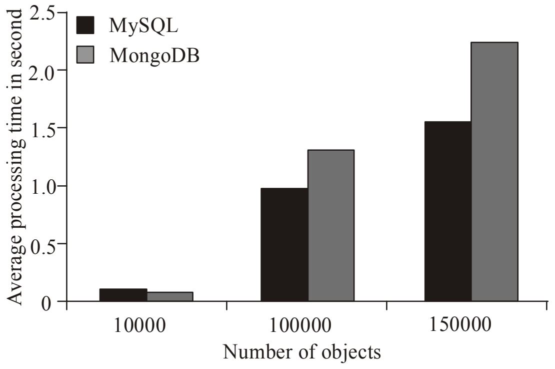

Fig. 3: Graph visualizing bulk insert speeds of MongoDB and MySQL (member table)

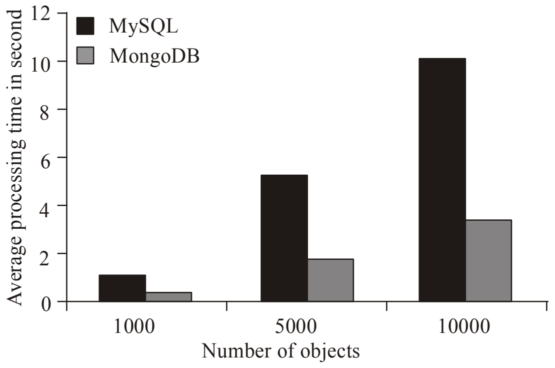

Fig. 4: The results of insert images into databases

And then we performed "Bulk (multi) insert" for 100000 members and more. We won't test small quantities of objects, because it is not relevant Table 2.

Surprisingly, according to the Fig. 3. MySQL came out one time faster than MongoDB in case of insert queries with bulk feature Fig. 3.

For some databases, bulk insertion is just a wrapper for sequential insertion. For others, bulk insertion could be implemented optimally knowing the architecture of TCP and file buffers. We didn't investigate how these databases implement bulk insertion, but MySQL surprised us. Comparing the results, we noticed that bulk insertion is overall quite faster than sequential insertion. However, MySQL is the winner in this test case.

Table 2: The results of bulk insert members

		MySQL	MongoDB
Number of objects	10000	0.1142	0.0890
	100000	0.9650	1.3016
	150000	1.5418	2.2156

Average processing time in second

Table 3: The results of insert images into databases

	Number	MySQL	MongoDB
Number of objects	1000	1.0577	0.3943
	5000	5.2277	1.7213
	10000	10.0027	3.4570

Average processing time in second

Insert to "image" object: Here, experiment performs insertion for different data type. We insert here images. We won't test big quantities of images, because it is take long time in MySQL. However, MongoDB in this type of data is defiantly the best Table 3 and Fig. 4.

As the graph clearly shows, MongoDB is a whole lot faster (more than three times) than MySQL at inserting images. The difference is very big particularly when we insert more than 10000 images. We tried to insert 100000 images into MongoDB, it was great comparing with MySQL which take very long time. So, MongoDB is very suitable for multimedia data types.

Data retrieval scenario: The following subsections showcase the results of the read (retrieval data) speed benchmarks performed. They answer the questions of which DBMS, MySQL or MongoDB, is the fastest at different amount of data.

Relational data model is based on MySQL that mainly contain product and category tables. Figure 5, shows the ERD and the many to many relationship between product and category entities.

We represent many to many relationship between product and category entities in relational database as following Fig. 6.

Document oriented NoSQL data model is based on MongoDB that contain product document. We can put all information about product in one document, because of flexibility which NoSQL data model has. All products will insert into one document and each collection has this structure:

```
//String generated randomly so, it has no meaning
    Product = {"_id": Object Id ("51701d51d06858e
    421000000"),
    "title" : "MvcPFxMUuHNT",
    "description":
    "rj1MLU0r6I4prjV2fMO8uJp4DfXHEVbk1RiPG
    E1x8nzj1DFtKx
    wSmERRC5ThurmssNsguKU13oWA7khh8E7z6
    Wb1siIeckpXM6u
```

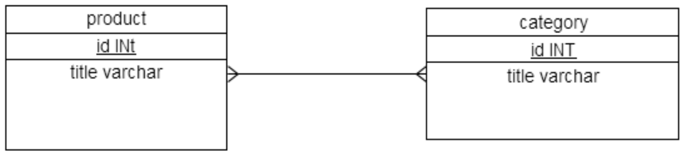

Fig. 5: ERD for product and category entities

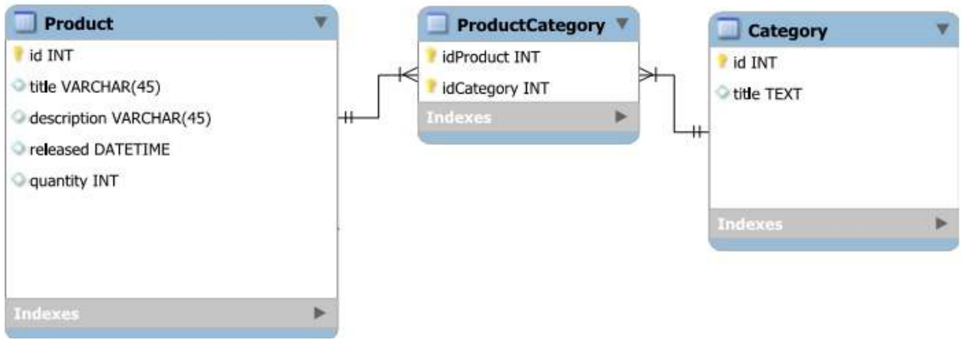

Fig. 6: Tables for product and category entities

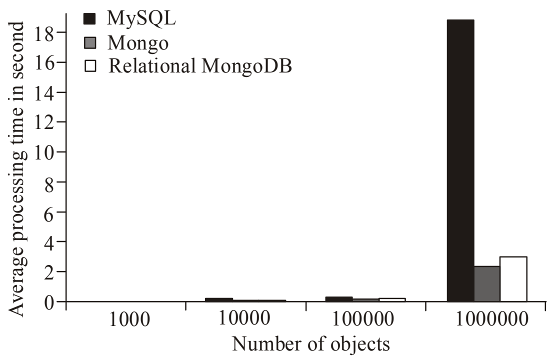

Fig. 7: Graph visualizing retrieve speeds of MongoDB (without and with relationship) and MySQL

Table 4: The results of retrieve members

	Number	MySQL	MongoDB	Relational mongoDB
Number of objects	1000	0.042	0.030	0.036
	10000	0.093	0.077	0.092
	100000	0.243	0.173	0.214
	1000000	18.620	2.400	3.100

Average processing time in second

3aeGRQNcdLpLto9At",
"quantitity" : 74,
"productCategory" : ["W1WPxLo", "DGA", "Atj", "5jSU7jRuWp",
"Yiys", "tp3PXAp4", "brYYT3o2WVGZ"]
}

Scheme free is one of the characteristics of NoSQL. NoSQL data model usually denormalize data. In MongoDB, we can use references. To normalize data, MongoDB stores references between two documents to indicate a relationship between the data represented in each document.
In general, use normalized data models:

- When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication
- To represent more complex many-to-many relationships
- To model large hierarchical data sets

In this case, we divide the document into two documents products and categories. Products has embedded document to stores references. This embedded document called product Category.

The following code represents the products structure in "relational" MongoDB:

Product = {"_id": objectId ("51704897d06858ec020000c8"),
"title" : "MvcPFxMUuHNT",
"description" : rj1MLU0r6I4prjV2fMO8uJp4DfXHEVbk1RiPGE1x8nzj1DFtKxwSmERRC5ThurmssNsguKU13oWA7khh8E7z6Wb1siIeckpXM6u3aeGRQNcdLpLto9At",
"quantity" : 74,
"productCategory" :
[ObjectId ("51704896d06858ec0200004e"),
ObjectId ("51704896d06858ec0200006f"),
ObjectId ("51704896d06858ec02000026"),
ObjectId ("51704896d06858ec020000b2"),
ObjectId ("51704896d06858ec02000050"),
ObjectId ("51704896d06858ec02000070"),
ObjectId ("51704896d06858ec0200009d")]

And this is the categories structure:

Category = {"_id": objected ("51704896d06858ec02000000"), "title" : "MzdBk8"}

We designed a query which has "INNER JOIN" and "OUTER JOIN" to prove our thinking that NoSQL database has advantage over relational database.

The cost time of retrieval data from MongoDB (with and without relationship) and MySQL were recorded in the Table 4 according the different numbers of rows/documents and different test cases Table 4.

Figure 7 clearly shows, MongoDB is a whole lot faster than MySQL at inserting objects. The increase in time for both DBMS seem to be linear. When we deal with huge amount of data the difference is very big between MongoDB and MySQL (e.g., when we retrieve 1 million objects, MongoDB is 6 times faster than MySQL) Fig. 7.

The differences between MongoDB with and without relationships is noticeable, but in this test case was not particularly critical. In 1 million objects it differs about 1 sec, which is not seen very much in comparison to MySQL. So, we can implement our applications with relationships where urgent need to use structure data.

CONCLUSION AND RECOMMENDATIONS

In the Cases and circumstances they were covered in this Study, we discovered that moving from MySQL to MongoDB, it is possible to get a significantly faster database with a relatively similar structure.

In the most of data insertion scenarios, MongoDB is very good particularly in images insertion. In the data retrieval scenarios, the test shows that MongoDB is faster than MySQL in the most of test cases, particularly when we deal with huge amount of data.

The experiments presented in this Study have tested only on a single server, but with data shared across clusters, things might look different. This should take in consideration in the future work for an interesting experiment. There are also a lots of additional tests that can be done for future work such as: Other types of queries and data models.

REFERENCES

Bhat, U. and S. Jadhav, 2010. Moving towards non-relational databases. Int. J. Comput. Appl., 1(13): 40-46.

Chang, F., J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes and R.E. Gruber, 2006. Bigtable: A distributed storage system for structured data. Proceeding of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06). Berkeley, CA, USA, pp: 15-15.

Chodorow, K. and M. Dirolf, 2010. MongoDB: The Definitive Guide. O'Reilly Media, Sebastopol, CA, USA.

Han, J., E. Haihong, G. Le and J. Du, 2011. Survey on NoSQL database. Proceeding of the 6th International Conference on Pervasive Computing and Applications (ICPCA, 2011), pp: 363-366.

Harrington, J.L., 2009. Relational Database Design and Implementation. 3rd Edn., Morgan Kaufmann, Burlington,.

Hecht, R. and S. Jablonski, 2011. Nosql evaluation. Proceeding of International Conference on Cloud and Service Computing, pp: 336-41.

Plugge, E., T. Hawkins and P. Membrey, 2010. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing. 1st Edn., Apress Berkely, CA, USA.

End notes:

[1]: http://www.mongodb.org/

[2]: https://www.mysql.com/

[3]: A Bulk insert is a process or method provided by a database management system to load multiple rows of data into a database table

[4]: https://github.com/mongodb/mongo-php-driver