Performance Comparison between MongoDB and MySQL

Yuqing Yang
Computer Science Department
yyang6@wpi.edu

Ruizhu Yang
ECE Department
ryang2@wpi.edu

Pengfei Geng
ECE Department
pgeng@wpi.edu

I. INTRODUCTION

For past decades, to access data is common to use Relational Database Management System (RDBMS), which uses SQL as its query language. This could be regarded as a collection of mechanisms with functions of storing, editing and extracting data. Over past years, RDBMS has become a synonym of database, whose size and complexity are defined by number of registers used. Besides, as the storage type, functionalities and interaction with databases has improved, databases has become a daily-use resource in millions number of applications. With the exponential growth of size of database, speed of accessing to data becomes more important. As all data should be structured and organized in best way to extract values, the efficiency in information extraction becomes a well-known problem.

To solve this problem, different types of databases emerged. For NoSQL, it can deal with unstructured data. Data could be semi-structured, or have different characteristics. Besides, there is no fixed schema for these data, which can be of any type and be any format.

As we know, ACID, as typical features of SQL databases, requires a large amount of overhead. In NoSQL database, they are relaxed or eliminated in order to maximize database performance. Some NoSQL database organize the data into documental type, items might be different structures, or into key-value pairs, whose value can be any type or a complex structure with unique semantics. For query, there is no standard query language, and obviously many limits to the operations, such as no join operation. However, processing of NoSQL database is more affordable, simpler and more flexible.

The Internet faces the challenges with the increase of the numbers of people that can be access to Internet and the higher demand about the network capacity. However, traditional database is difficult to meet the need of high performance, huge storage and high availability and scalability. Therefore, how to use the network resource effectively and cheaply is a key point to meet the everyday need of people. For this, a number of paradigms have been proposed, of which the latest one is known as Cloud computing [1]. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [2]. Cloud computing is based on huge amounts of data distributed storage technology, huge amounts of data management technology and Cloud computing platform management technology

In this project, we compare MongoDB, one of NoSQL solutions, to MySQL, the traditional SQL relational database.

The evaluation will analyze their differences of managing data and test speed in insert, updata, and select operation. For this paper, we will summarize the common results from related papers and related works. Then, there is an analysis of the experiment results, as well as a prediction of the performance between MongoDB and MySQL. Next, some extended questions would be suggested, one is about the problem of searching NoSQL, as there is no fixed schema of data. One is trying to combine the more than one relations in our further tests. We hope there comination could bring advantages of searching operation because Redis is a key-value type data, has commonality with relational data types.

The reminder of this paper is organized as follows. Section 2 is related work on related topic. In Section 3 there is a brief introduction of NoSQL database. Section 4 describes the experiment method and prediction. Section 5 show the results of experiments. Section 6 describes three extended questions about this topic.

II. RELATED WORK

Performance of NoSQL databases has been approached for a long time. These analyses mainly focus on two types: comparison between NoSQL databases, and studies of relational SQL databases and NoSQL ones.

Veronika Abramova et al. made a comparison between MongoDB and Cassandra, document-oriented data and column-type data. [3] Their method is inspiring in our work. Brain F.Copper et al. also studied performance between NoSQL and MySQL using YCSB benchmark about latency of operations per second. [4] Its results do help in prediction of our experiment. In reference “NoSQL evaluation” [5], Robin Hecht and Stefan Jablonski compared query possibilities, concurrency control, partitioning, replication and consistency about the four kinds of NoSQL databases mentioned above. They found that. Since no query language is common for all four kinds, every store differs in its supported query feature set, and developers have to trade between high performance through partitioning and load balanced replica servers, high availability supported by asynchronous replication and strict consistency. In general, key value is used for fast and simple operations, document stores offer a flexible data model with better query possibilities, column family stores are suitable for large datasets, and graph databases should be used in where entities are as important as the relationships between them.

As the development of large-scale distributed service and distributed storage which clouds computing needs, traditional relational database are facing many new challenges, especially in scenarios of the large scale and high concurrent SNS applications, by the use of a relational database to store and inquires users' dynamic data has been ragged and exposes many problems to overcome, such as there need high real-time insert performance; Need mass data storage capacity, still need high retrieval speed; Need to store data seamless spreading throughout the distributed cluster environment, and have ability of online expansion, etc. According this, NoSQL is made.

NoSQL database broke the paradigm constraint of traditional relational database. From a storage perspective, NoSQL is not a relational database, but a hash database with format of Key-value. Due to the abandoned powerful SQL query language, transaction-consistent and constraints in paradigm of relational database, NoSQL database largely solved challenges which the traditional relational database is facing.

III. NoSQL Introduction

Web applications motivate the trend of NoSQL since the relational database cannot meet the requirement of the mass data and flexible modification of the data. In order to deal with these requirements, many companies clusters many hardware machines to get the support. Relational database is not suitable for these tasks, because join and locks influence the performance of this kind of distributed system. For these requirement, database must be replicable easily and can provide an integrated machine to deal with node failure. In relational database, replication is limited by the consistency, these requirement can only be achieved by extra efforts. Due to these requirements, many companies have developed their own NoSQL storage systems. And every system have their own characteristic and meet different requirement. And the mainly classification of NoSQL database is as follow:

A. Key Value Stores

Key value stores is similar to maps data by a unique key. Since values are represented by arrays that are not opaque to the system, keys are the only method to retrieve data. Values are independent from each other wherefore relationships must be handled in application logic. Because the schemas of data structure is very simple, key value stores are free for structure. The only possible to add some kind of structure in to the data schemas is to group of key value pairs into collection. Key value stores are useful for simple operations, which are based on key attributes only.

B. Document Stores

Document Stores encapsulate key value pairs in ISDN or ISDN like documents. Within documents, keys have to be unique. Every document has a particular key that is unique within a collection of documents. User can find the document through this unique key. Complex data structures like nested objects can be handled more conveniently in this kind and it has advantage of supporting data types to make document stores easily. Like key value stores, document stores do not have any schema limitations.

C. Column Family Stores

Column Family Stores are extensible record stores and wide columnar stores, which is a "distributed storage system for managing structured data that is designed to scale to a very large size" [2]. Since values in this kind cannot be interpreted by the system, it cannot support dataset relationship and other data types than strings. Multiple versions of a value are stored in a chronological order to support better performance and consistency. Columns can be grouped by column families. And columns and rows can be added flexibly when running but user needs to pre-definite column families.

D. Graph Databases

Unlike other kinds of NoSQL databases, graph databases are specialized on efficient management of linked data. Therefore, this kind of database is more suitable for applications that based on data has strong relationship. Since cost intensive operations like recursive joins can be replaced by efficient traversals [3].

For this comparison task, we select MongoDB for experiment. MongoDB has document-based data model. It has rich data structure capable for holding arrays and other documents and we can use any popular programming language to handle on MongoDB. That is, we can represent in a single entity a construct that require several tables to represent in a relational database. This is very important and useful for immutable data. Its deep query-capacity supports dynamic queries on documents using a document-based query language almost power as SQL. In addition, no schema migration and easy to horizontal scalability are attractive. For relational database,

IV. Experiment Methods

In project, we will verify how these two databases will respond to read/update mix while running just on one node without a lot of memory and processor resources, just like personal computers. For initial setup, software are running on Intel i5 dual core processer with DDR3 Ram of 4GB. We will compare performance of both databases in the following aspects: insert speed, update speed, and query speed. we select MySQL, one of the most popular RDBMS nowadays.

We use N+1 tables for test. N is the different size of the same content, i.e., People in Table 1. We set the people100, people1000, people10000, people100000, people1000000 table of size of 100, 1000, 10000, 100000, 1000000 tuples. We can change the number of tuples for insert, query and update by changing the parameter and we can also change the query condition and update condition to do other query and update.

In People relation, there are three columns: ID is the Identical number of every tuple in the relation; pname is names of person and cid is the corresponding number of city. For the insert operation, we circle there ten tuples by 10 times to 100000 times to create the 100 tuples to 1000000 tuples tables. In City relation, we have ID and cname, which are the ID and names of city. Cid in People table has the primary key on ID in City table.

People		
ID	pname	cid
1	Mary	6
2	Ann	5
3	Bob	8
4	Rachel	4
5	Helen	3
6	Steven	2
7	Carrie	1
8	John	8
9	Jim	4
10	Zach	7

City	
ID	cname
1	London
2	Paris
3	New York
4	Boston
5	Beijing
6	Worcester
7	Tokyo
8	Rome

Table 1. Data Schema

By using this data schema, we test three operation speed, insert speed, update speed, and select speed. We test it on simple table, join table, table with and without index on pname. Join is easy done by MySQL, but it’s a little complicated in MongoDB, which is called MapReduce.

The test tool of MongoDB is Mongostat and Mongotop. The mongostat utility provides a quick overview of the status of a currently running mongod or mongosinstance. [6]

Mongostat is functionally similar to the UNIX/Linux file system utility vmstat, but provides data regarding mongod and mongos instances.

And mongotop provides a method to track the amount of time a MongoDB instance spends reading and writing data. mongotop provides statistics on a per-collection level. By default, mongotop returns values every second.[6]

For MySQL, the test tool is mysqlslap. Mysqlslap is the default test tool of benchmark for MySQL. It can generate schema, install data, execute benchmark and query. It is simple, flexible and easy to operate.

Our tests includes of five separate tests with 50 runs for each test and we take the average of the total time length for each tests as the final result. Table 1 shows the record we used for the tests. As we can see in Table 1, we use the same record for the operation of insert, query and update. For the tests of SQL and NoSQL, we use the same record for test.

For the experiments, we use java driver to achieve batch inserts, batch query and batch update in MongoDB and SQL syntax for MySQL operations. For insert portion, we inserted the record in different database for 100, 1000, 10000, 10000 and 1000000 times respectively. And for update portion, we use the database we inserted at the previous step and update the 100, 1000, 10000, 10000 and 1000000 tuples respectively. For query portion, the operation is like the operation in update tests, we also query the 100, 1000, 10000, 10000 and 1000000 tuples respectively.

For MySQL, there are two engines, MyISAM and Innodb. There are several differences between these two engines, so we operation each test on these two engines separately. MyISAM is not ACID compliant and non-transactional. It offers compression and requires full repair of index. It changes DB pages written to disk instantly and do not have ordering in storage of data. While Innodb is ACID compliant and fully transactional with ROLLBACK and COMMIT and supports for foreign keys. It also offers compression and auto recovery from crash via replay of logs. The row data stored in pages in PK order.

From the Figure 2 to Figure 4, we show the experiment results of engine tests on MySQL. To test this two engine, we use MYSQLSLAP to test it under Ubuntu system in the same computer hardware. MySQLSlap slaps the MySQL tables with a load of queries at a rate and concurrency that you set. After the queries have ran it will produce a small table breaking down the times for you, which allows you to easily compare and contrast. Here is the code for example on managing MySQLSLAP.

```
“mysqlslap --user=xx --password –auto-generate-sql “
```

```
“mysqlslap -uxx -p --auto-generate-sql --auto-generate-sql-load-type=write --engine=xx --number-of-queries=xx -uxx –p”
```

The first code is the automatically generate an SQL table and second one is add some parameters. We could choose the type we test, the engine, the concurrency, and even the iteration.

We test two engine in Read, Write and Update, the results are represented by Figures below.

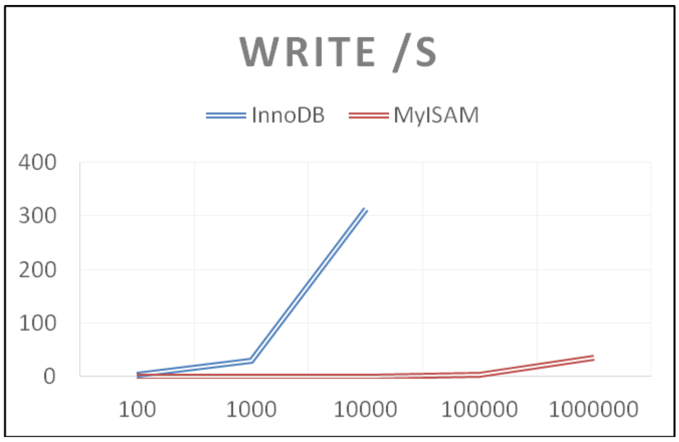

Figure 2. Insert Speed of MySQL

Figure 2 shows the insert speed of the two engines of MySQL. We can see that the insert speed of InnoDB are almost the same as the speed of MyISAM when the tuples are less than 1000. While, when the tuples increase, MyISAM is still fast in insert but InnoDB becomes much slower than before. Obviously, MyISQM has advantages in situation that insert amount of data.

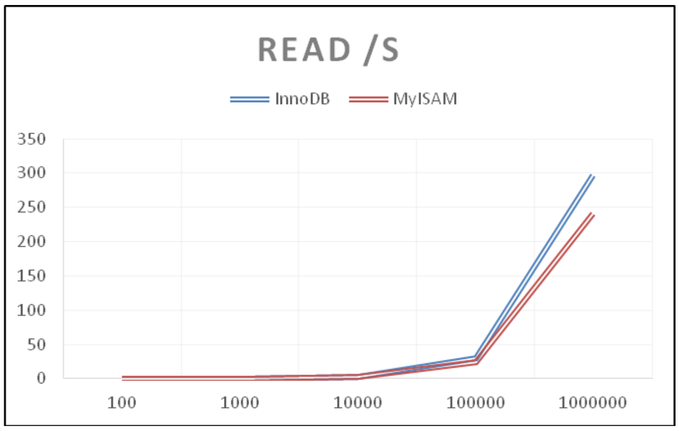

Figure 3. Query Speed of MySQL

Figure 3 shows the query speed of the two engines of MySQL. As shown in the figure, from 100 tuples to 1000000 tuples, the speeds of two engines are almost the same. And they have a relative fast speed when tuples are less 100000. And compared with insert speed, they have a better performance in query for small size data.

In Figure 4, the update speed of MySQL varies from InnoDB and MyISAM. Through Figure2 to Figure 4, the UPDATE speed and WRITE speed between two engine is very different, however, the read speed is similar. The reason is the parameters set of the two engines.

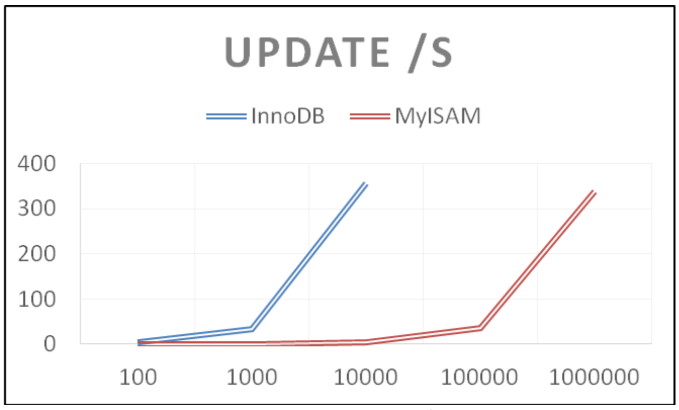

Figure 4. Update Speed of MySQL

From the above three Figures, we know that in insert, query and update, MyISAM always have better performance than InnoDB. For MyISAM, the performance of insert is better than query and update. And the performance of query and update are almost at the same level. So, as the analysis above, we know that it is because of locking and ACID concern, which makes InnoDB engine slower on Update and Insert.

V. Experiment Results

For Figure 5 to Figure 7 in the following, they are the performance speed of MongoDB. Figure 8 is the comparison among them.

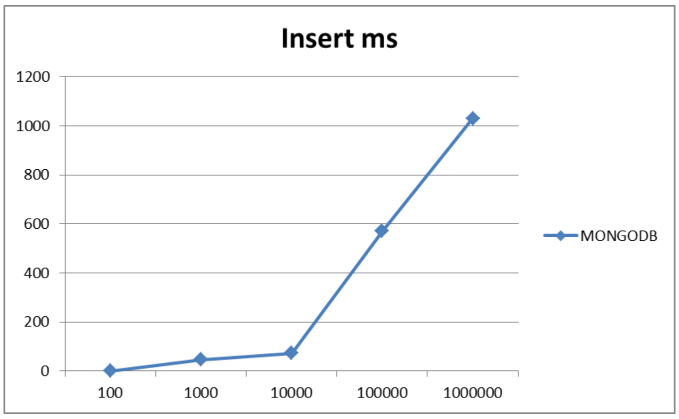

Figure 5. Insert Speed of MongoDB

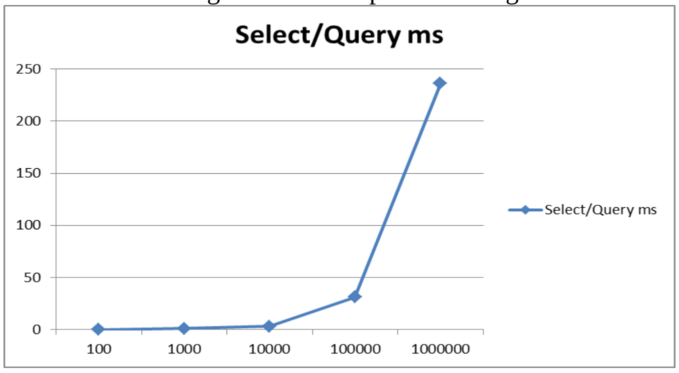

Figure 6. Query Speed of MongoDB

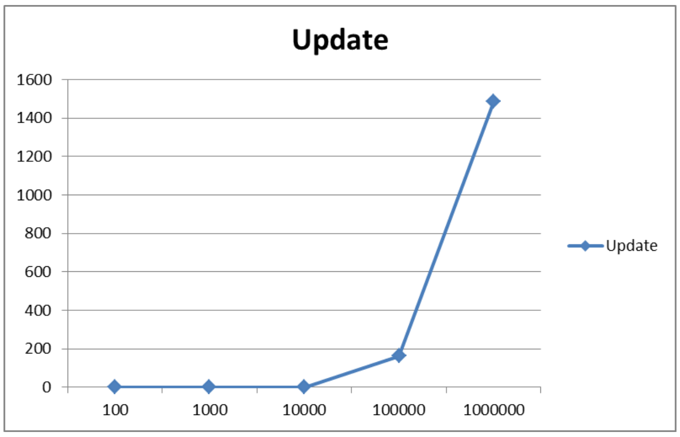

Figure 7. Update Speed of MongoDB

For MongoDB, we can see that all of these three operation cost less time than MySQL. For insert portion, when number of tuples is less than 10000, the time is almost zero. When tuples increases to 1000000, it only costs 3 second for insert which is a pretty good performance. And for query portion, MongoDB has nearly the same performance as insert. When the number of tuples is less than 10000, the time cost for query is almost zero, and when the tuples is larger than 10000, time cost will increase largely. And the performance of update is not such great as insert and query. The time cost of update is almost as twice as the time cost for query and update for the same number of tuples. Maybe MongoDB is good at insert and query much better than update.

The following figures are the comparison between MongoDB and MySQL in the three performance aspects.

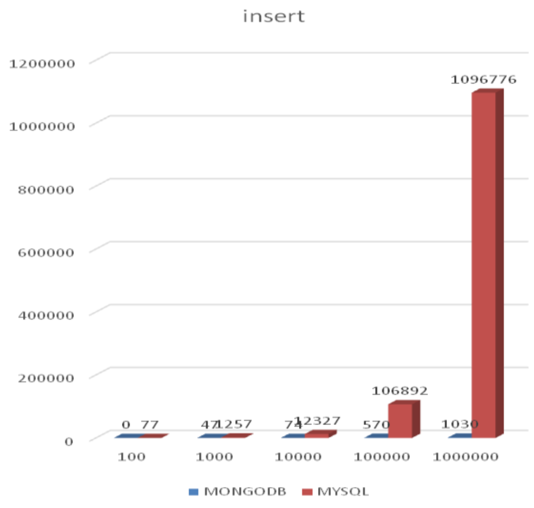

Figure 8. Insert Speed

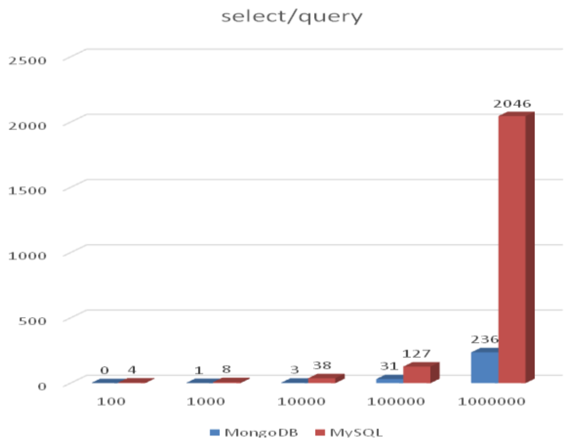

Figure 9. Query Speed

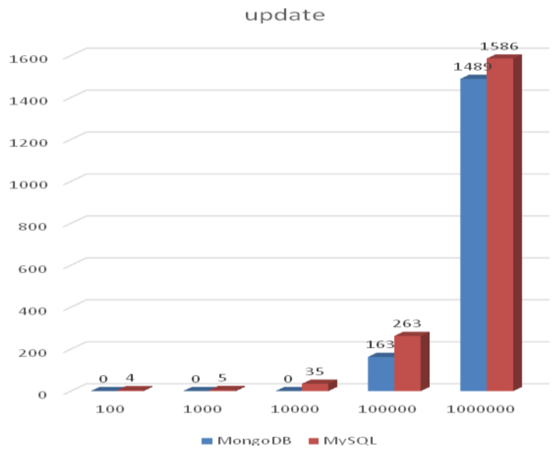

Figure 10. Update Speed

The results above are got from the average values if 10 to 50 times tests, with preprocessing of missing values and biased values. For the 1000000 of Insert and Query aspects of MySQL, the value is so big (aproxmiately 10 min) that we ignore their values in the figures.

From the Figure 8, insert speed of both databases, MySQL speeds are increased exponentially. However, MongoDB's are not. For magnitude of 100000 and 1000000, the later speed is twice of the former one. Same things happen on magnitude of 100 and 1000. The reason might because the index of nosql structure. Future work would test the insert speed on bigger magnitude and insert not only on ID, but also on other 'columns'.

For Figure 9, the query speed, value of 100000 is even smaller than 1000, which need our analysis of MongoDB documental type data. The Figure 10 is update speed, both databases are increased exponentially. So, the nosql has the high ability of selection because of its special structure. For the write aspect, the dataset we create by codes are of the same size of each tuple. However, in real world, that some data might be null, for which MongoDB would be faster for unified dataset.

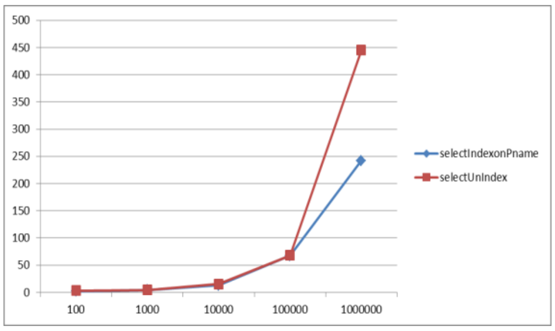

Figure11: Select With(out) Index

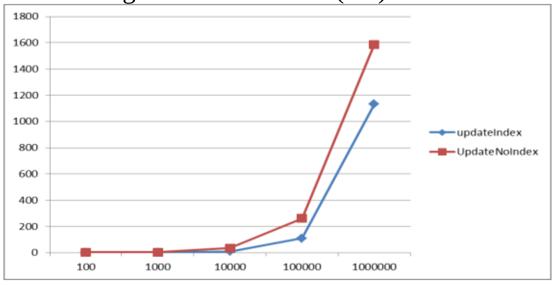

Figure12: Update With(out) Index

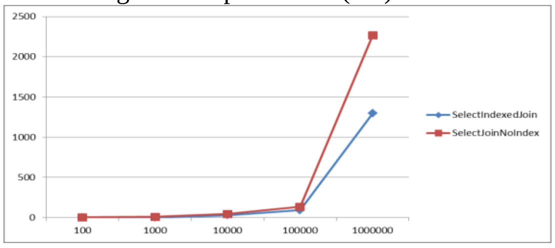

Figure13: Select from (Un)Indexed-Joined Tables

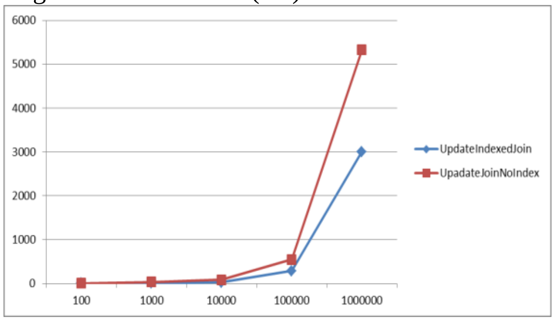

Figure14: Update from (Un)Indexed-Joined Tables

Figure 11 is select speed of MySQL default engine of indexed table and nonindexed one. Figure 12 is update speed of indexed and not indexed table. Figure 13 is the select operation from index-joined table and simple joined tables. Figure 14 is update speed of index-joined tables and simple joined tables. These tests are using Java code and the table people and city. It is obvious that in the magnititute of 100000 and 1000000 tuples, the speed of indexed queries is faster.

In conclusion, MongoDB is faster than MySQL by using Java code to perform the test. For update section, the results will highly depend on value types. Before experiment, we predicted that MongoDB will be much slower than MySQL when we update the attribute with no index. However, the result is the speeds of these two are similar, and MongoDB is a little bit faster. For query section, MongoDB is expected to have better performance because it makes all documents mapped in memory, instead of read disk in relational database. The result verified this assumption.

For Join operation, MongoDB spends around 9 ms on 10 documents, and MySQL spends only 7 ms on 100 tuples join. So, MySQL has more advantages than MongoDB on join.

VI. Extended Topic

Based on the result of comparing NoSQL and SQL databases, we can know the advantages and disadvantages of these two types of databases in different aspects. For further study, we tentatively conceive about three different directions to extend our project, and we will determine which direction to pick according to the results of our experiment.

And for further test, we can test the operation involves two or more relations. To test insert speed, we can use tuples in two or three relations for MySQL, include the smallest one and the biggest one. It is a common syntrx in MySQL in SQL language, however, for MongoDB, because of its document-type data, we might use an reference to other document instead of nesting whole file. To test update seed, we can use three different types of updates. One might be update attribute based on the primary key, one is a common attribute with no constraints, and one is an attribute having a reference key. To select operation speed, two types can be involved: the simple one is selecting data from only one object, the complex one is from nested queries, aggregation functions, etc. However, there is no JOIN operation is NoSQL, so we need other ways to figure it out

Although NoSQL is promising for future database management, it still has some issue need to be solved. For MongoDB that we tentatively used for the project, there are no database schemas and tables. It generates a primary key to uniquely indentify each document. And it attempts to remain most of the data in memory to eliminate need for retrieving from disk. Therefore, the flaw exists when data size excesses than the capacity of memory and need to query from disk. Since there is not a rigid schema definition, it is difficult to querying. For this, we can depend on the complete structure of the SQL querying to combine these two types of database to overcome this problem.

Second, since the online application developed rapidly these decades, the relational database shows poor performance to meet the requirement of such mass. Since NoSQL provides key-value storage and ensure the high performance. To take the advantages of NoSQL to perform ideally for could platform, we will combine it with distributed memory database technology to provide a mean to store huge data easily in a scalable method [1]. We desired to get the steady method that can improve the performance of querying and storage for cloud computing and node can work well when other nodes are modified.

References

[1] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation computer systems, 2009, 25(6): 599-616.

[2] Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges[J]. Journal of Internet Services and Applications, 2010, 1(1): 7-18..

[3] Abramova V, Bernardino J. NoSQL databases: MongoDB vs cassandra[C]//Proceedings of the International C* Conference on Computer Science and Software Engineering. ACM, 2013: 14-22.

[4] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB[C]//Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010: 143-154.

[5] Hecht R. NoSQL Evaluation[J]. 2011.

[6] http://docs.mongodb.org/manual/reference/program/mongotop/

Appendix

Appendix A.1 Insert Code for MongoDB

```
List datas = new ArrayList();
for (int i=0; i < 10000; i++) {
BasicDBObject bo = new BasicDBObject();
bo.put("title", "MongoDB");
bo.put("description", "database");
bo.put("likes", "100");
bo.put("url", "http://www.w3cschool.cc/mongodb/");
bo.put("by", "w3cschool.cc");
datas.add(bo);
}
collection.insert(datas);
System.out.println("Document inserted successfully");
```

Appendix A.1 Query code for MongoDB

```
DBObject myDoc = collection.findOne();
System.out.println(myDoc);

DBCursor cursor = collection.find();
int i=1;
while (cursor.hasNext()) {
System.out.println("Inserted Document: "+i);
System.out.println(cursor.next());
i++;}
```

Query code for MongoDB

Appendix A.3 Update Code for MongoDB

```
DBCursor cursor = collection.find();
while (cursor.hasNext()) {
DBObject updateDocument = cursor.next();
updateDocument.put("likes","200");
//change.add(updateDocument);
collection.update(updateDocument,updateDocument);
}
System.out.println("Document updated successfully");
//  cursor = collection.find();
int j=1;
while (cursor.hasNext()) {
System.out.println("Updated Document: "+j);
System.out.println(cursor.next());
j++;}
```

Appendix A.4 MapReduce for MongoDB

```
var mapp = function () {
            var output= {name : this.pname,cid:this.cid, city:null}
              emit(this.cid, output);
            };

var mapc = function () {

            var output= {name:null,cid:this._id , city:this.cname}
              emit(this._id, output);
            };

var reduce = function(key, values) {
   var outs = {name:null,cid:null,city:null};

   values.forEach(function(v){

             if(outs.name ==null){
                 outs.name = v.name
              }
              if(outs.cid ==null){
                 outs.cid = v.cid
              }
              if(outs.city ==null){
                 outs.city = v.city
              }

    });
   return outs;
};

result = db.people.mapReduce(mapp, reduce, {out: {reduce: 'peopleandcity'}})

result = db.city.mapReduce(mapc,reduce, {out: {reduce: 'peopleandcity'}})
```

Appendix B.1 Test Code for MySQL Insert

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class testinsert {

   public static void main(String[] args) throws Exception {
      Connection connection;
      Statement statement;

      Class.forName("com.mysql.jdbc.Driver");
      connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/yyq", "thinkpad", "");
      statement = connection.createStatement();


      statement.executeUpdate("truncate table people");
      long a=System.currentTimeMillis();

      for (int x = 0; x < 100; x++) {
```

```
    int I;

      I=x*10+1;
     String sql1 = String.format("insert into people(ID,pname,cid)"+
     "values(%d,'Mary',6)",I);
     statement.executeUpdate(sql1);

    I=x*10+2;
    String sql2 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Ann',5)",I);
    statement.executeUpdate(sql2);

    I=x*10+3;
    String sql3 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Bob',8)",I);
    statement.executeUpdate(sql3);

    I=x*10+4;
    String sql4 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Rachel',4)",I);
    statement.executeUpdate(sql4);

    I=x*10+5;
    String sql5 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Helen',3)",I);
    statement.executeUpdate(sql5);

    I=x*10+6;
    String sql6 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Steven',2)",I);
    statement.executeUpdate(sql6);

    I=x*10+7;
    String sql7 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Carrie',1)",I);
    statement.executeUpdate(sql7);

    I=x*10+8;
    String sql8 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'John',8)",I);
    statement.executeUpdate(sql8);

    I=x*10+9;
    String sql9 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Jim',4)",I);
    statement.executeUpdate(sql9);

    I=x*10+10;
    String sql10 = String.format("insert into people(ID,pname,cid)"+
    "values(%d,'Zach',7)",I);
    statement.executeUpdate(sql10);

}

long b=System.currentTimeMillis();
System.out.println("Insert takes"+ (b-a)+" ms");
```

```
        statement.close();
        connection.close();
    }
}
```

Appendix B.2 Test Code for MySQL Update

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class testupdate {

        public static void main(String[] args) throws Exception {
                Connection connection;
                Statement statement;

                Class.forName("com.mysql.jdbc.Driver");
                connection = DriverManager.getConnection(
                                "jdbc:mysql://localhost:3306/yyq", "thinkpad", "");
                statement = connection.createStatement();

                long a = System.currentTimeMillis();
                // This update is used for JOIN tables
//              String sql1 = String.format("update people join city set cname='Seattle' where pname='Bob'"+ "");
                // This update is used for single table
//              String sql1 = String.format("update people set cid=6 where pname='Ann'"+ "");
                // This update is used for single table with index on pname
//              String sql1 = String.format("update people1000000 set cid=6 where pname='Ann'"+ "");
                // This update is used for join table with index on pname
                String sql1 = String.format("update people100 join city set cid=6 where pname='Ann'"+ "");

                statement.executeUpdate(sql1);
                long b = System.currentTimeMillis();
                System.out.println("update takes" + (b - a) + " ms");

                statement.close();
                connection.close();
        }
}
```

Appendix B.1 Test Code for MySQL Select

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class testselect {

        public static void main(String[] args) throws Exception {
                Connection connection;
                Statement statement;

                Class.forName("com.mysql.jdbc.Driver");
                connection = DriverManager.getConnection(
                        "jdbc:mysql://localhost:3306/yyq", "thinkpad", "");
                statement = connection.createStatement();

                long a = System.currentTimeMillis();
                // This is used for test JOIN select with index on pname
                String sql1 = String.format("select pname,cname from people100 join city where
pname='Mary'"+ "");
                // String sql1 =String.format("select pname,cname from people1000000 join city where
cid=1"+"");

                // This is used for test select on one single table
//           String sql1 = String.format("select * from people1000000"+"");

                // This is used for test select on index on pname
                // String sql1 =
                // String.format("select * from people1000000 where pname='John'"+"");
                // String sql1 =
                // String.format("select * from people1000000 where cid=8"+"");

                statement.execute(sql1);
                long b = System.currentTimeMillis();
                System.out.println("selection takes" + (b - a) + " ms");

                statement.close();
                connection.close();
        }
}
```