Handling Big Data: Performance Comparison for MySQL, MongoDB and Redis.

Anastasiia Karpenko
EIT Master Student
University of Trento
Via Sommarive 9, Povo, Italy
(+39) 0461 28 5226
anastasiia.karpenko@studenti.unitn.it

Goksel Tolan
EIT Master Student
University of Trento
Via Sommarive 9, Povo, Italy
(+39) 0461 28 5226
goksel.tolan@studenti.unitn.it

ABSTRACT
In this paper, we test 3 various types of storage systems and measure their performance while handling operations on a large dataset of more than 7 million rows.

CCS Concepts
Information systems → Data management systems

Keywords
Big data, storage systems, performance, MySQL, Redis, MongoDB, databases, key-value store, relational database, document-oriented database.

1.INTRODUCTION
This project was done with the aim to observe the practical solution to the problem of big data retrieval and usage across various storage systems. Industry has an increasing need to store and retrieve the data efficiently. Nowadays there is an existing problem of handling big data due to the fast development of Cloud technologies and Internet of Things sphere.

Big data is characterised by volume, velocity and variety [2]. This kind of data usually is received in a constant manner, and due to its huge amount it becomes difficult to store and process it. Usually comes from different sources and, in many cases, is not well-structured and must be modified int he most useful way in order to process and analyse it further. Additionally, Veracity [4], Value and Variability are also characteristics of big data[5]. Due to large amounts of data which is processed and gathered by the companies require more sophisticated storage solutions.

Databases are used to store and make operations with data in a well-defined organized manner. Nowadays there are several types of databases which have their features and advantages over one another. From relational databases, object-oriented databases to popular NoSQL databases that have been emerging over the past years.

The performance of the databases is one of the most important factors, also with reliability taking into account that choosing the various operations and comparing them for different databases can be complicated due to different design, configurations and data access methods. This practical project takes the three different database systems and compares them by trying to find the most fast one in handling operations with big data set.

Relational databases started to be widely used in the mid 1970's and have a reputation of being very successful and reliable. MySQL is one of the most popular databases of relational type. It was chosen for this project due to its popularity, usability of user interface and SQL features.

NoSQL databases became recently popular due to their fast performance and because of their ability to handle unstructured and unrelated data. They do not have fixed schema.

MongoDB is the leading document-oriented NoSQL database that gained popularity nowadays, it was also chosen to participate in this project. Due to the structure of documents as independent units Mongo DB reads the data contiguously off disk and makes it easier to distribute data across multiple servers while preserving its locality, it leads to better performance. "Document databases generally have very powerful query engines and indexing features that make it easy and fast to execute many different optimized queries. The strength of a document database's query language is an important differentiator between these databases" [12].

Since the storing a big data and managing it is a problem of our ages, the IT world needs a quick solution. Because of the quick increase of the data on the servers and the importance of the data, data professionals want to obtain beneficial results by using this data. However this is not easy if the data is really huge to store, retrieve, read or write. In this moment new technologies such as key-value stores (for example, Redis that was chosen for this project) started to be very useful for everyone. Redis (Remote Dictionary Server) was created about 6 years ago for practical reasons. Basically, Salvatore Sanfilippo, the creator of Redis, was trying to do the impossible with an on-disk SQL database. His problem was conceptually simple: his server was receiving a stream of page views from multiple websites using a small JavaScript tracker. He needed to store the latest n page views for every site and show them in real time to users connected to a web interface, while maintaining a small history. There was no way for his SQL store to handle the load with such poor hardware. To make a long story short, the concept worked, he rewrote the first prototype using the C programming language, added a fork-based persistence feature, and Redis was born.[7, p. XV]

"Redis is an in-memory remote database that offers high performance, replication, and a unique data model to produce a platform for solving problems. Redis is a very fast non-relational, key-value store database that stores a mapping of keys to five different types of values."[7, p. 3]. Even if it is a new technology, Redis is very popular nowadays. The most common usage area of the Redis are: Caching, Expiring Data, Cookie Storage, Search Engine, Big Data operations. Nowadays, apart from small companies and users, it is possible to see that big companies such as Twitter, Pinterest**,** Github, Stack Overflow and Tumblr are using Redis as a solution [8].

2.RELATED RESEARCH

Related previously made research and benchmarking by other scientists and industry professionals were searched in the web and carefully studied for this report. For example a study of End Point company about benchmarking of top NoSQL databases (Couchbase, Cassandra, Hbase and MongoDB) was taken into account when making preliminary research for understanding about how the current project can be implemented[6]. This benchmark was carried out on Amazon Web Services EC2 instances which is an industry standard for hosting horizontally scalable databases for tests such as NoSQL databases. Though these instances are much more powerful than the machine that was used in our project and the choice of databases was concentrated only on NoSQL databases, the End Point benchmarking report was very informative and valuable in the sense of understanding the possible report structure and main points that we would like to discuss in our work.

Another work of benchmarking the performance and scalability of MySQL and MongoDB was done by Christoforos Hadjigeorgiou of University of Edinburg. The researcher used several tables in the databases and also several servers for each of them to use different sets of data and increase the performance. This project was also very interesting but very different from the current project which was implemented just on one machine only for 8 queries (not 1000s of queries like in Hadjigeorgiou's project). This work was also valuable as an existing experience of carrying out a data science project for our understanding how to proceed further with our research. [21]

A few research projects that were using Redis key-value store were found and considered as valuable information for the current work. For example, there is one research made by research group about Solving Big Data Challenges for Enterprise Application Performance Management [9]. In this research they are comparing and experiencing the performance of six modern (open-source) data stores which are Apache Cassandra, Apache HBase, Project Voldemort, Redis, VoltDB, and a MySQL Cluster. In their experiment they are using 10 million records per node and the result is with only one node, Redis has the highest throughput (more than 50K ops/sec). While throughput of Cassandra and MySQL are about half that of Redis (25K ops/sec). Workload R read intensive with 95% reads and only 5% writes. This research has shown that when it comes to read latency, both shared stores, Redis, and MySQL, have a similar pattern as well, with Redis having the best latency among all systems. In contrast to the web data stores, they have a latency that tends to decrease with the scale of the system. On the other hand when they had experiment with workload Write+Read which has 50% writes and workload Write which has a write rate of 99%, Redis does not have the same success although it has effective results.

There is also another useful research made by Matti Paksula from University of Helsinki about "Persisting Objects in Redis Key-Value Database". In this research he is comparing the Redis performance with active record pattern which is an architectural pattern found in software that stores in-memory object data in relational databases.In the test 10,000 Posts (author: String, topic: String, content: Text) with random length values were created, loaded, updated, counted and deleted. Results in the of the research show that with this special scenario ActiveRedis performs significantly faster than ActiveRecord+PostgreSQL [10].

Another journal article is about implementing database system using combination of MongoDB and Redis. According to the source "both MongoDB and Redis are scalable and Redis is used for its speed as it used key-value store whereas MongoDB is used for its flexibility, scalabilty, and efficient performance." [11] At the end the researchers could reach the result of efficiency of their hybrid system even more than Oracle Database. Which shows that there is no the best database system, but there is the best database for specific missions.

3.PROBLEM STATEMENT

3.1.Goal of project

The goal of this project is to test various types of storage systems and measure their performance while handling operations on big dataset.

3.2.Databases used

3 types of databases have been chosen: relational database (MySQL), document-oriented database (Mongo DB) and key-value store (Redis).

3.3.Dataset used

Dataset *2008.csv* was obtained from the American Statistical Association web-site, initially sourced by RITA[1]. The dataset represents the data about on-time flight performance for the year of 2008. The size of the dataset is 689,4 MB. The dataset is initially in CSV format including 29 columns and more than 7 million of rows. The choice of the dataset is due to numerous columns of information that gave the opportunity to ask interesting questions and apply several types of different queries and play with results (even if the answers on these questions about the information of the dataset is not important in this project). Data in the dataset is quite well-structured.

3.4.Queries

As for the operations, 8 various queries have been chosen that represent basic operations that can be performed with data. According to the *Yahoo! Cloud Serving Benchmark (YCSB) framework* which goal is to facilitate performance comparisons of the new generation of cloud data serving systems, there are several main types of workload operations, the majority of which were taken into account in this project [1]:

A. Update heavy workload. This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions. We have implemented 3 queries of such type (queries 3,5,6 in Table1).

B. Workload B: Read mostly workload. This workload has a 95/5 reads/write mix. Application example: photo tagging; add a tag is an update, but most operations are to read tags. We implemented 1 query of this type, slightly different from one another (query 2 in Table 1).

C. Workload C: Read only. This workload is 100% read. Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop). We implemented 1 query of this type (query 1 in Table 1), reading all the information of the dataset.

D. Workload D: Read latest workload. In this workload, new records are inserted, and the most recently inserted records are the most popular. Application example: user status updates; people want to read the latest. In this project this query was not implemented.

E. Workload E: Short ranges. In this workload, short ranges of records are queried, instead of individual records. Application example: threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id). In this project this query was not implemented.

F. Workload F: Read-modify-write. In this workload, the client will read a record, modify it, and write back the changes.

Application example: user database, where user records are read and modified by the user or to record user activity. We have implemented 3 queries of this type (queries 4,7, 8 in Table 1).

The goal of choosing these various type of queries was to test databases' performance in really different conditions.

Table 1. Queries for project implementation

Query	Type	SQL	MongoDB
1. Print all the information stored in table	Read	SELECT * FROM ontime;	db.ontime.find()
2. Print destinations column stored in the table	Read	SELECT dest FROM ontime;	db.ontime.find({ }, { dest: 1 })
3. Print the number of flights for each of the Tail Number	Read+ aggregate +write	SELECT TailNum, COUNT(FlightNum) as num FROM ontime GROUP BY TailNum;	db.ontime.aggregate([{$group : {_id : "$TailNum", num_flight : {$sum : 1}}}])
4. Update the weather delay column for the flights which were delayed on more than 30 mins.	Read +Modify +write	UPDATE ontime SET WeatherDelay = 10 WHERE DepDelay > 30;	db.ontime.update({ DepDelay: { $gt: 30 } }, { $set: { WeatherDelay: 10 } }, { multi: true })
5. To calculate various information	Read+ aggregate +write	SELECT Month, sum(AirTime) as sum_AirTime, avg(DepDelay) as avg_depdelay, avg(AirTime) as avg_airtime, max(AirTime) as max_airtime, min(AirTime) as min_airtime, count(*) as count_order FROM ontime GROUP BY(Month);	db.ontime.aggregate([{ $group: { _id: "$Month", total: { $sum: "$AirTime" }, average1: { $avg: "$DepDelay" }, average2: { $avg: "$AirTime" }, max: { $max: "$AirTime" }, min: { $min: "$AirTime" }, count: { $sum: 1 } } }, { $sort: { _id: 1 } }])

6. To calculate various information	Read+ aggregate +write	SELECT Month, sum(AirTime) as sum_AirTime, avg(DepDelay) as avg_depdelay, avg(AirTime) as avg_airtime, max(AirTime) as max_airtime, min(AirTime) as min_airtime, count(*) as count_order FROM ontime GROUP BY(Month) ORDER BY avg_depdelay;	db.ontime.aggregate([{ $group: { _id: "$Month", total: { $sum: "$AirTime" }, average1: { $avg: "$DepDelay" }, average2: { $avg: "$AirTime" }, max: { $max: "$AirTime" }, min: { $min: "$AirTime" }, count: { $sum: 1 } } }, { $sort: { _id: 1 } }, { $sort: { average1: 1 } }])
7. Add new column to the dataset table	Modify +write	ALTER TABLE ontime ADD AirTimeHours double;	db.ontime.update({ }, { $set: { AirTimeHours: "" } }, { multi: true })
8. Transform minutes into hours and add to the newly added column	Modify +write	UPDATE ontime SET AirTimeHours = AirTime / 60 WHERE 1 = 1;	Could not implement

4.SOLUTION

4.1.Set up and configurations

4.1.1.Machine configuration

For this project implantation MacBook Pro 13 was used with the following configuration:

- 2.7GHz Dual-core Intel Core i548GB 1866MHz LPDDR3 SDRAM
- 256 GB PCIe-based Flash Storage
- OS X El Capitan.

4.1.2.MySQL set up and configurations

In oder to install MySQL 5.7 the Native Package Installer, which uses the native OS X installer (DMG) was used to walk us through the installation of MySQL. MySQL Workbench was installed and used as a client interface to implement the queries on the dataset. InnoDB storage engine was used as the default one. It was proved to be the best solution for MySQL when working with large amounts of data, it has been designed for CPU efficiency and maximum performance[13]. Database ‘bigdata’, table ‘ontime’ and 4 indexes have been created in MySQL:

create index year on ontime(year);

create index date on ontime(year, month, dayofmonth);

create index origin on ontime(origin);

create index dest on ontime(dest).

4.1.3.MongoDB set up and configurations

MongoDB Community Edition was installed through the popular OS X package manager Homebrew that was relatively easy. The server Mongo Daemon ‘mongod’ was run in the bash and the client terminal ‘mongo’ was also used. For running test queries on the dataset mongo shell was used. Mongo shell is a Javascript interface to MongoDB and is a component of the MongoDB package. Queries for MongoDB were inside the Javascript scripts with were run in the mongo shell. These scripts collected the results of each operation - measure of time that running of each operation took. Database ‘bigdata’ with the collection ‘ontime’ were created in MongoDB for this project.

4.1.4.Redis set up and configurations

It was not difficult to set up Redis and run it on the computer. There is an package on the web site of Redis (redis.io) for each operating system. After downloading this package, simple installation was needed. Then Redis is ready to use. There was one redis-server.exe which is needed to be started first. Then reis-cli.exe was run. Regis desktop client was also used to check the modifications in the database visually. Since Redis does not support all the SQL commands such as GROUPBY, it was decided to use one of the programming languages to develop the necessary operations for testing the storage system. Redis supports many of the programming languages. Since Redis is written in ANSI C and works in most POSIX systems like Linux it is recommended to use Linux for deploying (in this case iOS was used). There is no official support for Windows builds, though Microsoft develops and maintains a Win-64 port of Redis. In this project we have used Java programming language to implement queries for Redis due to some previous experience of using it. To be able to use Java the related Java library of Redis, Jedi.jar was downloaded. After giving the class path of the JAR file it was be ready to use. This library supports the commands of Redis in Java programming language.

4.2.Databases

4.2.1.MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So you have nothing to pay to use it.

- It is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- It uses a standard form of the well-known SQL data language.
- It works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- It is very friendly to PHP, the most appreciated language for web development.
- It supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).[17]

MySQL also can handle big amount of data with the clustering solution that it offers: MySQL Cluster. It is the technology providing shared-nothing clustering and auto-sharding for the MySQL database management system.This solution is build upon ACID (Atomicity, Consistency, Isolation, Durability) framework that guarantees the reliability of data operations in the database.[18] In this project we do not use clusters and we use basic MySQL database and MySQL server.

4.2.2.MongoDB

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document. Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.A document is a set of key-value pairs. Documents have dynamic schema it means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data. These characteristics of MongoDB are the reasons of high popularity of this database nowadays[19]:

- Document Oriented Storage : Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication & High Availability
- Auto-Sharding
- Rich Queries
- Fast In-Place Updates.

MongoDB is suitable for big data handling. With horizontal scalability MongoDB can support thousands of nodes, petabytes of data, and hundreds of thousands of ops per second without requiring you to build custom partitioning and caching layers. Document model of MongoDB allows to store and process each type of data. The structure of a document schema can be easily adapted. MongoDB is also compatible with Hadoop and MapReduce (technologies for big data that we are not using in this project). All of these factors made us chose MongoDB as a database for current project.[20]

4.2.3.Redis

Redis is is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets. The values that can be stored in hashes are the same as what can be stored as normal strings: strings themselves, or if a value can be interpreted as a number, that value can be incremented or decremented. Redis is very effective to use when there is rapidly changing data with a foreseeable database size. Foreseeable because it should fit mostly in memory. First of all Redis is good in counting. Considering the data is stored in RAM, counting can be a simple bit operations. Secondly, Redis is also very good by caching, for this reason it is a main competitor of Memcahced. [22]

In this project as we mentioned above we have worked on 4 different kind of workload including 8 different queries. They are mostly about writing and reading, and Jedis (Will be explained with details in the next topic) the java environment of Redis has special commands for Redis in Java. For the workloads including read, Redis has HGETALL command which returns all fields and values of the hash stored at key. In the returned value, every field name is followed by its value, so the length of the reply is twice the size of the hash. [22] In the meanwhile there is hgetAll(); method in Jedis.

```
.....
Jedis jedis = new Jedis("localhost");
Map<String, String> properties = jedis.hgetAll("newdata1");
for (int i=0; i<10000; i++)
{
        properties = jedis.hgetAll("newdata"+i);
        System.out.println("Year: " + properties.get("Year"));
        System.out.println("Month: " + properties.get("Month"));
.....
```

On the other hand for the workloads including writing Redis has HMSET command which Sets the specified fields to their respective values in the hash stored at key. This command overwrites any existing fields in the hash. If key does not exist, a new key holding a hash is created. And in Jedis it has jedis.hmset(newdata"+i, Flight); and pipeline.hmset("newdata"+ i, map); methods. The difference will be explained in the "Why Redis is slower" part.

```
.....
Jedis jedis = new Jedis("localhost");
Map<String, String> Flight = new HashMap<String, String>();
value= properties.get("AirTime");
airtime= Double.valueOf(value)/60;
Flight.put("AirTimeHours", Double.toString(airtime));
jedis.hmset("newdata"+i, Flight);
.....
```

Sometimes it may be hard to use SQL commands easily on non relational databases since their working mechanism is different. As it is mentioned in book of "Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data" distributed hash table stores, unlike relational databases, usually do not support ad hoc aggregation very well. Such as GROUP BY and SUM operations, requiring client-side aggregation. [23, p. 216]

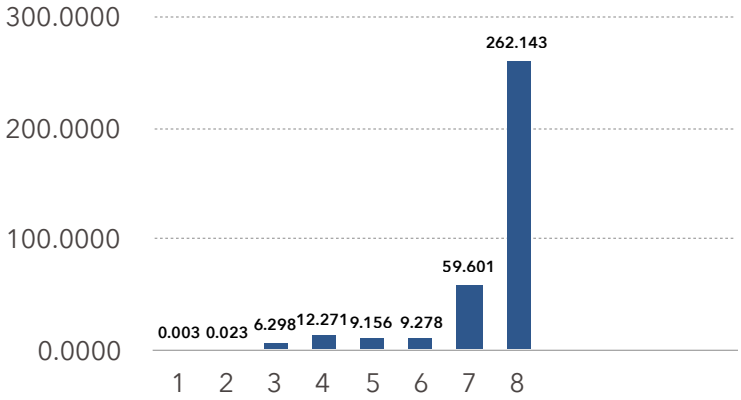
For this reason java code is used to do operations such as SUM, GROUP BY, ORDER BY. For example for GROUP BY switch case loop is used in our implementation.

```
properties = jedis.hgetAll("newdata"+i);
month= properties.get("Month");
switch (Integer.parseInt(month)) {
case 1: .....
case 2: .....
```

4.3.Results

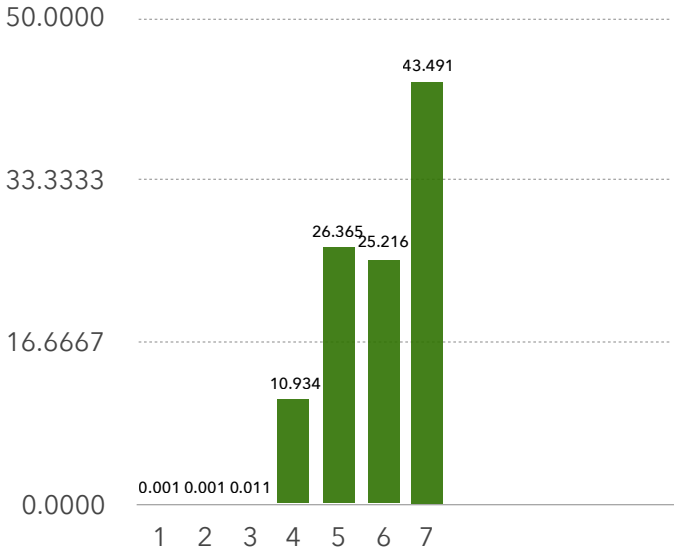
4.3.1.MySQL results

Figure 1. MySQL performance over 8 queries, sec.



4.3.2.MongoDB results

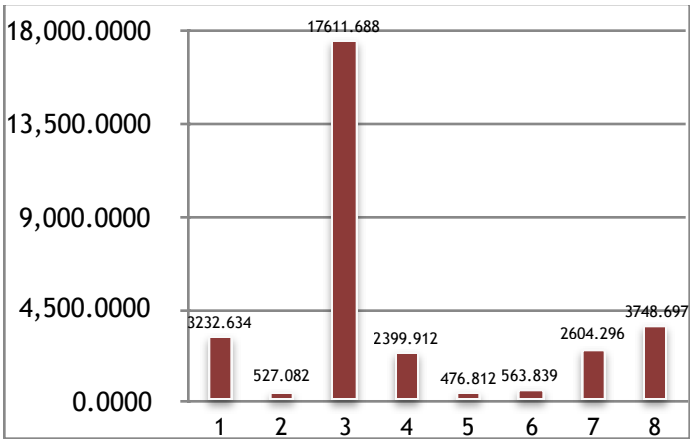
Figure 2. MongoDB performance over 7 queries, sec.



4.3.3.Redis results

According to our implementation of Redis with Java, the results are quite slow.We have used a data which has more than 7 million records to test these queries and the results are below in the diagram.

Figure 3. Redis performance over 8 queries, sec.



4.3.4.Result comparison

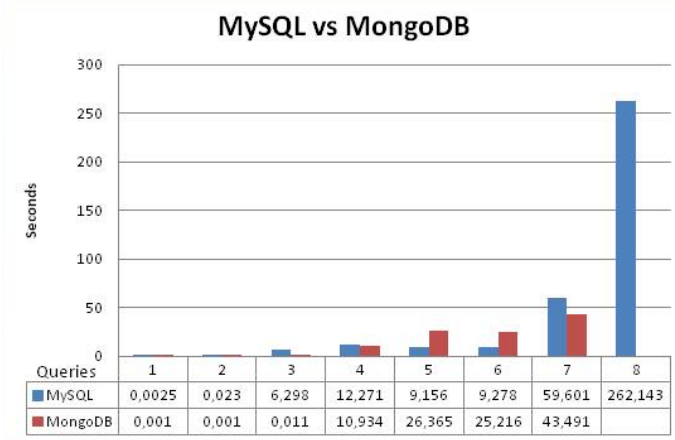
It is obvious that according to results (Table 2.) that we have taken, there is huge time difference between Redis and the others; MongoDB and MySQL. And Redis is much slower than the other 2 databases. However these are the results according to our implementation. In the available practice and related research the situation appears different. So let us look why Redis can work much slower? There are lots of reasons which may affect the performance of Redis.

Table 2. Comparative performance of 3 databases

Query	MySQL, sec	MongoDB, sec	Redis, sec
1	0.0025	0.001	3232.634
2	0.023	0.001	527.082
3	6.298	0.011	17611.688
4	12.271	10.934	2399.912
5	9.156	26.365	476.812
6	9.278	25.216	563.839
7	59.601	43.491	2604.296
8	262.143		3748.697

If we compare performance of MySQL and MongoDB (Figure 4.) that are more or less the same (except Redis), we will see that for 2 of the queries (5,6) which are quite similar and do a lot of mathematical aggregation operations MySQL is much more efficient than MongoDB. For the other majority of queries MongoDB is more efficient and fast. We can notice that for read type of queries (1,2) MongoDB shows better performance. Also queries 3 (read+aggregate+write), 4 (modify+write) and 7 (modify+write) are faster in MongoDB.

Figure 4. Comparative performance of 2 databases



Queries	1	2	3	4	5	6	7	8
MySQL	0,0025	0,023	6,298	12,271	9,156	9,278	59,601	262,143
MongoDB	0,001	0,001	0,011	10,934	26,365	25,216	43,491	

4.3.5. Why Redis is slower?

First of all, it can be beneficial to check some features of Redis to understand how it works. Inside Redis there is redis-benchmark utility. This utility simulates running commands done by N clients while sending M total queries. Unfortunately Redis commands does not benchmark Redis itself, instead it measures the network latency. To be able to really test Redis, the user should multiple the connections or it can be nice to use pipelining (it will be mentioned with details below) for aggregation commands. So network bandwidth and latency have a great effect on the performance of Redis.

The other important effect is RAM of the system. Since Redis using RAM to store the values it may be really critical to have enough RAM capacity according to data that will be used. [14]. Lets continue with pipelining. The term pipelining is a technique, when there is a Request/Response server, and this server able to process new requests without needing the client to read the old responses first. In this way it will be possible to send a lot of commands to the server before waiting for the replies and then the server can read all of them together. There is a nice table on Redis web page [15] which shows clearly with and without pipelining how the result may change with some specific commands.

Table 3. Performance with and without pipelining in Redis

Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (with pipelining)	**Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (without pipelining)**
$ ./redis-benchmark -r 1000000 -n 2000000 -t get, set, lpush, lpop -P 16 -q	$ ./redis-benchmark -r 1000000 -n 2000000 -t get, set, lpush, lpop -q
SET: 552028.75 requests per second	SET: 122556.53 requests per second
GET: 707463.75 requests per second	GET: 123601.76 requests per second
LPUSH: 767459.75 requests per second	LPUSH: 136752.14 requests per second
LPOP: 770119.38 requests per second	LPOP: 132424.03 requests per second

One last example from the web site of Redis is states that with the command of KEY, even entry laptop is able to scan a 1 million key database in approximately 40 milliseconds.[16] Which is totally different from our results. So in conclusion, we can assume that, since our code that we have used was not really compatible with Redis features, we got a slower results. By using pipeline method with Jedis, the results could be improved. On the other hand benchmarking is another important feature of Redis which is definitely be used.

4.4.Issues of the project

First of all we were too unfamiliar with Data management technologies, especially with Redis that we have never used before, and since it is also a relatively new technology in the data science (just 6 years old) it was quite hard to find necessary information online. During the project we had problems by using Redis' native query commands. The examples are either so simple that we cannot use for the project or so complicated that is hard to understand and implement to our own project. Also Redis does not support some SQL aggregation commands. This requires to program it with programming language (not SQL). Mostly in the web researchers use scripts by LUA (embeddable scripting language for Redis server side) which we did not use before. And in a limited time it would be hard to discover the features of LUA for the goal of the current project. Then we decided to use JAVA environment because we were a little familiar with Java language from the previous experience. After a long search period we could run all the queries but then we realized that we did not do it in an efficient way. Then we have discovered new methods in Java. When we ran an example of read operation *pipeline.hmset()* on a limited dataset (1 million rows limit) rather than *jedis.hmset()*, we clearly see that results are changing a lot but we did not have time to implement all the queries with pipeline and benchmark features again with the large dataset that we were using in the current project. So in the future research the performance of Redis can be improved with using pipeline.

In MySQL we tried to run INNER JOIN query just to check the performance but we spent around 17 hours waiting for this operation to be handled (using another similar big dataset of the year 2007 from RITA source to join with our project dataset). The operation running was so long that we had to interrupt the query. We had problems with stopping MySQL server after this operation due to the fact that it was refusing to stop and it seemed that it was still implementing this operation. Later when using MongoDB we decided to drop JOIN query since MongoDB does not support this kind of operation.

When using MongoDB we could not design the 8th query in mongo query language since it seemed that MongoDB could not implement this type of complicated query. We could not use $divide aggregation operator at the same time with $set operator during the update operation at the same time using the information from the existing field. We tested several times the query code but could not find out how to implement it right. We assumed that this query is not supported by MongoDB and did not implement it in this database.

5.CONCLUSION

Since big data is a nowadays' big problem for lots of companies, we wanted to search what is the problem and how can we create a solution to this. The main problem is how to store the big data and how to get logical results inside of information mass. There are lots of kind of Databases which can be used for specific solutions such as Relational Databases, Key-value stores, Graph Databases, etc. In this project we have chosen 3 of them. One is MongoDB document-oriented database, Redis Key-value store and lastly

MySQL relational database. We tried to find which one is better in which conditions and how they are reacting to specific queries that we have selected. While choosing queries we used the Yahoo! Cloud Serving Benchmark (YCSB) framework. To be able to compare the databases much better by using well accepted frameworks. Our results show that there is no absolute performance-winners among the tested databases. 2 databases (MySQL and MongoDB) were much more efficient and fast than Redis. MongoDB was faster than MyQSL for the 4 queries of 7, MySQL was more efficient in 2 queries which used mathematical aggregate operators. As for Redis , it showed extremely slow performance. We admit that the reason could be our implementation through Jedis Java library that could be affecting its performance.

For future research it could be interesting to implement these solution for more different datasets and queries and compare how the performance will change according to the change of the size of the data.

6.ACKNOWLEDGMENTS

Our thanks to professor Yannis Velegrakis for motivation and encouragement. Thanks to our colleagues Tiziano Antico, Andrea Galloni, Ana Daniel and Daniel Bruzual for support.

7.REFERENCES

1. American Statistical Association http://stat-computing.org/dataexpo/2009/the-data.html [Accessed on 25 January, 2016]
2. Berman J. Principles of Big Data: preparing, sharing, and analysing complex information. Morgan Kaufmann, 2013.
3. YCSB Core Workloads. https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads [Accessed on 25 January, 2016]
4. IBM. Big Data & Analytics HUB. On-line:http://http://www.ibmbigdatahub.com/infographic/four-vs-big-data [Accessed on 30 January, 2016].
5. Mc Nully E. Understanding Big Data: the sevenV's. On-line: http://dataconomy.com/seven-vs-bigdata/ [Accessed on 30 January, 2016].
6. Benchmarking Top NoSQL Databases. Apache Cassandra, Originally Published by End Point: April 13, 2015.Revised: May 27, 2015 http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf [Accessed on 30 January, 2016].
7. Josiah L. Carlson 2013. Redis in Action. ISBN-10: 1617290858.
8. Redis Roundup: What Companies Use Redis? http://blog.togo.io/redisphere/redis-roundup-what-companies-use-redis/ [Accessed on 30 January, 2016].
9. Rabl, Sadoghi, Jacobsen (2012) Solving Big Data Challenges for Enterprise Application Performance Management. http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf [Accessed on 30 January, 2016].
10. Matti Paksula. Persisting Objects in Redis Key-Value Database. University of Helsinki. http://www.cs.helsinki.fi/u/paksula/misc/redis.pdf [Accessed on 30 January, 2016].
11. Punia, Aggarwal. Implementing Information System Using MongoDB and Redis. International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.2, Pages : 16 - 20 (2014) *Special Issue of ICACE 2014 - Held on March 10, 2014 in Hotel Sandesh The Prince, Mysore, India.* http://www.warse.org/pdfs/2014/icace2014sp05.pdf [Accessed on 30 January, 2016].
12. Document Databases. https://www.mongodb.com/document-databases. [Accessed on 30 January, 2016].
13. MySQL documentation. InnoDB storage engine. https://dev.mysql.com/doc/refman/5.5/en/innodb-introduction.html [Accessed on 30 January, 2016].
14. Redis Benchmarks. http://redis.io/topics/benchmarks [Accessed on 30 January, 2016].
15. Redis Pipelining. http://redis.io/topics/pipelining. [Accessed on 30 January, 2016].
16. KEY command. http://redis.io/commands/KEYS [Accessed on 30 January, 2016].
17. MySQL introduction. http://www.tutorialspoint.com/mysql/mysql-introduction.htm [Accessed on 30 January, 2016].
18. Unlocking New Big Data Insights with MySQL & Hadoop. Oracle whitepaper. http://www.mysql.it/why-mysql/white-papers/mysql-and-hadoop-guide-to-big-data-integration/ [Accessed on 30 January, 2016].
19. MongoDB Advantages. http://www.tutorialspoint.com/mongodb/mongodb_advantages.htm [Accessed on 30 January, 2016].
20. MongoDB Real-Time Analytics https://www.mongodb.com/use-cases/real-time-analytics [Accessed on 30 January, 2016].
21. Christoforos Hadjigeorgiou(2013), RDBMS vs NoSQL: Performance and Scaling Comparison https://static.ph.ed.ac.uk/dissertations/hpc-msc/2012-2013/RDBMS%20vs%20NoSQL%20-%20Performance%20and%20Scaling%20Comparison.pdf [Accessed on 30 January, 2016].
22. Redis Web-site http://redis.io/commands/hgetall [Accessed on 30 January, 2016].
23. Byron Ellis, Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. https://books.google.it/books?id=DFnOAwAAQBAJ&pg=PA216&lpg=PA216&dq=does+redis+supports+groupby&source=bl&ots=CTAUteqgZh&sig=bkRbkKx3NdcaWhg1JK2WAJ80388&hl=tr&sa=X&ved=0ahUKEwilmbe_39jKAhVFYQ8KHUnxCBgQ6AEIJzAB#v=onepage&q=does%20redis%20supports%20groupby&f=false [Accessed on 30 January, 2016].