Oct 31st, 9:00 AM - 10:30 AM

Performance Evaluation of Non-Relational Data on Big Data Environments

Edmond Jajaga
University for Business and Technology, edmond.jajaga@ubt-uni.net

Edi Hasaj
University for Business and Technology - UBT, eh46604@ubt-uni.net

Performance Evaluation of Non-Relational Data on Big Data Environments

Edmond Jajaga, Assist. Prof. Dr
Department of Computer Science and Technology
University for Business and Technology
Prishtina, Kosovo
edmond.jajaga@ubt-uni.net

Edi Hasaj
Department of Computer Science and Technology
University for Business and Technology
Prishtina, Kosovo
eh46604@ubt-uni.net

***Abstract*—Big data management is a real challenge for traditional systems. The experimental evaluation is performed on measurement of performance of five different databases with an open non-relational dataset. It was structured and tested separately in each store, giving some advantages and limitations to them from a practical point of view. The results are drawn based on the throughput per number of users executed respectively. It was loaded and executed more than a million of records in each and every database. Following its semi-persistent model, Redis performed better than other databases.**

***Index terms*—Non-relational data, Big Data, Cassandra, Couchbase, HBase, MongoDB, Redis.**

I. Introduction

The relational database has revolutionized data management by structuring data since its appearance fifty years ago. In addition, and thanks to the SQL language, the relational model has remained the predominant choice for storing and retrieving structured data [1]. However, the phenomenon of large data is changing this situation [2]. Due to the advent of social networks, IoT and telephones, spatial data becomes easy to collect and has acquired the characteristics of large data such as volume, speed and variety. This is where the so called NoSQL databases come into play, that basically in structure are non-relational stores.

The growing demand for higher and faster data storage is transforming the database market. The number of applications releasing a high volume of data is increasing and data-intensive implications are increasingly being used to support decisions [3]. The non-relational database is designed to solve many of the problems encountered when dealing with specific applications such as multi-data. So the storage of textual as well as spatial data has received considerable attention over the last few years due to the accumulation of large amounts of data (very unstructured) over the years. These types of data storage systems are commonly used to provide flexibility and availability for handling Big Data. However, their architectural variety produces different scalable performance. Our main challenge remained testing each database with exactly the same data format, size, load, latency and other characteristics that would not indicate unfair results. Thus having unified workload to be run upon each database was really a challenge because each database contains different architectural style in itself.

In this paper it was used a tool called YCSB (Yahoo! Cloud Serving Benchmark)[1] to evaluate the performance of NoSQL databases when put under different workloads. This tool can also be extended to use any kind of data depending on the use case. That data is refined and evaluated and conclusions have been drawn.

The paper is organized as follows. Chapter 2 the different NoSQL systems and the methodology used to evaluate them in Big Data environments. The results are analyzed in chapter 3. Chapter 4 discusses our approach against state-of-the-art works on our hypothesis. Finally, the paper ends with conclusion notes.

II. Methodology

The term *NoSql* was used by Carlo Strozzi in 1998 who named it Strozzi NoSQL open-source database that did not expose any SQL interface, but was strill relational [11]. Basically NoSQL sacrifices strong consistency in exchange for high availability. There are different classifications of NoSQL although a more widely used way is to seperate NoSQL in the following four categories [15, 16]:

A) Key-value store: Where the data is stored in the form of key-value pairs, which is also called hash table where the value can be obtained quickly by using the key.

B) Document store: Just as the name suggests, it's designed to store documents or information that is semi-structured, and data also is stored and managed as a document style (like XML) [17] or as JSON [18], which can be used in a broad range of applications [19].

C) Column family store: The data is stored as columns and rows while the similar columns are stored together in what is called column family. This kind of storing is easier in extension and distribution and also is well suited for storing large data.

D) Graph database: This database is used to store data which is similar to the graph structure data, such as social networking or in recommendation systems.

E) Multi-model: supports more than one data model in the same database.

1 https://github.com/brianfrankcooper/YCSB Last accessed 29.02.2020

Distributed Database: Is the database in which not all storage devices are attached to a common processor. It may be stored in multiple computers, located in the same physical location, or it may be distributed over a network of interconnected computers [20].

This paper evaluates five NoSQL databases: Cassandra, Couch-Base, HBase, MongoDB and Redis.

- Cassandra Apache Cassandra is column-based family store and a distributed system where each node acts as both master and slave, where the system performs all the critical functions in a decentralized manner. The nodes communicate with each other through a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about [10].
- Couch-Base is document-oriented database, is an open-source multi-model NoSQL document-based database software [12].
- Hbase is a distributed database, column family store. It was developed out of a need to process massive amounts of data for the purposes of natural-language search [13]. The architecture of HBase is composed of a commonly used master-slave type of architecture and it uses ZooKeeper as a distributed configuration and synchronization service to maintain the server state [14].
- MongoDB is a document-oriented database where tuples or records are stored as documents in BSON (Binary JSON) syntax. Auto-sharding is a feature in MongoDB that facilitates scaling horizontally by splitting data across multiple nodes [21].
- Redis lays in the family of key-value stores. It is open-source, networked, in memory database system which is very flexible and promises very fast performance. A value in Redis can be stored as a string, a list of strings with intersections either at the head or tail of the list. Redis database is replicated using common master-slave approach.

Yahoo under YCSB project has developed a framework with a set of common workloads to evaluate the performance of databases [22]. It is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs. Often it is used to compare relative performance of NoSQL database management systems. This program has two key components. First, the client that is an expandable workload generator. Second, the core workloads that are a set of workload scenarios to be executed by the generator [22]. There are 6 core workloads built in YCSB but here were used only 3 of them as depicted in Table 1.

TABLE 1

YCSB WORKLOAD

Workload	Operations
Workload A	Update heavy, 50/50 of read/update
Workload C	Read only, 100% read operation
Workload F	Read-modify-write.

In the starting phase, data gets loaded into the database where each record has 10 columns and each column has 100 bytes of data, approximately 1kb in total per record. These are generated randomly and are uniquely identified by a key which is a combo of a string “user” and has some other random digits [15].
All the tests were executed in a personal PC with Ubuntu OS machine installed. The machine has 12GB RAM, 256GB SSD, and a quad-core CPU clocking 2.3GHZ processor. It was tested out each database equally in the same machine with the same workloads each.

- Cassandra version: 3.11[2]
- Couch-Base version used: 6.0[3]
- HBase version used: 2.2.3[4]
- MongoDB version used: 4.2.2[5]
- Redis version used: 4.0.9[6]
- YCSB version used: 0.17.0[7]
- Ubuntu version used: 18.02 LTS[8]

We highlight research directions and challenges in relation to Big Data processing and storage management system by NoSQL which is emerging impact of Big Data analysis. In NoSQL databases, transactional issue into NoSQL database and structural gap between cloud infrastructures can be more improved. Here, we describe major databases features that require further research in terms of Big Data management.

III. IMPLEMENTATION

The framework tool YCSB comes with ready to run commands and with easy to use tools. In the implementation phase, databases were properly configured as described in the previous chapter. Thus, the environment supported data storage and querying through the use of YCSB tool. This tool has all the necessary drivers that are needed to connect and exchange the data between the database and the tool itself. The drivers are developed by the community developers and are implemented in Java.

2 https://cassandra.apache.org/ Last accessed 29.02.2020
3 https://www.couchbase.com/ Last accessed 29.02.2020
4 https://hbase.apache.org/ Last accessed 29.02.2020
5 https://www.mongodb.org/ Last accessed 29.02.2020
6 https://redis.io/ Last accessed 29.02.2020
7 https://github.com/brianfrankcooper/YCSB/wiki Last accessed 29.02.2020
8 https://ubuntu.com/ Last accessed 29.02.2020

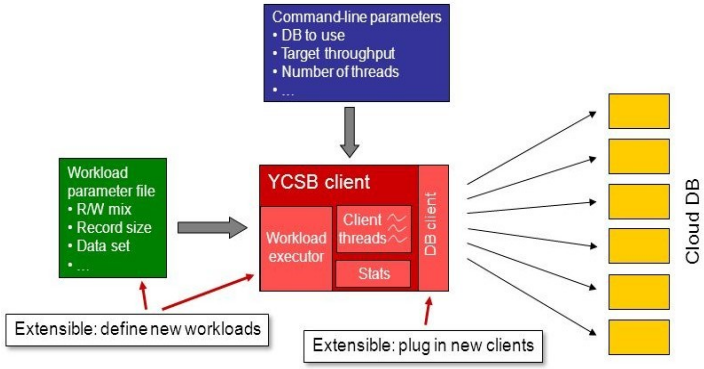

Fig.1. Benchmarking tool YCSB.

In the figure 1 we can observe the simple architecture of the YCSB tool. It is the YCSB client as the main actor that accepts inputs and executes based on the given inputs. As we will se later we pass parameters through the command line and in the parameters we describe and set the workload to be used, the throughput target, number of threads, data set, etc. The tool will execute the specified workloads uppon databases which can be in the cloud connecting through the HTTP[9] protocol or served locally as servers.

We will skip the store installation for all of the databases as these steps can be found in the corresponding site's of each database.

Since there is no unified standard for setting up the configuration between the stores and the YCSB tool, we are going to briefly describe the configuration of each database used.

Configuring the Apache Cassandra store was as easy as creating a keyspace[10] with the name called 'ycsb' and a new table in that keyspace with the name 'usertable' (as the default data to be saved are user related data).

Configuring Couch-Base store for the YCSB is not necessary because once the database node or cluster is created and pointed to the client, the tool will create the schema and store data accordingly.

For HBase it would be beter if used the pre-splitting strategy for gaining the best performance results. This means creating a new table for testing porposes otherwise all the writes will target a single region server[11].

```
hbase(main):001:0> n_splits = 200 # HBase
recommends (10 * number of regionservers)
```

```
hbase(main):002:0>   create   'usertable',
'family',  {SPLITS  =>  (1..n_splits).map  {|i|
"user#{1000+i*(9999-1000)/n_splits}"}
```

Basically these two commands will split the number of region servers to be used for data storage this for the purpose of really high value input output to the store specifically for interaction with the YCSB tool.

Configuring MongoDb is very simple, default platform installation is only needed. And the connection string parameters will be passed when we run the workloads.

Configuring Redis is the same procedure as configuring Mongo, no additional creation or specific step is needed. Except when runing the workloads.

YCSB tool is runnable only through the Terminal/Command Line Interface commands, and it can be downloaded as open source project in the internet and is free under Apache-2.0 License. The tool is available only for Linux, Windows and macOS at the time of writing, and the commands to run workloads are almost the same in both platforms. There is no configuration needed for the tool tu function properly, the only requirements needed as a dependency to run the tool are Java and Apache Maven[12]. To build the full distribution of the tool will all database bindings with Maven we run:

```
mvn clean package
```

To build a single database for example MongoDb bindings we run:

```
mvn -pl site.ycsb:mongodb-binding -am
      clean package
```

We can set the framework(tool) binaries folder to point in the operating system path variables, or we can head over to the / bin directory in the downloaded file folder.
We described some of the workloads we will be using in our experiment, but there some other customised workloads that can be used depending on the use case scenario. Tool extensibility helps in adding new customised workloads that can be created and used to execute specific queries uppon database/s for benchmarking purposes.
The procedure to test a specific workload in a specific database is firstly loading the workload data into the database and the runing the same workload type loaded into the database.
Loading and runing the workloads is as simple as runing one line command, e.g. in Cassandra we first load a workload by runing:

```
/bin/ycsb load cassandra-cql -P
      workloads/workloada

/bin/ycsb run cassandra-cql -P
      workloads/workloada
```

this means that in the bin directory of ycsb folder find the ycsb binary file and run it with the parameters being 'load' which means load the workload data into the database data, then is 'cassandra-cql' the database client to be used, we can also specify the threads to be used '-threads 4' for the maximum threads to be used concurrently when benchmarking, and -p is for inline parameter vs -P for file type parameter, and after the -P parameter we pass the workloads/workloada which basically means in the workloads directory find the workloada (Workload A) file and execute it. This command will insert default 1000 rows x 10 columns with random user data, and we can insert a specific number of rows by specifying it with additional

9 Hypertext Transfer Protocol (**HTTP**) is an application-layer protocol for transmitting hypermedia documents, such as HTML.

10 A keyspace in Cassandra is a namespace that defines data replication on nodes.

11 https://issues.apache.org/jira/browse/HBASE-4163 Last accessed 07.05.2020

12 Apache Maven is a software project management and comprehension tool. Maven can manage a project's build, reporting and documentation from a central piece of information.

parameter like '-p recordcount=100000000' number of rows. The above command will output:

```
$./bin/ycsb load basic -P workloads/workloada
-P large.dat -s > load.txt
"Loading workload... (might take a few
minutes      in some cases for large data
sets)
Starting test.
0 sec: 0 operations
10 sec: 61731 operations; 6170.6317473010795
      operations/sec
20 sec: 129054 operations; 6450.76477056883
      operations/sec
..."
```

To execute the added workload we run the following command:

```
$ ./bin/ycsb run basic -P workloads/workloada
  -p recordcount=100000000 -s > results.txt
```

This will save the results of the transaction in an output file called results.txt and can be read as a text file outputing similar text:

```
"[OVERALL],RunTime(ms), 10110
[OVERALL],Throughput(ops/sec),
      98.91196834817013
[UPDATE], Operations, 491
[UPDATE], AverageLatency(ms),
      0.054989816700611
[UPDATE], MinLatency(ms), 0
[UPDATE], MaxLatency(ms), 1
[UPDATE], 95thPercentileLatency(ms), 1
[UPDATE], 99thPercentileLatency(ms), 1
[UPDATE], Return=0, 491
[UPDATE], 0, 464
[UPDATE], 1, 27
[UPDATE], 2, 0
[UPDATE], 3, 0
[UPDATE], 4, 0
..."
```

This output indicates:

- The total execution time was 10.11 seconds
- The average throughput was 98.9 operations/sec (across all threads)
- There were 491 update operations, with associated average, min, max, 95th and 99th percentile latencies
- All 491 update operations had a return code of zero (success in this case)
- 464 operations completed in less than 1 ms, while 27 completed between 1 and 2 ms.

Similar statistics are available for the read operations.

We did load and run the same workloads as mentioned above i.e. loading and running workloads as below:

Loading the data into Couch-Base store:

```
bin/ycsb load couchbase -s -P
     workloads/workloada
```

Runing the loaded data in Couch-Base store:

```
bin/ycsb run couchbase -s -P
     workloads/workloada
```

Loading the data into HBase store:

```
bin/ycsb load hbase2 -P workloads/workloada -
cp /HBASE-HOME-DIR/conf -p table=usertable -p
     columnfamily=family
```

Runing the loaded data in HBase store:

```
bin/ycsb run hbase2 -P workloads/workloada -
cp /HBASE-HOME-DIR/conf -p table=usertable -p
     columnfamily=family
```

Loading the data into MongoDb store:

```
./bin/ycsb load mongodb-async -s -P
workloads/workloada > outputLoad.txt
```

Runing the loaded data in MongoDb store:

```
./bin/ycsb run mongodb-async -s -P workloads/
     workloada > outputRun.txt
```

Loading the data into Redis store:

```
./bin/ycsb load redis -s -P
workloads/workloada -p "redis.host=127.0.0.1"
-p "redis.port=6379" > outputLoad.txt
```

Runing the loaded data in Redis store:

```
./bin/ycsb run redis -s -P
workloads/workloada > outputRun.txt
```

Similar commands were executed for every database and for every workload naming workload A, C and F, of which resulted in many lines of logs in output files.
The data from files was extracted as mean value, and that data was compared with other stores data from the same workload. Simple data was compared statistically and that data is shown in graphs. See the experimental section below for more information.

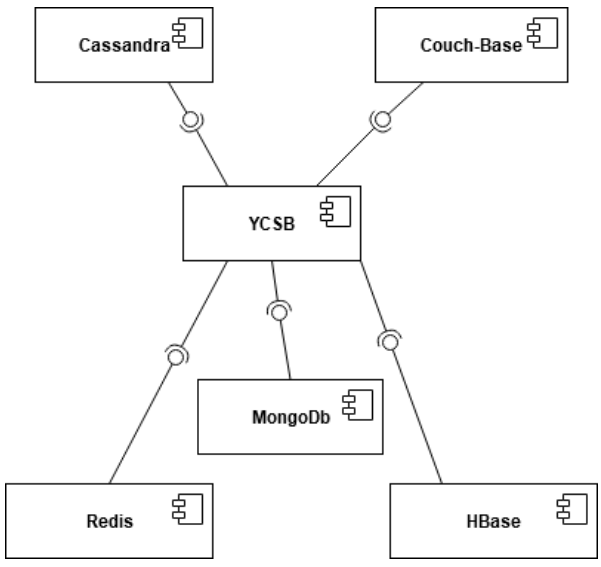

Fig. 2. Component diagram of the YCSB and other databases used.

Figure above can be conceptualised in a way of how the framework and databases interact with each other, we see a more modular way of connection between them This connection is available through drivers that behave like a communication protocol written in Java or as mentioned before they can communicate through HTTP protocols.

IV. Experimental Results

Let's observe the performance of the NoSQL systems individually. More than 30 benchmark tests were ran to find out the performance of the NoSQL systems across a variety of custom workloads. We varied the read, update, insert, and read-modify-write proportions of the workload. We also changed the number of operation counts and the record length and reported the performance of each NoSQL systems. This section will present a detailed report of the performance expectations of each database system. Here we do a comparative study of the benchmarking tests and present an experimental evaluation of the five NoSQL Systems using the various custom workloads. We compare the runtime and throughput of the NoSQL systems by changing the proportion of Insert-Read, Read-Update, Read-Read Modify-Write and Read-Modify-Write operations. The proportion of these operations were varied at 50 percent and 100 percent. The observations are reported below.

Overall Redis has the best performance, followed by MongoDB, Couch-Base, Cassandra and Hbase respectively based on the time they take to perform different operations. Redis is obviously optimized for writes and can perform them faster than reads even when the database is not heavily contended. Operations in Redis are fast enough because of its in-memory nature.

Based on the YCSB standard process, firstly, it was needed to load the data into the database, so they were inserted 1200000 records into the database and the databases stored the data automatically according to the storage configuration. Because it is difficult for us to focus on different aspects of NoSQL performance, in the loading phase only the total time of loading data into the database was taken. It was presented the outcomes and analysis of relevant experiments that were made. In this experiment were used the throughput metrics and the number of users to operate on. Throughput metric represents the operations per second that a database completes the workloads. In other words, throughput is the rate of successful execution or the successful number of operations executed per second towards the number of users (in this case), i.e. 3000 throughput or (ops/sec) means that 3000 operations (users) has been successfully executed/completed per second.

Every workload was executed separately with predefined records (10K to 1M 200K records), then data was refined and collected from the results.

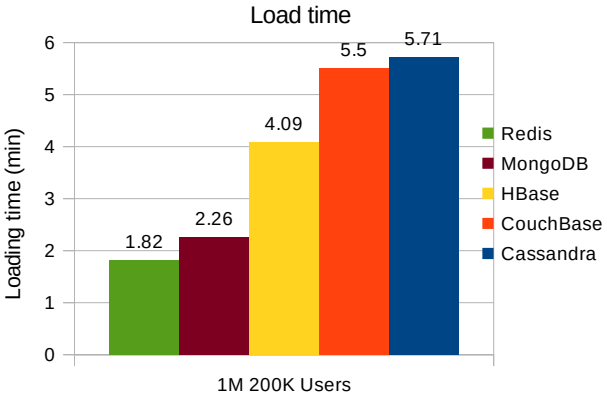

Fig.3. Loading time for 1200000 records in 5 databases.

Results were sorted out the databases from fast to slow according to the loading speed. Figure 3 shows loading of 1M and 200K of records into the databases, which results Redis getting the best performance of inserting operation among all other databases, with a loading time of 1.82 minutes which is 1.24 times faster than the second place MongoDB. This performance comes from Redis because of semi-persistent model, which means all the data is stored in memory and then asynchronously saved to disk on a regular basis for long lasting storage. The two column-based databases, HBase and Cassandra were 2.25 times and 3.14 times slower than Redis. And the other document-based database Couch-Base which is 3.02 times slower than Redis. The worst performance in making insertions in this case was Cassandra.

The test was executed on three different workloads and five chosen databases. The execution led to the performance in different scenarios seeing which database performed better. Our testing scenarios were focused only on comparing the throughput of five databases. Thus, it was compared the speed and efficiency of workload execution between different workloads.

It is important to note that Redis and MongoDb loading time is the fastest because of the way these two stores manage the data especially when inserting new data.

In all the figures, the x-axis represents the number of records and the y-axis represents throughput. All charts are grouped based on the workloads. Furthermore, each chart shows the metric measurement for each operation. For example, if a workload has both read and insert operations, then, there exists the chart showing these operations.

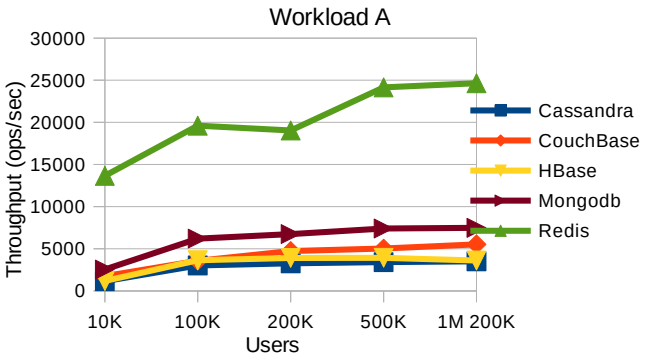

Fig.4. Executing time for 1200000 records in workload A.

Figure 4 shows the results, in seconds, obtained while executing workload A that consists of 50% reads and 50% updates, over 1,200,000 records. It was observed the execution of five tested databases executing workload A with heavy read and update (50/50), as records were increasing in size. It turned out again that Redis showed the best performance in throughput when the records were under 100K. Then, we see a slight falling after 100K, but again it increases after 200K records are read and updated. Thus, in the overall average, it was 3.33 times better than MongoDB in the second place, 4.9 times better than Couch-Base. Furthermore, HBase and Cassandra presented a similar trend in this execution phase. HBase was 6.22 times slower in execution than Redis, and Cassandra performed worst again as in the loading phase being 7.03 times slower than Redis.

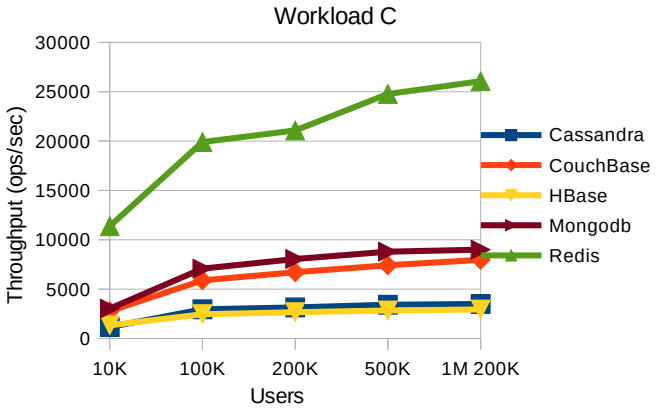

Fig.5. Executing time for 1200000 records in workload C.

Figure 5 shows the results obtained while executing workload C that consists of execution of 1,200,000 read operations over 1,200,000 records. It was observed testing the same databases, but executed with a different workload, which included 100% read on the databases as records increased. Again the Redis store obviously is better in performance compared to other stores when running workload C that corresponds to 100% reading. Redis led with an average of 20600 operations/second, on average being 2.87 times faster than MongoDB, and 3.35 faster than Couch-Base. In this workload Cassandra performed 7.28 times slower than Redis, but is performed better than HBase, which comes in the last place and was 8.45 times slower than the first place.

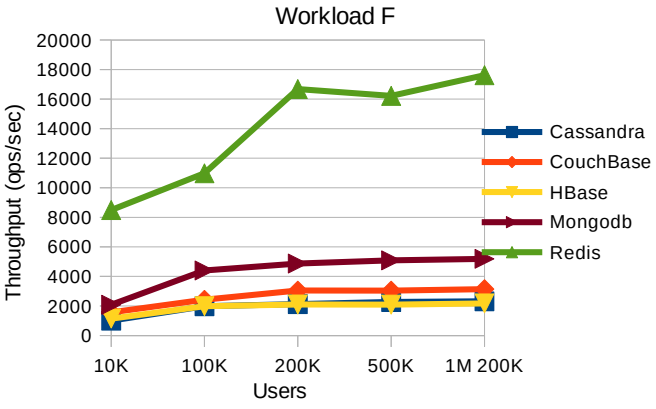

Fig.6. Executing time for 1200000 records in workload F.

Figure 6 shows the results obtained while executing workload F that consists of execution of 1,200,000 read-modify-write records. The figure describes the results about the last workload being executed against our databases. In this workload, a record was read, modified, and wrote back changes to the database. This was applied to every record (user) respectively. As in the previous scenarios, the same databases performed better, while also having a considerable gap in performance from other databases. Namely, Redis performed 3.23 times faster than MongoDB on average, 5.29 times faster than Couch-Base, 7.22 times faster than Cassandra and 7.36 times faster than HBase. It is obvious that Redis performance slightly started decreasing when reading-modifying-writing 500K records, but it went up again increasing linearly up to the 1M 200K records of data. We observe other stores having a moderate execution time with Mongo having the best performance after Redis.

Over previous paragraphs we presented results obtained over different workloads and data loading. In order to show more clearly the overall performance of these evaluated databases regardless of the type of performed operations, graph figures are generated. The figures show the total execution time, values in seconds, for each of the number of records tested in databases. These values were obtained by summing the execution times of all the same workloads for each database, and sorted in ascending order, from lowest number of records to highest.

V. Related Work

In their study Md. Razu Ahmed, et al., provided evaluation of four NoSQL databases, including document-based databases (MongoDB, Couch-Base), column-based (Cassandra, HBase), value-key (Redis) and graph based (Neoj4), where they compared, evaluated and categorized strengths, weaknesses, etc. They pointed out that their research would be useful to the business leaders in order to select appropriate NoSQL database for storage and management of Big Data [4].

Chandranil in their study presented an evaluation of the performance of four different NoSQL database systems MongoDB, Cassandra, Redis and OrientDB executing a series of custom loads by modeling different real-world scenarios. Their report can be useful to find out the performance of databases they use under different scenarios [5].

Abramova and others in their study evaluate the five most popular NoSQL databases: Cassandra, HBase, MongoDB, OrientDB, and Redis. They compare the performance databases of queries based on readings and updates, taking into account workloads. The measurement was done by the YCSB (Yahoo! Cloud Serving Benchmark) tool [6].

In their study Kamal et al, provided a qualitative comparison between three known databases of different types (Redis, Neo4j, and MongoDB) using a real-time use case of each type, translated to others. Thus it highlights the inherent differences between them, and which data structures each fit the most [7].

Tang et al in their study evaluated the performance of five NoSQL databases (Redis, MongoDB, Couchbase, Cassandra, HBase) using the measuring tool YCSB, explaining the experimental results by analyzing the model and mechanism of data of any database and provide advice to NoSQL developers and users. Their conclusions are that Redis is particularly suitable for loading and executing

workloads. Although showing the best efficiency, Redis lacks performance when facing extremely large data; Document databases. According to them the results of comparing runtime and throughput under four node cluster case are not a thorough evaluation of NoSQL databases [8].

Oussous et al in their study article have provided an accurate overview about the evolution and mechanisms of NoSQL, as well as the advantages and disadvantages of key NoSQL data models and frameworks [9].

E. H. Nassif et al., in their study about assesing NoSQL approaches for managing spatial big data, they did an experimental comparison of Accumulo vs Elasticsearch, and concluded that based on their results Accumulo is better than Elasticsearch on ingesting the data. And they concluded that the same can be applied when it comes to spatial queries like bounding box, polygons, distance, etc [23].

As it was seen in their findings Md. Razu Ahmed, et al., offered a theoretical comparison for pros and cons between different NoSQL stores for big data [4]. Instead we were focused on the aspect of how each database executes queries successfully (the throughput), and also the specified load quantity of data to test, thus getting a different perspective of how different stores react at different times.

We also looked at a more technical document from Kamal et al., where they qualified architecture level of the NoSQL stores in finding the best structure for storing big data [7]. On the other side we were more focused on the aspect of how each database executes queries successfully (the throughput), and also the specified load quantity of data to test, thus getting a different perspective of how different stores react at different times.

We took a step further from Abramova, et al., when it comes to records, where we doubled up records and took a different aspect from the results [6], so we were focused on the aspect of how each database executes queries successfully (the throughput), and also the specified load quantity of data to test, thus getting a different perspective of how different stores react at different times.

There were also differences in stores when it comes to the types of data being saved in Chakraborttii's findings, where we found that mixed types of data does not matter in what environment are being saved. The testing was done for 100K of records and they were based on the overall time of execution which in that case cannot be very accurate [5], and we were more focused on the aspect of how each database executes queries successfully (the throughput), and also the specified load quantity of data to test, thus getting a different perspective of how different stores react at different times.

Tang et al., took 100K of records to test the databases but in the execution time they only did a comparison with 1K of records which was not sufficient enough for big data [8]. We were more focused on the aspect of how each database executes queries successfully (the throughput), and also the specified load quantity of data to test, thus getting a different perspective of how different stores react at different times.

We were also overall more focused on having a practical point of view when developers have to choose between databases based on the type of data they will be saving.

VI. Conclusion and Future Work

Big data applications require that the database be optimized for the workloads they have to handle. Good performance is crucial to almost every aforementioned system. IT professionals need to do their best to ensure that the database they select is appropriate and targeted for their application use cases as fast performance is important for nearly every data-driven system. One of the ways to do this is to conduct a Benchmark test in the environment in which the database will run and under the expected data and concurrent user workloads. Benchmarks such as those contained in this paper can be useful as well in that they give database users a good idea of what the core strengths and weaknesses of the database they intend to use possesses.

This paper can be used to describe that no matter the kind of data being put in non-relational databases when there are large amounts of data the focus is on how the database will perform when making operations on the database, it may have impact if the data is only spatial, otherwise as long as data is saved in different classifications (document-based, column-based, etc) there is no significant impact in the performance.

We concluded that Redis is specifically suitable for loading and executing workloads overall in specific kind of environments and with specific kind of data. Based on different observations Redis has limitations when it faces extremely large amounts of data, this is theoretically acceptable when judged from the architecture perspective, and based on how much it is used today is used as cache or short time. This is more as a niche for Redis. If this is to be proven that Redis is not very performant at large amount of data (true in theory), then, at that level comes document-based and column-based databases, which have a good performance since they own efficiency and scalability. It is important to note that the results extracted for the throughput and the number of records under a random personal computer are not a comprehensive evaluation of NoSQL databases. Thus, in some specific applications, often it is needed to optimize configuration of NoSQL databases to the actual needs accordingly, and then compare the performance, which in turn is more commonsense and significant in NoSQL selection.

The study can be further extended for comparison of NoSQL stores in other aspects, such as operating delay, bigger datasets, the efficiency of horizontal scaling and sharding, etc, which remains as per future works.

REFERENCES

[1] D. Chamberlin, and R. Boyce: 1974. “SEQUEL: A structured English query language”. In Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control (SIGFIDET ’74). Association for Computing Machinery, New York, NY, USA, 249–264.

[2] A. De Mauro, M. Greco, and M. Grimaldi: “A formal definition of Big Data based on its essential features”. Libr. Rev. 65(3), 122–135 (2016).

[3] D.A. Pereira, W.O. Moraism and E.P. Freitas (2018). “NoSQL real-time database performance comparison”, International Journal of Parallel, Emergent and Distributed Systems, 33:2, 144-156.

[4] R. Ahmed, A. Khatun, A. Ali, and K. Sundaraj, (2018). “A literature review on NoSQL database for big data processing”. International Journal of Engineering & Technology. 7. 902-906.

[5] Ch. Chandranil, (2015). “Performance Evaluation of NoSQL Systems Using Yahoo Cloud Serving Benchmarking Tool”.

[6] V. Abramova, J. Bernardino, and P. Furtado, (2014). “Which NoSQL Database? A Performance Overview”. Open Journal of Databases (OJDB). 1.

[7] S. Kamal, H. Elazhary, and E. Hassanein, (2019). “A Qualitative Comparison of NoSQL Data Stores”. International Journal of Advanced Computer Science and Applications. 10.

[8] E. Tang, and Y. Fan, (2016). “Performance Comparison between Five NoSQL Databases”. 105-109.

[9] A. Oussous, F.Z. Benjelloun, A.A. Lahcen, and S.Belfkih, (2015). “Comparison and Classification of NoSQL Databases for Big Data”.

[10] A. Lakshman and P. Malik. “Cassandra: a decentralized structured storage system”. ACM SIGOPS Operating Systems Review, 44(2):35–40, 2010.

[11] A. Lith, J. Mattson, (2010). “Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data”. Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology. p. 70. Retrieved 12 May 2011.

[12] MC. Brown, (June 22, 2012). “Getting Started with Couchbase Server (1st edition)”. O'Reilly Media. p. 88.

[13] N. Dimiduk, A. Khurana, (28 November 2012). “HBase in Action (1st ed.)”. Manning Publications. p. 350.

[14] A. Dey, A. Fekete, R. Nambiar, and U. Rohm. “Ycsb+ t: Benchmarking web-scale transactional databases”. In Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on, pages 223-230. IEEE, 2014.

[15] V. Abramova, J. Bernardino, and P. Furtado. "Experimental evaluation of NoSQL databases." International Journal of Database Management Systems 6.3 (2014): 1.

[16] K. Orend, "Analysis and classification of NoSQL databases and evaluation of their ability to replace an object-relational Persistence Layer." Architecture (2010): 1

[17] D. Crockford. “JavaScript: The Good Parts: The Good Parts”. O'Reilly Media, Inc., 2008.

[18] M. Carro, "NoSQL Databases." arXiv preprint arXiv:1401.2101 (2014).

[19] "Document Store." https://en.wikipedia.org/wiki/Document-oriented_database Last accessed 29.02.2020

[20] “Distributed database.” https://www.its.bldrdoc.gov/fs-1037/dir-012/_1750.htm Last accessed 29.02.2020.

[21] “MongoDB Database” https://en.wikipedia.org/wiki/MongoDB Last accessed 29.02.2020.

[22] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. “Benchmarking cloud serving systems with YCSB”. In Proceedings of the 1st ACM symposium on Cloud computing, pages 143–154. ACM, 2010.

[23] E.H. Nassif, H. H. Hicham, R. Yaagoubi, H. Badir. (2020) “Assessing NoSql Approaches for Spatial Big Data Management”. In: Ezziyyani M. (eds) Advanced Intelligent Systems for Sustainable Development (AI2SD’2019). AI2SD 2019. Lecture Notes in Networks and Systems, vol 92. Springer, Cham.