Comparison between three NoSQL database architectures in a Big Data context

Abderrahim Ammour
dept. Informatique et Logiciel of Polytechnique Montréal
Montreal, Canada
abderrahim.ammour@polymtl.ca

Nour Abiad
dept. Informatique et Logiciel Polytechnique Montréal
Montreal, Canada
nour.abiad@polymtl.ca

Christophe Couturier
dept. Informatique et Logiciel of Polytechnique Montréal
Montreal, Canada
christophe.couturier@polymtl.ca

Mark Bekhet
dept. Informatique et Logiciel Polytechnique Montréal
Montreal, Canada
mark.bekhet@polymtl.ca

***Abstract*— Big data came to answer the problem which was to have a more powerful and flexible infrastructure to support a larger volume and more complex data. With the increase in digital storage capacity, the number of devices connected to the internet and the increase in the volume of processing applications, we had no choice other than to talk about Big Data. One of the solutions to help the complexity and large volume of data was the use of NoSQL databases (DB). High performance is a primary concern in the choice of data model, architecture and database design. NoSQL databases have different data models and architectures, each with its own advantages and disadvantages. Not all NoSQL databases are identical in terms of performance. In this article, we will see the difference in the performance of three popular databases such as Redis DB, MongoDB and Cassandra DB. Then, we will present the experimentation, our method and test results by using the Yahoo! Cloud Serving Benchmark (YCSB) tool. Finally, we will discuss the results in comparisons of each database. Our work aims to provide an overview of the performance and distinction of the selected NoSQL databases. What are the Relational and Non-Relational Models and their advantages? What is big data's challenge, and why are traditional database systems insufficient in handling Big Data? What makes the NoSQL system better? What are the benefits and weaknesses of the selected databases? Those questions will be answered in this paper.**

Keywords*— *Architecture, Data Models, Database, Big Data, NoSQL, Performance, Benchmarking.

I. Introduction

With the arrival of the Internet of Things (IoT) and cloud computing, the need for data stores that can store and process large amounts of data in an efficient way and cost-effective manner has increased dramatically. Big Data refers to a set of large amounts of data, sometimes more complex, that traditional data processing applications cannot process. When we talk about big data we can mention the four Vs as Volume, Velocity, Variety and Veracity [1]. Volume represents the large volume of data but also a large number of sources. Data velocity is the rate at which data is generated and processed. Variety refers to the type and nature of the data. Veracity is the quality of the captured data, that data can widely vary and can affect the accuracy of the analysis. It is also associated with the reliability of the source and the type of processing done.

A data model is a visualization that organizes data elements and normalizes their relationship with each other. The objective is to support the development of computer systems by providing the definition and format of data.

A. Relational Model

A Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values based on the concept of relation. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship [6]. The main strengths of a relational database are maintaining data integrity and accuracy, reducing data redundancy, increasing consistency, benefiting from data scalability and data flexibility, and facilitating the implementation of security methods. Above all, a relational database management system is a simpler database model to both design and implement. The main disadvantage is that it is very time consuming, the setup can become complex as the amount of data grows and relations between pieces of data become more complicated and can also have limits on the field lengths [7].

B. Non-Relational Model (NoSQL)

A non-relational model or also called "NoSQL" is a database that does not use the tabular schema of rows and columns and stores data differently than a relational table model. Instead of having a table organization, we have a document-oriented organization. This type of organization allows more flexibility and adaptability. The main types of NoSQL databases are document, key-value, wide-column, and graph. They offer flexible schemas and scale easily with large amounts of data and high user workload [6].

The key advantage of a non-relational model is that the data is not limited to a specifically structured group and this allows us to take advantage of a larger data set. Scale and speed are crucial advantages of non-relational databases. Flexible scalability, multiple data structures, and cloud-ready design are all considerable advantages when choosing a database that is less complex and easier to manage than relational databases [5].

C. Big data's challenge

Taking advantage of big data requires acquiring the data, cleaning it, representing and analyzing it, and finally

interpreting the results of the analysis [2]. In this paragraph, we discuss the challenges related to big data and how NoSQL systems help to overcome them. The most important challenges are volume and the heterogeneity of the data [2]. As mentioned in a previous paragraph, managing the growing number of data becomes difficult with a traditional database as we have to maintain the relationship between the data. However, big data is not only voluminous but also heterogeneous, which makes a relational database impractical to manage as there can be a lot of inconsistencies between the data [2]. NoSQL storages are more efficient in this case as they are scalable for high data volumes by partitioning the database across multiple servers. In fact, NoSQL databases follow the BASE properties [3]. The BASE stands for Basic-availability, Soft-state and eventual consistency[3]. This means that the data can be inconsistent, which is a trade-off for partition tolerance. In addition, NoSQL databases do not require a predefined structure for the data, unlike relational databases, but instead, store the data in a key-value model. This model makes them flexible to store heterogeneous and unstructured data [4].

D. Three data models

In this paper, we present one database for each of the three different data models used to store big data. First, let's introduce Redis DB, our choice for Key-Value data model implementation. Redis DB is an in-memory persistent database [11] which means that it stores the data in memory as well as on a disk. In fact, it is possible to configure the intervals in which Redis DB saves the state of the memory on disk. It is also flexible, as it includes four different structures: hashes, lists, sets, and sorted sets, in addition to the key-value format[11]. In addition, Redis DB scales horizontally by replication using a method called Redis Cluster[12]. Replication enables Redis DB to handle a high number of requests from different clients, as each client can communicate with a replica. However, Redis Cluster uses asynchronous replication, which decreases data consistency. [12]. Since it uses replication, Redis DB has a process called FAILOVER, which ensures fault tolerance by preventing data loss. The process delegates a replica to become a primary node, in case the original primary node is inaccessible. [14]

In the picture below, we can see the architecture of Redis DB.

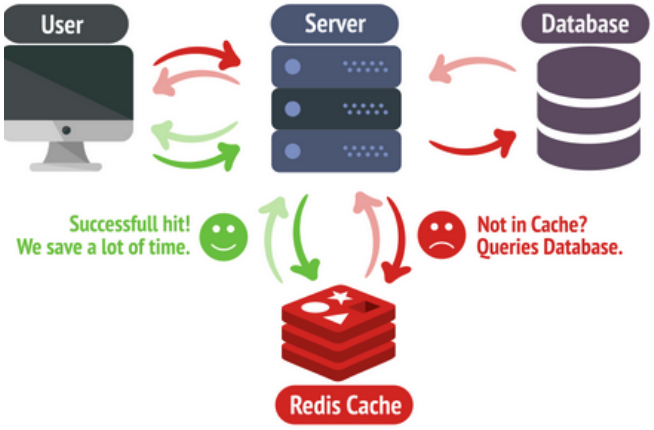

Fig. 1. Client-Server architecture using Redis DB [17]

Redis offers a great solution to satisfy the need for velocity when processing Big Data as it stores values mainly on RAM making the data highly accessible for processing.

The second data model presented in this paper is a document-based database in MongoDB. This model stores data in the form of JSON-like documents, and each document consists of field and value pairs. This makes it less flexible than Redis, but it is a useful feature when we want to store related data. In addition, MongoDB stores data in collections which are the equivalent of tables in a relational database [5]. It also provides high availability as it uses a similar failover process used in Redis DB since it also uses a primary-secondary architecture[15]. In fact, MongoDB is scalable horizontally by sharding and each shard contains a primary node and its replication. This provides data redundancy, and thus, high availability. It also provides consistency by causal consistency. This means that for an operation that depends on a previous one, it has to verify it before starting. For example, if we want to execute a read operation on data written by a previous operation, the read operation has to acknowledge the previous operation.

The figure below shows the architecture of MongoDB. We can see the shards and the primary-secondary nodes.

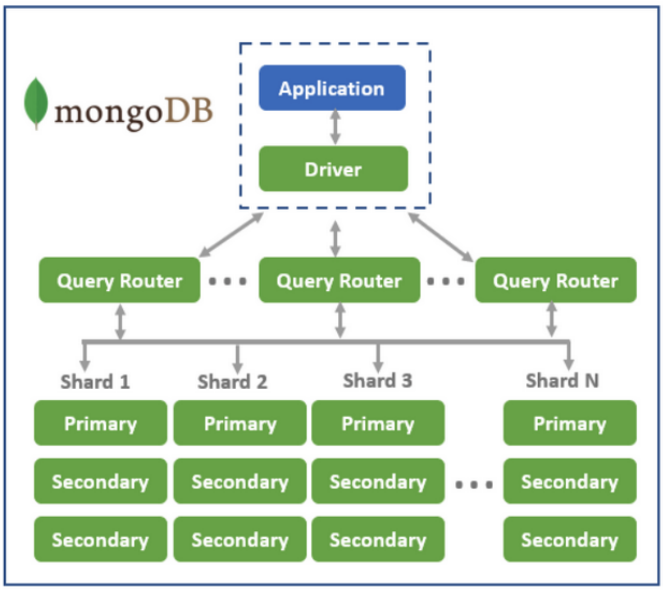

Fig. 2. Architecture of MongoDB [19]

Finally, the third data model is a column-based database. We chose to analyze Cassandra DB as the implementation. This column-based database uses a peer-to-peer distributed architecture where all the nodes are similar and connected to each other to form a database cluster. The data can be distributed in order as well as it can be distributed randomly on these nodes [6]. All the nodes are connected to each other and each node communicates information about its state to the other nodes in the cluster using a protocol called gossipping [13]. Cassandra DB is fault-tolerant since all nodes are primaries (masters), so it does not need a specific process in case of failure, but they need to be synchronized constantly. In addition to that, it supports the replication of data to multiple nodes according to the configuration of the replication factor (RF) [13]. Cassandra DB's distributed architecture enables it to scale

horizontally by increasing the number of nodes, however, also decreases the performance. In fact, when performing an operation on a node, other nodes have to acknowledge the operation to ensure data consistency, but this process of acknowledgement heavily increases the latency of the operations [13]. It is possible to configure the level of consistency (CL) by setting a smaller number of nodes that have to acknowledge the operations, however, this can affect the consistency of data [13]. In the following figure, we can see the architecture of a cluster in Cassandra.

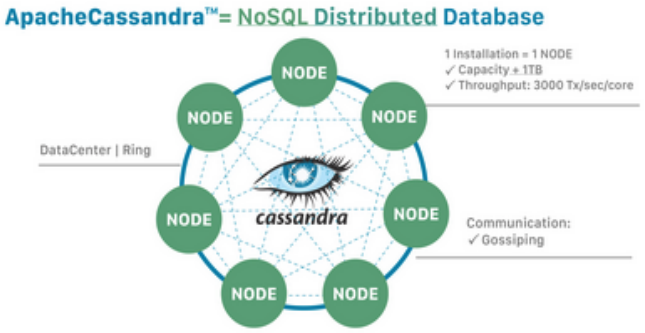

Fig. 3. Architecture of Cassandra [18]

Cassandra helps address the problem of volume that big data creates by combining the storage of the nodes to make a bigger storage pool.

II. Context

After studying the architecture of each one of the three presented databases, it is important to highlight the main differences between them in order to understand the results of the benchmarking experiment.

As mentioned in the previous section, Redis is highly performant, as it stores the data in the memory, it is also scalable, but it does not provide a high consistency. On the other hand, MongoDB provides consistency over availability whenever there is a Partition tolerance. It also scales horizontally and provides redundancy, but it is not flexible since it only supports document forms of data. Finally, Cassandra is a resilient model that is highly scalable and provides flexibility in terms of the types of data that it stores. However, it can not provide performance and consistency simultaneously.

A. Benefits and weaknesses

This table allows you to compare the benefits and weaknesses of each database.

TABLE I. Comparison between Redis DB, MongoDB and Cassandra DB

Database	*Strengths*	*Weaknesses*
Redis DB	-Fastest Read/Write [10] -Stores in RAM to go faster -Fault tolerant [11] - Flexible to store different types of data.	-Uses asynchronous replication[12] -Decreases data consistency [12]
Mongo DB	-Suitable for JSON document data -Scales horizontally for large volumes -High availability [5]	-Only supports document forms of data
Cassandra DB	-Horizontal scaling -High availability -Multiple master nodes -Large volumes -Unstructured data [13]	-Slower Read/Write -Difficulty providing consistency[13]

III. Methods

For the methodology of the tests, we made a total of 18 tests, 3 runs for each workload, on each database with the benchmarking tool to finally make an average of the results of each database. This ensures we have a final value that is quite accurate, reliable and representative of the database according to our configuration. We use for performance testing a tool called The Yahoo! Cloud Serving Benchmark (YCSB) which is open-source. It is normally used to compare the relative performance of NoSQL database management systems. To replicate the experience yourself, a script and a starting guide have been made public on our public repository [9]. The first step is the installation of the required libraries, then the creation and configuration of each virtual environment and finally the execution of the tests.

IV. Experimentation

In this study, we had to create a virtual environment and activate it to run the benchmarking tool. The goal is to deploy an instance of each of our selected implementations and test their performance using the workloads provided by the benchmark tool.

A. Configuration

For each tested database, we have a total of 4 nodes, that being 1 primary three replicas for both MongoDB and RedisDB, and 4 primaries for Cassandra DB. We ran our experiment on an AWS instance of type T2.large (2 vCPU and 8GB of RAM) and an Ubuntu OS. According to our tests, the only requirement is a minimum of 8 GB of RAM to ensure the tests completes without problems. To replicate the study, follow the readme, which is in the root directory of the repository, to install the needed dependencies and run the script. The details of the configuration of containers are defined in the script.

B. Infrastructure and deployment

We used GitHub to store our source code [9] and which contains a bash shell script to automate that benchmarking procedure. Before running the script, make sure to install the required dependencies as indicated in the README. The script installs Git, Python2 and Python3 Virtualenv to be able to run the whole script. First, the script creates a virtual environment using Virtualenv to perform the tests. Then it clones the benchmark repository and cleans up the environment.

To deploy the databases inside the virtual environment, we used docker-compose and docker technologies. The

docker images used to do this experiment are openly accessible with the docker hub.

Step by step three operations are launched one after the other in the first Redis DB, then MongoDB and finally Cassandra DB. We will create for each operation a docker container and roll the YCSB tests. For all databases when the configuration is complete a total of three tests will be run to determine an average of the output values for a total of eight workloads. First, to Redis DB, we have 3 replicates and 1 master. The open port for Redis DB is 6379. Then, for MongoDB, only one primary will be active with a total of 4 secondary. The open port for MongoDB is 27017. Finally, for Cassandra DB we will use the Simple strategy with a replication factor of 3. We will have a total of 4 nodes to replicate a network environment. At the end of the three tests, three result files will be generated automatically in the respective directories as CSV files. The docker-compose will shut down and display a finishing message.

C. *Workloads*

For the experiments, several workloads have been defined in order to test the databases. The workloads are those suggested by the benchmarking tool [8]. For the experiment, 6 different workloads have been tested each has a different configuration that has as parameters the number of operations equivalent to 1000 and the distribution of these between the following operations:

- INSERT
- CLEANUP
- UPDATE
- READ
- SCAN
- READ-MODIFY_WRITE

Each workload has a different distribution in order to test the different query contexts. The CLEANUP operation isn't included in the partition defined by the workload but each workload performs a CLEANUP operation at least once.

TABLE II. OPERATIONS DISTRIBUTION IN EACH WORKLOAD

Workload	***Operation***	***Percentage (%)***	***Characteristics***
Workload A	READ	0.5	Read/Update oriented
	UPDATE	0.5	
Workload B	READ	0.95	Read/Update oriented
	UPDATE	0.05	
Workload C	READ	1	Read oriented
Workload D	READ	0.95	Read/Insert oriented
	INSERT	0.05	
Workload E	SCAN	0.95	Scan oriented
	INSERT	0.05	
Workload F	READ-WRITE-MODIFY	0.5	Read-Write-Modify /Read oriented
	READ	0.5	

D. *Results and discussion*

For each database, we analyzed the average latency per operation as well as the throughput and runtime per workload. As mentioned before, the dataset used by the benchmarking tool is 1000 operations per workload. The time units are in microseconds which is equivalent to 1.0E-6 seconds.

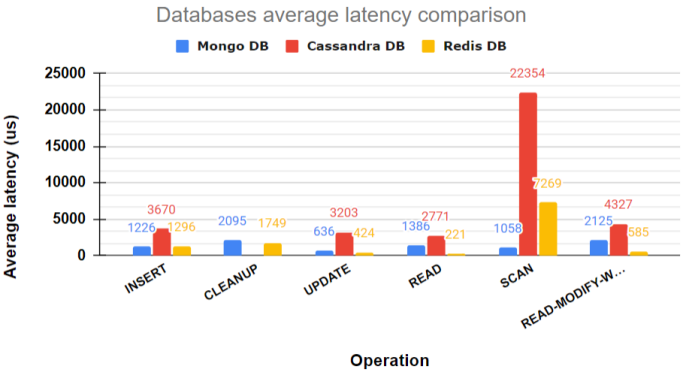

Fig. 4. Comparison graph of the latency of operations

In blue we have MongoDB, in orange Cassandra DB and Redis DB in yellow. Those graphs allow us to make several interesting observations of the average latency graph.

First in figure 4, a major gap in SCAN operation with Cassandra which is very high compared to the other. Also for the CLEANUP result of Cassandra DB, it was a huge result that made the graph not readable, we had a result of more than 2 million microseconds which is equivalent to 2.22 seconds. Then, we can see that Redis DB is the most efficient of the 6 operations except for the SCAN operation where MongoDB is better. We can also see that the second place goes to Mongo DB which is quite well-performing compared to Casandra DB. For a READ, INSERT and UPDATE operation Redis DB is a good value but MongoDB is not very far in terms of latency. For all kinds of operations, Cassandra DB is never the best performer.

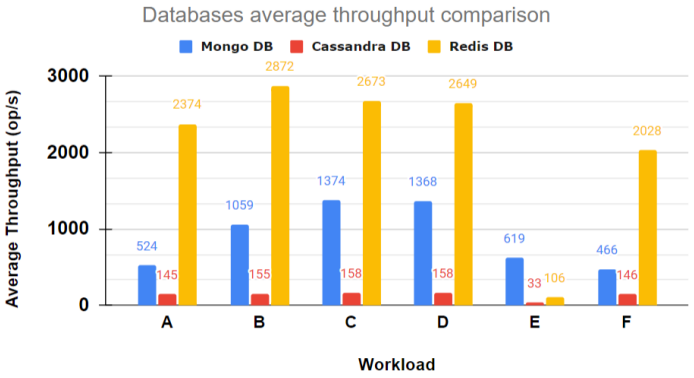

Fig. 5. Comparison graph of the throughput of workloads

In the workload graph, it is possible to see that Redis DB in yellow is the one that is the most performing in terms of Throughput which represents the operations per second. Except for the workload E which is mainly the operation SCAN. These results are coherent with the previous latency graph. For workload E, we can find that MongoDB is the most efficient followed by Redis DB. For the other workloads, the ranking is quite similar, Redis DB is the one that performs best followed by MongoDB and Cassandra DB. It is possible to observe that there is a big gap between the results of Redis DB and Cassandra DB. If we take for example the workload D, Redis DB is almost 17 times more efficient in operations per second than Cassandra DB.

TABLE III. AVERAGE LATENCY FOR THE DATABASES PER OPERATION IN MICROSECOND

Database/ Operation	*Redis DB (us)*	*MongoDB (us)*	*Cassandra DB (us)*
READ	221	1386	2771
INSERT	1296	1226	3670
UPDATE	424	636	3203
SCAN	7269	1058	22354
MODIFY-WRITE-READ	585	2125	4327
CLEANUP	1749	2095	2232661

The value of the latency represents the time taken by the request to be processed by the database. If the database has a high latency that suggests that it takes more time to process the request coming from the user, hence we can determine that it is slow.

From the extracted data, we can notice that Redis DB is the most performant DB in the READ, CLEANUP, UPDATE, and READ-WRITE-MODIFY operations. MongoDB is more performant than Redis DB in the INSERT and SCAN operations. Cassandra DB is the slowest DB in every observed operation.

TABLE IV. THROUGHPUT FOR THE DATABASES PER WORKLOAD IN OPS/S

Database/ Workload	*Redis DB (ops/s)*	*MongoDB (ops/s)*	*Cassandra DB (ops/s)*
A	2374	524	145
B	2872	1059	155
C	2673	1374	158
D	2649	1368	158
E	106	619	33
F	2028	466	146

The value of the throughput is inversely proportional with the value of average latency per operation. If a workload has only one operation, the throughput will directly show this inversely proportional relation. But since every workload has a mix of operations, it is interesting to see how many requests can the database process following the pattern of the workload. For example in workload D, we can notice that Redis DB is almost twice more performant as MongoDB. Despite MongoDB being more performant in the INSERT operation. That's due to the fact that 95% of workload D's operations are READ. However, when we look at workload E which is more SCAN oriented, we find that MongoDB is 6 times more performant than Redis DB.

For the other workloads, we notice that Redis DB is the most performant database. There are probably three main reasons as shown in Figures 6, 7 and 8.

- First, it can be associated with the fact that Redis DB stores values in memory RAM instead of static memory also called solid-state drive (SSD) or hard disk drive (HDD).
- Secondly, following an interval, Redis DB saves its state on the I/O multiplexer; it doesn't need to communicate with the disk on a regular basis like in the case of MongoDB. In addition, the single-threaded execution loop makes execution more efficient.
- Finally, the last reason is the choice of a more primary and efficient data structure.

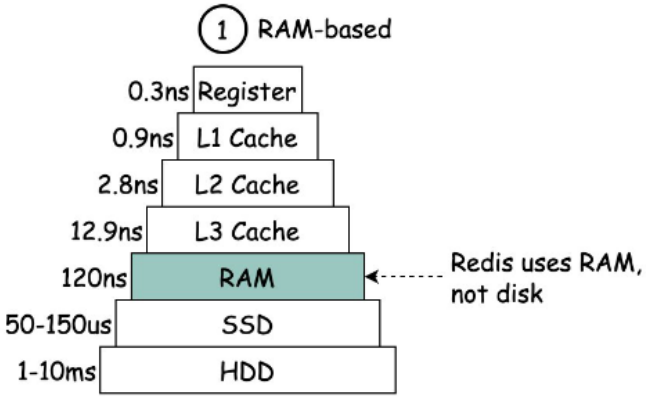

Fig. 6. Diagram of why Redis DB is so fast part 1 [18]

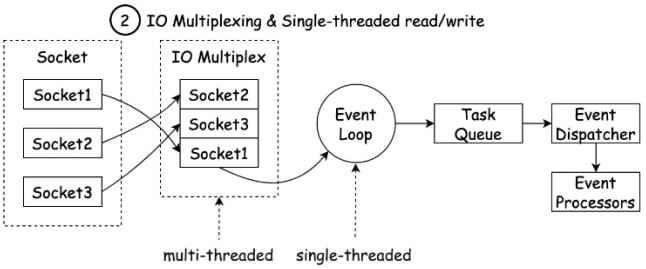

Fig. 7. Diagram of why Redis DB is so fast part 2 [18]

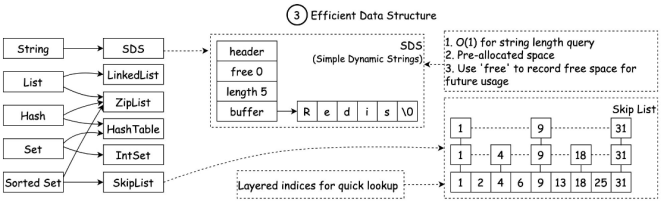

Fig. 8. Diagram of why Redis DB is so fast part 3 [18]

For example, Redis DB can communicate with the disk every 5 seconds whereas MongoDB can communicate with the disk every second. Its architecture (primary-secondary) also helps to reach this result as the replicas need to be eventually consistent with the primary meaning that data can be inconsistent for a brief amount of time.

MongoDB is the second performant database. Like MongoDB, its architecture is (primary-secondary) hence the replicas don't need to communicate with the master as frequently as they need to be eventually consistent. But unlike Redis DB, MongoDB is highly persistent, which means that it will prioritize communication with the disk, making it slower than Redis DB which uses the RAM.

Finally, Cassandra DB is the least performant database. We can attribute this to the architecture. Cassandra DB is a slaveless DB meaning that all nodes in the database act as primaries. This puts an overload on the network between nodes and causes a lot of communication between nodes as every node needs to be highly consistent with the others to respond to the next request.

V. Conclusion

In conclusion, we have studied three different data models and presented their properties and attributes. We compared these databases using a performance testing tool called Yahoo! Cloud Service Benchmark. It is important to solve the performance challenge in the Big Data context. We were therefore able to conduct performance tests in a controlled and virtual environment in order to compare three databases. The benchmarks that resulted from our tests show their performance on several query axes. These give database users a good idea of the strengths and weaknesses of each database. For benchmarking we used 6 workloads that we ran three times against each database. This gives us more accuracy on the data observed. The workloads used for benchmarking have different characteristics, for example, workload B and C are read-oriented and workload E is scan-oriented. This ensures a variety of operations and helps simulate different traffic from the users.

Our key findings were that Redis DB outperformed MongoDB and Cassandra DB in the CLEAN, UPDATE, READ and READ-MODIFY operations. MongoDB outperformed Redis DB in the SCAN and INSERT operations. Depending on the assumptions, there could be two reasons for this, one being a problem with the function that is called on the benchmarking tool that is not adapted to Redis DB and makes it take a long time or a problem with the architecture of Redis DB. This question would be interesting to investigate in future research and to detail the solution to the issue.

As mentioned in the previous section, Redis stores the data in RAM which provides better performance of the operations to retrieve and read the data. Also, the results show greater values for Cassandra's latency. This can be explained by Cassandra DB's distributed architecture, which is explained in the Context part of this paper. We can conclude that Redis DB is a fast database suitable for smaller volumes and different structures of data. MongoDB is also a fast and scalable database which is suitable for large volumes and document structured data. Cassandra DB is a highly available and scalable data model which is suitable for large volumes of data, but with a slower performance than the other mentioned models. In fact, a future subject for research could be to improve Cassandra's performance without affecting data consistency, as it is already a powerful model to store large volumes of data used a lot in the industry.

We then compared the throughputs of different workloads to see if different configurations make for a better result. For the scan-oriented workload, we found that MongoDB is the most performant database. Redis DB is faster than the other data model implementations in the five other workloads. Again, this comes to confirm the same results obtained with the average latency per operation.

This study has some limitations. First, the number of operations chosen is 1000 which is not representative of the industry but does demonstrate the differences between databases fairly. In addition, the virtual environment is run on an AWS machine which may show variations in performance from one test to another.

References

[1] danah . boyd and K. Crawford, "Six Provocations for Big Data", 04-Jan-2017. [Online]. Available: osf.io/nrjhn.

[2] Agrawal, Divyakant; Bernstein, Philip; Bertino, Elisa; Davidson, Susan; Dayal, Umeshwas; Franklin, Michael; Gehrke, Johannes; Haas, Laura; Halevy, Alon; Han, Jiawei; Jagadish, H.V.; Labrinidis, Alexandros; Madden, Sam; Papakonstantinou, Yannis; Patel, Jignesh; Ramakrishnan, Raghu; Ross, Kenneth; Shahabi, Cyrus; Suciu, Dan; Vaithyanathan, Shiv; and Widom, Jennifer, "Challenges and Opportunities with Big Data 2011-1". Cyber Center Technical Reports. Paper 1, 2011. [Online]. Available: http://docs.lib.purdue.edu/cctech/1

[3] C. Nilsson and J. Bengtson, "Storage and Transformation for Data Analysis Using NoSQL," Dissertation, 2017. [Online]. Available :http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-142004

[4] [1]J. Pokorny, "NoSQL databases: a step to database scalability in web environment," International Journal of Web Information Systems, vol. 9, no. 1. Emerald, pp. 69–82, Mar. 29, 2013. [Online]. doi: 10.1108/17440081311316398.

[5] MongoDB. (n.d.). Introduction to MongoDB, MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/introduction/

[6] R. Peterson, (2022). Relational Data Model in DBMS: Concepts, Constraints, Example. [Online]. Available: https://www.guru99.com/relational-data-model-dbms.html

[7] B. Lutkevich and J. Biscobing. (n.d.). What is a Relational Database? SearchDataManagement. [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/relational-database

[8] YCSB/workloads at master · brianfrankcooper/YCSB. (n.d.). GitHub. [Online]. Available: https://github.com/brianfrankcooper/YCSB/tree/master/workloads

[9] markbekhet. (2022). markbekhet/DB-TP3-LOG8430E. GitHub. [Online]. Available https://github.com/markbekhet/DB-TP3-LOG8430E

[10] N. B. Seghier and O. Kazar, “Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool,” (2021). 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI). [Online]. doi: 10.1109/icrami52622.2021.9585956.

[11] Redis. (n.d.). Introduction to Redis. [Online]. Available: https://redis.io/docs/about/

[12] Redis. (n.d.). Scaling with Redis Cluster. [Online]. Available: https://redis.io/docs/management/scaling/

[13] Apache Cassandra. (n.d.) What is Apache Cassandra?, Cassandra Basics. [Online]. Available: https://cassandra.apache.org/_/cassandra-basics.html

[14] Redis. (n.d.). FAILOVER. [Online]. Available: https://redis.io/commands/failover/

[15] MongoDB. (n.d.). Replica Set High Availability, MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/v6.0/core/replica-set-high-availability/

[16] MongoDB. (n.d.). Read Isolation, Consistency, and Recency, MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/core/read-isolation-consistency-recency/

[17] B. Borisov. (2022). Redis as Cache: How it Works and Why to Use it. Linuxiac. [Online]. Available: https://linuxiac.com/redis-as-cache/

[18] Apache Cassandra. (n.d.). Apache Cassandra Documentation. [Online]. Available: https://cassandra.apache.org/_/cassandra-basics.html

[19] A. Xu. (2022). Why is redis so fast? [Online]. Available: https://blog.bytebytego.com/p/why-is-redis-so-fast