INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

A Study of Performance and Comparison of NoSQL Databases: MongoDB, Cassandra, and Redis Using YCSB

Mohammad Abu Kausar[1]*, **Mohammad Nasar**[2], **Arockiasamy Soosaimanickam**[1]

1 Department of Information Systems, University of Nizwa, Sultanate of Oman
2 Department of Computing & Informatics, Mazoon College, Sultanate of Oman

OPEN ACCESS

Received: 28-06-2022
Accepted: 23-07-2022
Published: 13.08.2022

Citation: Abu Kausar M, Nasar M, Soosaimanickam A (2022) A Study of Performance and Comparison of NoSQL Databases: MongoDB, Cassandra, and Redis Using YCSB. Indian Journal of Science and Technology 15(31): 1532-1540. https://doi.org/10.17485/IJST/v15i31.1352

* **Corresponding author**.

kausar@unizwa.edu.om

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Background/Objectives: Relational databases are a commonly utilized technology that allows for the storage, administration, and retrieval of various data schemas. However, for certain big databases, executing queries can become a time-consuming and inefficient procedure. Furthermore, storing enormous volumes of data necessitates servers with greater capacity and scalability. Relational databases have limits when it comes to dealing with scalability for big amounts of data. On the other hand, non-relational database systems, often known as NoSQL, were created to better fulfill the demands of key-value storing of enormous volumes of records. However, there are several NoSQL options, and the majority have not yet been extensively compared. The goal of this research is to examine different NoSQL databases and evaluate their performance in terms of typical data storage and retrieval. **Methods:** In this study, we use the YCSB tool to measure the performance of three NoSQL databases: MongoDB, Cassandra, and Redis. We test six different workloads with 100000, 250000, 500000, 750000, and 1000000 operations. Our test was designed with five different operations, i.e., 100000, 250000, 500000, 750000, and 1000000, with six different workloads to see which database is most suitable for applications which use a large amount of data to process. **Findings:** MongoDB is a superior performing NoSQL database among Cassandra and Redis. The numerous optimizations used by the designers of NoSQL solutions to improve performance, such as good cache memory operation, have a direct impact on the execution time. In all workloads except workload D, MongoDB has significantly reduced latency across all operation counts. **Novelty:** We also measure the average latency of different workload scenarios that include a mix of read, write, and update activities.

Keywords: NoSQL; YCSB; big data; cloud computing; MongoDB; Cassandra; Redis

1 Introduction

The following are the primary reasons why ”data storage mechanism” is regarded as the core of corporate software systems: (1) The most important part of software is what controls how quickly an application responds to a request, and (2) data loss is frequently regarded as undesirable because it disrupts critical business operations. Relational database management systems (RDMS) were the sole option up to the advent of NoSQL (Not-Only SQL) databases. However, as the amount of data kept grows, relational database management system constraints, such as scalability and storage, as well as query efficiency loss due to huge volumes of data, become more complex, making the storage and maintenance of bigger databases more challenging [(1)].

NoSQL databases, meaning ”Not only SQL”, have become known since 2009, to meet new performance needs when processing large volumes of data. NoSQL does not replace relational databases; rather, it complements or replaces the functionality of relational databases to provide more interesting solutions in specific situations. The term ”NoSQL” is made up of two words: ”no” and ”SQL” [(2)]. The name may appear to be in opposition to SQL databases, leading some to believe that it indicates the end of the SQL language and, as a result, should be avoided. In truth, ”No” is an acronym for ”Not only,” emphasizing that there is more to relational databases and SQL than that. The movement's primary goal is not to replace relational databases like SQL but to provide alternatives or expand the capabilities of current models to manage vast amounts of dispersed data. The NoSQL movement brings together many data management solutions that are no longer based on the classic architecture of relational databases and are distinguished from the SQL model by a non-relational data representation logic [(3)]. The logical unit is no longer the table, and the data is generally not manipulated with SQL. Originally, it was used to modify massive databases for websites with big audiences, like Amazon.com, Google, Facebook, or eBay. These new data storage structures, in contrast to traditional DBMSs and data warehouses, are dispersed over many servers and created to accommodate the hundreds or millions of users that make changes as well as reads [(4)]. They rely on parallel file systems to boost efficiency and deal with resource constraints by multiplying hardware to enable parallelization of information storage and access. NoSQL databases tend to use low-end servers at lower costs to equip the ”clusters”. Servers for NoSQL databases are generally cheap and of average quality, unlike those used by relational databases. In addition, the vast majority of NoSQL solutions are open-source, which reflects, on the one hand, a significant saving on the price of licenses. The volume of data and its diversity are so important today that the systems of data storage and management have emerged. This ranges from file management systems to highly structured systems [(5)] (such as relational systems), passing through a variety of semi-structured data storage systems. This diversity has also led to the wide use of data exchange formats. Regarding the data, some is semantically unstructured, like sound. Semi-structured data has a certain form of structuring which is generally not explicit and which does not impose strict typing.

The CAP theorem is an acronym for ”coherence”, ”availability”, and ”partition tolerance”, also known as Brewer's theorem. This theorem, formulated by Eric Brewer in 2000 and demonstrated by Seth Gilbert and Nancy Lych in 2002, is a conjecture that states that it is impossible, on a computer system with ribbed computing, to guarantee the following three constraints at the same time [(6)]:

Consistency: At the same moment, all nodes (servers) in the system see the same data.

Availability: Ensure that any request gets a response, even if it hasn't been changed.

Tolerance for partitions: Except in the event of a broad network outage, the system must be able to respond correctly to all requests in all conditions. When splitting a network into subnets, each subnet must be able to function independently.

MongoDB is a document-oriented database that is open-source and provided under the AGPL license. (Free license), ensuring excellent performance, availability, and scalability on demand. The MongoDB database has been created in C++ by the 10gen firm since 2007, when it was working on a broadly distributed data cloud computing system akin to Google's App Engine. The initial version was released in 2009, but version 1.4 was only declared commercially acceptable in 2010 [(7)].

Cassandra is a large-scale data management system originally designed in 2007 by engineers from Facebook to address issues related to the storage and use of large volumes of data. In 2008, they tried to democratize it by providing a stable, documented version, available on GoogleCode. However, Cassandra did not receive a particularly enthusiastic reception [(8)]. The engineers from Facebook therefore decided in 2009 to bring Cassandra to the Apache Incubator. In 2010, Cassandra was promoted to the top-level Apache Project.

Redis, which stands for Remote Dictionary Server, is a BSD-licensed key/value type NoSQL database that was created in C [(9)]. It belongs to the NoSQL movement and strives to deliver the highest performance. Redis supports a variety of basic data types, including lists, associative arrays, sets, and ordered sets. With the release of ”sentinelfailover,” which handles monitoring, alerting, and automatically switching instances in the case of issues, Redis quietly continues its course after seeing significant growth in 2010.

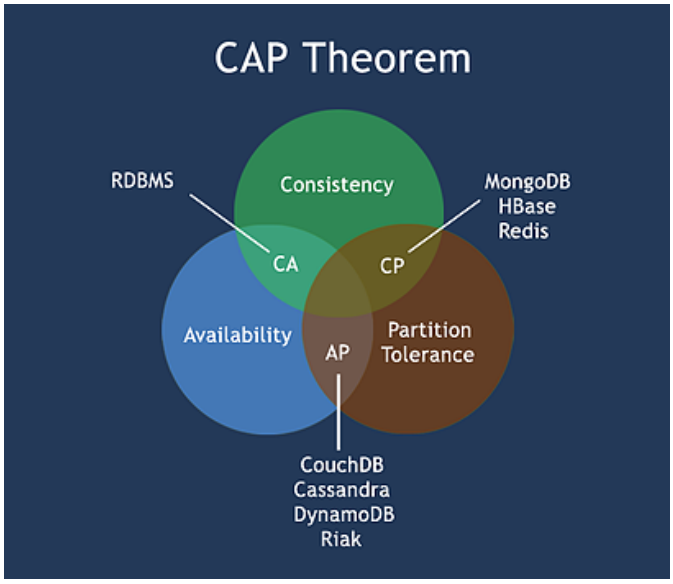

Fig 1. The CAP Theorem [5]

Related Work

Author [10] contrasts SQL and NoSQL databases, detailing their history and applications. Author [11] gives a thorough overview of NoSQL databases, as well as a comparison based on the following characteristics: (1) scalability, (2) transactional integrity and consistency, (3) data modeling, (4) query functionality, and (5) access and interface availability [11]. Hecht and Jablonski give a survey on NoSQL databases based on use cases [12]. They evaluate NoSQL databases on the basis of their data models, query capabilities, concurrency controls, partitioning, and replication capabilities.

Yahoo! Cloud Serving Benchmark is used by [1] to evaluate and compare the performance of NoSQL databases. By altering the read, update, and insert operations ratios, they produced 600,000 records at random and utilized them with different workloads. The databases utilized in the experiment were Redis, Cassandra, HBase, MongoDB, and OrientDB8. They assert that Redis, an in-memory database, performs at the highest level overall. Additionally, they assert that because Cassandra and HBase are optimized for update operations, they perform well.

The behavior of two of the most popular document-based NoSQL databases, MongoDB and document-based MySQL, was examined in this research [13] in terms of the complexity and effectiveness of CRUD operations, particularly in query operations. The MongoDB and MySQL databases were used to create a case-study application that aims to model and simplify the operations of service providers who require a lot of data in order to conduct this research. Additionally, these tests are run on data sets with 1000–100,000 records. The data was produced at random using an iterative process and only allowing particular fields. The author highlighted the YCSB tool as a potential future work.

In Article [14], the author compared CouchDB and MongoDB, two document-based NoSQL databases. The author conducted major parametric comparisons between these two datasets. The replication technique and platform support are two key distinctions. The author's comparisons make it apparent that MongoDB is a superior option than CouchDB if an application needs greater efficiency and speed. If the database is expanding quickly, CouchDB is less suitable than MongoDB. The author did not apply any data-based compression.

In this research, two widely used NoSQL database management systems, MongoDB and Apache Cassandra, are compared and contrasted in the author's [15] performance benchmarking research. The performance metric that has been looked at is total runtime. The measured findings show that Apache Cassandra outperforms MongoDB when the number of operations and level of parallelism are high.

Cornelia A. Gyorodi, Diana V. Dumse-Burescu, Doina R. Zmaranda, and Robert S. Gyorodi [16] conducted a comparison of two relatively current NoSQL databases, MongoDB and document-based MySQL, taking into account their impact on application performance when performing CRUD requests. Finally, regardless of the complexity of the queries or the amount of data, both databases are suited for Big Data applications requiring a vast volume of data as well as very complicated databases, with very quick response times.

In the above lecture review, the authors test the performance based on specific records like 1000 and 10000. Our attempt to address this in this article is to do comparative analysis of the performance of three commonly used databases: MongoDB, Cassandra, and Redis. Existing tests are based on overall throughput of 100000, 250000, 500000, 750000, and 1000000 operations. We will also measure the average latency of different workload scenarios that include a mix of read, write, and update activities. The authors employ the widely used Yahoo Cloud Serving Benchmarking Tool (YCSB), which is a performance measurement tool for NoSQL databases. This helps a user to better understand database performance and choose which system is best for a given workload.

The rest of this research study is arranged in the following manner. The following section covers related work. The methodology is described in Section 2. The results and discussion for the tests are presented in Section 3, followed by the experimental evaluation. Finally, we offer our conclusions and recommendations in section 4.

2 Methodology

A complex tool provided by Yahoo is called YCSB (Yahoo! Cloud Serving Benchmark). It's a brand-new open-source benchmarking approach that lets users make their own packages by adding additional workload parameters or, if necessary, writing Java code. In a Yahoo! research that included benchmark data for four commonly used systems, it was discovered that: Apache HBase, Apache Cassandra, YahooPNUTS,!, and a sharded MySQL version are the best in terms of performance and elasticity. In [17] , the author argues for scalability benchmarks and suggests that a good place to start is with the YCSB. He thinks that YCSB is the top NoSQL benchmark right now. The benchmark analyzes scalability, which includes how the benchmarked system scales when additional servers are added and how quickly it adapts to new servers, as well as basic performance, such as latency characteristics as server load increases [17]. The majority of NoSQL systems are supported by this tool, which is cross-platform and easy to customize for these solutions. It has been used in several research studies [17–21] and is widely used to compare the performance of NoSQL database management systems in the cloud. A data generator and a group of performance tests for adding, updating, and deleting items make up YCSB. Utilizing various workloads, we may specify the amount of records to be loaded, the quantity of operations to be performed, and the split between reading and writing in each test (Example: Workload A: 50 percent reads, 50 percent updates). These workloads can be changed and tailored depending on the type of test results anticipated. How YCSB interacts with a database is shown in the picture below:

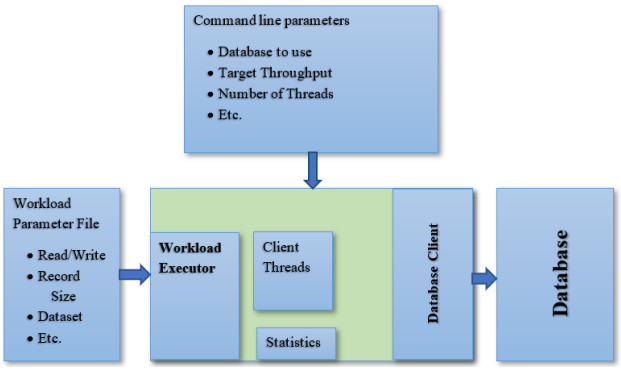

Fig 2. YCSB Architecture

Test Configuration Details

Table 1 contains a list of all the hardware and software components used in our investigation.

2.2 Performance Test Plan

We'll use the three databases covered in this paper for our testing: MongoDB, Cassandra, and Redis. Using the Yahoo! Cloud Serving Benchmark framework, we'll put each of the three databases to the test. The YCSB tool is made up of two parts: a ycsb-client that creates the workload and defined workloads, which are the scenarios that the client will run.

We continued with the installation and configuration of YCSB 0.17.0 after downloading and installing MongoDB 4.4, Cassandra 4.0.3, and Redis 6.2.6, but first, we needed to install Java, Maven, and Git on our machine. Each test began with a

Table 1. Details of the test setting

Hardware	Software
Processor : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz RAM : 12.00 GB HDD: 1 TB	Windows 10 , PC 64-bit YCSB 0.17.0 MongoDB 4.4. Cassandra 4.0.3 Redis 6.2.6

blank database. We began by creating six workloads in the YCSB tool. Workloads (detailed below) are run on the three databases after the data has been loaded. Checks on the database's health are performed between each task.

The general objectives of the tests were to:

1. Choose workloads that are representative of today's current apps.
2. Use data amounts similar to those found in "big data" datasets.
3. Vary the read/write workload amounts to compare the two solutions' performance.
4. For a more comparability of results and a clearer comparison, keep the same titles of workloads with the same rates as in [(1)].

Workloads are a set of scenarios that include a mix of read, write, and update activities. The following are the workloads that we used in our testing:

- Workload A has a 50% reads and 50% updates ratio;
- Workload B has a 95% reads and 5% updates ratio.
- Workload C, all readings;
- Workload D, consisting of 95% reads and 5% inserts;
- Workload E, consisting of 95% scanning and 5% inserting;
- Workload F, which is composed of 50% reads and 50% read-modify-write;

100000, 250000, 500000, 750000, and 1000000 operations were chosen for testing. We're also selecting entries for Uniform using the default distribution.

3 Results and Discussion

For all databases and workloads across all operation counts, all tests have been executed successfully, confirming no insert, read, or update failure.

3.1 Workload A (50% Read and 50% Update)

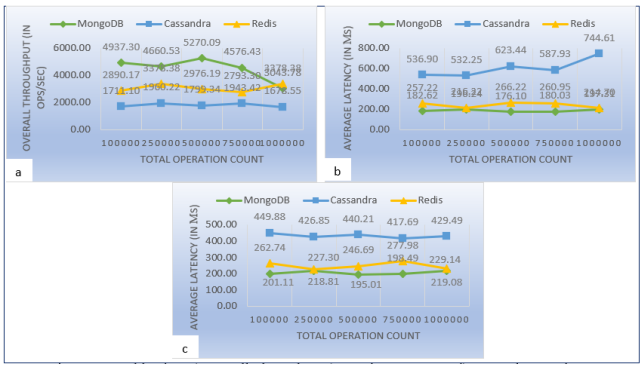

Fig 3. Workload A: a) Overall Throughput (operations per second) vs Total Operations. b) ReadAverage Latency (in μs) vs Total Operations. c) UpdateAverage Latency (in μs) vs Total Operations

The total performance of MongoDB and Cassandra databases drops precipitously as the number of operations rises, but Redis' overall throughput rises (Figure 3 a). When evaluated for 100,000 operation counts, MongoDB had a throughput that was 2.9 times higher and Redis had a throughput that was 1.68 times higher than Cassandra. However, once the operation counts

are increased to 1,000,000, this drops to just 1.81 times for MongoDB and increases to 2 times for Redis. As a result, even while MongoDB offers superior throughput, there is a greater reduction in overall throughput in its case.

The read average latency for the Cassandra database is steadily rising (Figure 3b). Compared to MongoDB and Redis, Cassandra has greater read latency. When compared to Cassandra, MongoDB has a read latency of 34 percent, and Redis has a read latency of 47.9 percent. At 100,000 total operations, or around 50,000 read operations, MongoDB's latency progressively increases to 26.5 percent, while Redis's latency increases to 28.8 percent.

Compared to the overall throughput and read latency graphs, the update latency has a somewhat distinct curve. From 100,000 to 1,000,000 operations, MongoDB and Redis have a steady rise in latency, but Cassandra has fairly stable latency. However, it is clear from Figure 3c, that MongoDB still has very little latency while performing update operations.

3.2 Workload B (95% Read and 5% Update)

When compared to Redis and Cassandra, MongoDB offers many more operations per second in workload B. Redis and Cassandra ensure stable performance as the number of operations increases. However, MongoDB's performance may vary. When 500,000 and 750,000 procedures are performed, it drastically decreases. But compared to Redis and Cassandra, MongoDB continues to retain a substantially higher throughput (Figure 4a).

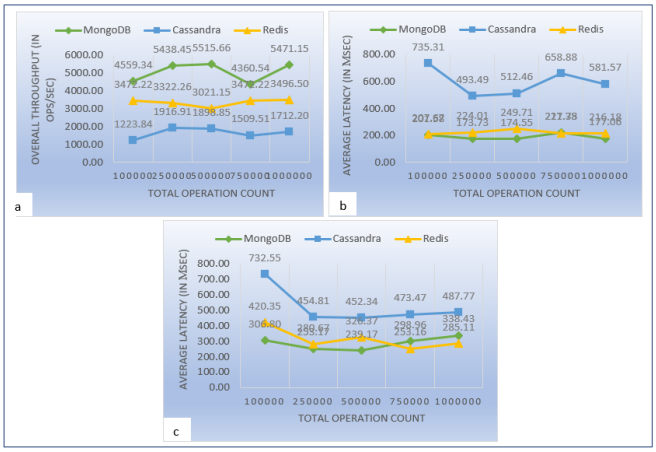

Fig 4. Workload B: a) Overall Throughput (operations per second) vs Total Operations. b) ReadAverage Latency (in μs) vs Total Operations. c) UpdateAverage Latency (in μs) vs Total Operations

As might be predicted with a 95 percent read operation mix, MongoDB has significantly lower read operation latency than Redis and Cassandra. Due to the high volume of active operations, there is only a little increase in MongoDB's latency when compared to Redis and Cassandra (Figure 4b).

Once more, we see that Cassandra still has a larger update latency than Redis and MongoDB in workload B. With an increase in operation counts, Cassandra's latency increases marginally, while MongoDB's latency is also rising (Figure 4c).

Workload C (100% Read)

In workload C, MongoDB has a substantially higher total operations per second than Redis and Cassandra. Cassandra's performance falls as the number of operations rises, but MongoDB's throughput begins to marginally decline at 1,000,000 operations. With Redis, throughput is almost constant throughout the process. But compared to Redis and Cassandra, MongoDB still has a considerably greater throughput (Figure 5a).

With a 100% read operation, the latency in read operations is lower for MongoDB as compared to Redis and Cassandra. From the above figure, MongoDB and Redis latency are almost consistent across the operation count. Overall, MongoDB's latency is lower than the other two databases (Figure 5b).

3.4 Workload D (5% Insert and 95% Read)

Compared to MongoDB and Cassandra, Redis has a substantially higher total operations per second in workload D. Redis improves throughput whereas Cassandra reduces it as the number of operations rises, although MongoDB experiences a tiny

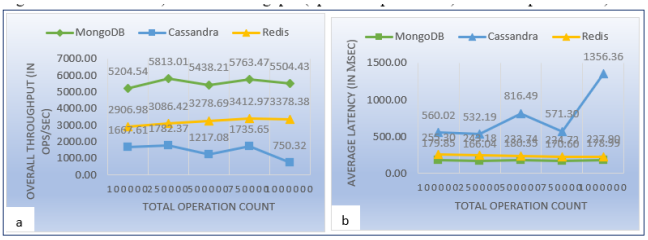

Fig 5. Workload C: a) Overall Throughput (operations per second) vs TotalOperations. b) Read Average Latency (in μs) vs TotalOperations

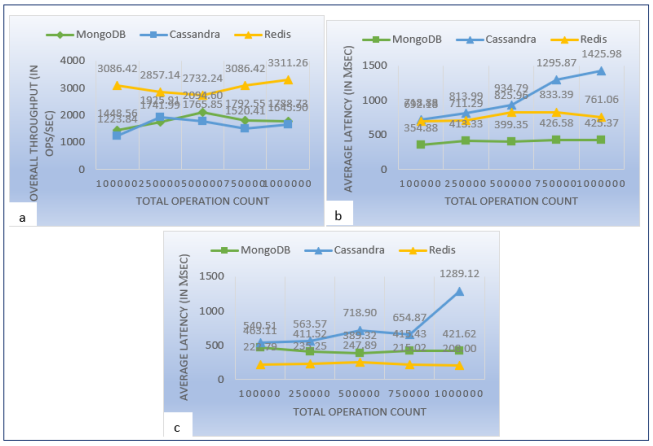

Fig 6. Workload D: a) Overall Throughput (operations per second) vs Total Operations. b) InsertAverage Latency (in μs) vs Total Operations. c) ReadAverage Latency (in μs) vs Total Operations

drop in performance when the number of operations reaches one million. Figure 6a shows that Redis maintains a significantly greater throughput than Cassandra and MongoDB.

Figure 6b shows that Cassandra still has a greater insert latency than Redis and MongoDB in workload D. In the case of Cassandra, there is a modest decrease in latency with an increase in operation counts, which is virtually consistent with MongoDB.

With a 95% read operation, the latency in read operations is lower for Redis as compared to MongoDB and Cassandra. From the Figure 6c MongoDB and Redis latency are almost consistent across the operation count. Overall, Radis's latency is lower than the other two databases.

For workload D Redis performance better in case of overall throughput and read operation.

3.5 Workload E (95% Scan and 5% Insert)

Compared to Redis and Cassandra, MongoDB has a substantially higher total operations per second for workload E. While MongoDB modestly increases throughput as the operation count reaches 7500,000, Cassandra loses throughput as the operation count rises. With Redis, throughput is almost constant throughout the process. Figure 7a shows that MongoDB consistently maintains a significantly greater throughput than Redis and Cassandra.

Figure 7b shows that Redis has a larger insert latency than Cassandra and MongoDB for workload E. In the case of Cassandra, there is a modest decrease in latency with an increase in operation counts, which is virtually consistent with MongoDB.

Figure 7c shows that Cassandra still has a greater insert latency than Redis and MongoDB in workload E. In the case of Redis, there is a modest decrease in latency with an increase in operation counts, which is virtually consistent with MongoDB. Therefore, MongoDB is the best option for handling workload E.

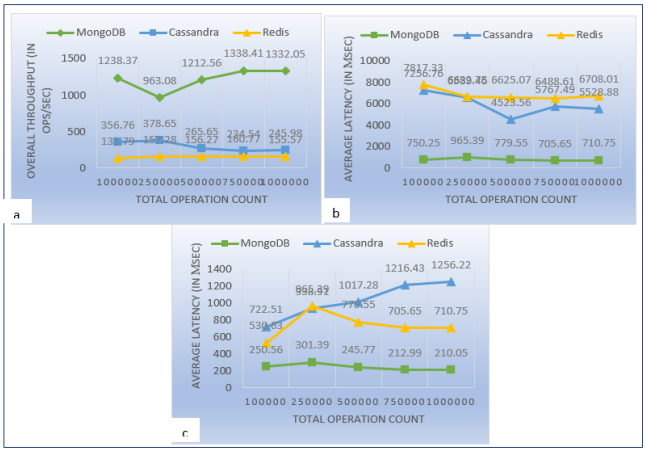

Fig 7. Workload E: a) Overall Throughput (operations per second) vs Total Operations. b) Scan Average Latency (in μs) vs Total Operations. c) Insert Average Latency (in μs) vs Total Operations

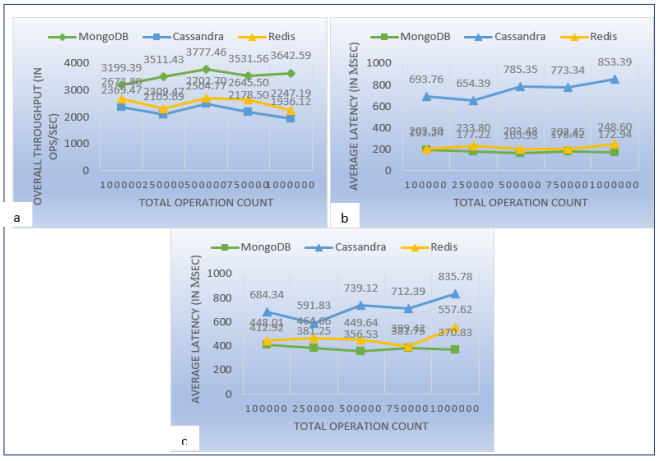

Fig 8. Workload F: a) Overall Throughput (operations per second) vs Total Operations. b) Read Average Latency (in μs) vs Total Operations. c) Read-Modify-Write Average Latency (in μs) vs Total Operations

3.6 Workload F (50% Read and 50% Read-Modify-Write)

Again, MongoDB outperforms Redis and Cassandra in workload F in terms of total operations per second. As with Cassandra and Redis, their performance falls as the number of operations rises. However, MongoDB shows a tiny gain in throughput at a million operations. Figure 8a shows that MongoDB consistently maintains a significantly greater throughput than Redis and Cassandra.

With workload F read operations, the latency in read operations is lower for MongoDB as compared to Redis and Cassandra. From Figure 8b, MongoDB and Redis latency are almost consistent across the operation count. Overall, MongoDB's latency is lower than the other two databases.

Figure 8c shows that Cassandra has a greater Read-Modify-Write latency than Redis and MongoDB in workload F. The latency changes with an increase in operation counts, somewhat decreasing for MongoDB and slightly increasing for Redis. Therefore, MongoDB is the best option for handling workload F of this kind.

4 Conclusion and Recommendations

After analyzing the results from the three NoSQL databases, MongoDB 4.4 as document store, Cassandra 4.0.3 as column store, and Redis 6.2.6 as key-value store, and after executing six workloads made up of 100000, 250000, 500000, 750000, and 1000000

operations, we came to the conclusion that the numerous optimizations used by the designers of NoSQL solutions to improve performance, such as good cache memory operation, have a direct impact on the execution time.

From the performance tests of MongoDB, Cassandra, and Redis, we have learned a few things.

• Redis has the best read performance of all the databases. This is due to the fact that data is stored and retrieved using volatile memory.

• In terms of read operations, MongoDB outperformed Cassandra. The register mapping for MongoDB is loaded into RAM as a result, improving reading performance.

• MongoDB outperformed Redis and Cassandra when it came to scan operations.

• Cassandra outperformed Redis in terms of scan operations.

• Cassandra was harder to work with when it came to reading and updating. This is mostly due to the lack of optimization for these types of procedures.

• In all workloads except workload D, MongoDB has significantly reduced latency across all operations.

As a consequence of our tests and research, we can conclude that MongoDB is a superior performing NoSQL database.

However, the study described in the article has a number of shortcomings that can be fixed with new research approaches. One of these options for expanding on the study that has been provided will entail evaluating several NoSQL databases over the cloud. The examination of additional NoSql databases for testing, in order to be able to test other elements of performance, might also be a second route for the development and enhancement of this article.

References

1) Martins P, Abbasi M, Sá F. A Study over NoSQL Performance. In: Advances in Intelligent Systems and Computing. Springer International Publishing. 2019;p. 603–611. Available from: https://doi.org/10.1007/978-3-030-16181-1_57.
2) Seghier NB, Kazar O. Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool. *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*. 2021;p. 1–6. Available from: https://doi.org/10.1109/ICRAMI52622.2021.9585956.
3) Nasar M, Kausar MA. Suitability of influxdb database for iot applications. *International Journal of Innovative Technology and Exploring Engineering*. 2019;8(10):1850–1857.
4) Kausar MA, Nasar M. An effective technique for detection and prevention of SQLIA by utilizing CHECKSUM based string matching. *International Journal of Scientific & Engineering Research*. 2018;9(1):1177–1182.
5) Kausar MA, Nasar M. SQL versus NoSQL databases to assess their appropriateness for big data application. *Recent Advances in Computer Science and Communications*. 20211;14:1098–108.
6) Kausar MA, Nasar M, Moyaid A. SQL Injection Detection and Prevention Techniques in ASP .NET Web Application. *International Journal of Recent Technology and Engineering (IJRTE)*. 2019;8(3):7759–66.
7) Bagga S, Sharma A. A Comparative Study of NoSQL Databases. In: Lecture Notes in Electrical Engineering. Springer Singapore. 2021;p. 51–61.
8) Gorbenko A, Romanovsky A, Tarasyuk O. Interplaying Cassandra NoSQL Consistency and Performance: A Benchmarking Approach. *Communications in Computer and Information Science*. 2020;1279:168–184. Available from: https://doi.org/10.1007/978-3-030-58462-7_14.
9) Eddelbuettel D. A Brief Introduction to Redis. . Available from: https://arxiv.org/pdf/2203.06559.pdf.
10) Meier A, Kaufmann M. Nosql databases. InSQL & NoSQL databases. Springer Vieweg. 2019. Available from: https://doi.org/10.1007/978-3-658-24549-8_7.
11) Meier A, Kaufmann M. SQL & NoSQL Databases. Berlin/Heidelberg, Germany; Fachmedien Wiesbaden. Springer Fachmedien Wiesbaden. 2019. Available from: https://doi.org/10.1007/978-3-658-24549-8.
12) Diogo M, Cabral B, Bernardino J. Consistency Models of NoSQL Databases. *Future Internet*. 2019;11(2):43–43. Available from: https://doi.org/10.3390/fi11020043.
13) Győrödi CA, Dumşe-Burescu DV, Zmaranda DR, Győrödi RŞ. A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. *Big Data and Cognitive Computing*. 2022;6(2):49–49. Available from: https://doi.org/10.3390/bdcc6020049.
14) Kaur R, Sahiwal JK. A review of comparison between NoSQL Databases: MongoDB and CouchDB. *International Journal of Recent Technology and Engineering*. 2019;p. 892–898. Available from: https://www.ijrte.org/wp-content/uploads/papers/v7i6s/F03820376S19.pdf.
15) Andor CF. Runtime Metric Analysis in NoSQL Database Performance Benchmarking. *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2021;p. 1–6. Available from: https://doi.org/10.23919/SoftCOM52868.2021.9559083.
16) Győrödi CA, Dumşe-Burescu DV, Zmaranda DR, Győrödi RŞ. A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. *Big Data and Cognitive Computing*. 2022;6(2):49–49. Available from: https://doi.org/10.3390/bdcc6020049.
17) Diogo M, Cabral B, Bernardino J. Consistency Models of NoSQL Databases. *Future Internet*. 2019;11(2):43–43. Available from: https://doi.org/10.3390/bdcc6020049.
18) Martins P, Tomé P, Wanzeller C, Sá F, Abbasi M. NoSQL comparative performance study. InWorld Conference on Information Systems and Technologies. Cham. Springer. 2021. Available from: https://doi.org/10.1007/978-3-030-72651-5_41.
19) Pandey R. Performance benchmarking and comparison of cloud-based databases MongoDB (NoSQL) vs MySQL (Relational) using YCSB. *Technical Report*. 2020. Available from: https://doi.org/10.13140/RG.2.2.10789.32484.
20) Matallah H, Belalem G, Bouamrane K. Evaluation of NoSQL databases: MongoDB, Cassandra, HBase, Redis, Couchbase, OrientDB. *International Journal of Software Science and Computational Intelligence (IJSSCI)*. 2020;12(4):71–91. Available from: https://doi.org/10.4018/IJSSCI.2020100105.
21) Matallah H, Belalem G, Bouamrane K. Comparative Study Between the MySQL Relational Database and the MongoDB NoSQL Database. *International Journal of Software Science and Computational Intelligence*. 2021;13(3):38–63. Available from: https://doi.org/10.4018/IJSSCI.2021070104.