



Article

Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB

Inês Carvalho , Filipe Sá and Jorge Bernardino * 

Polytechnic of Coimbra, Institute of Engineering of Coimbra—ISEC, Rua Pedro Nunes, Quinta da Nora, 3000-199 Coimbra, Portugal

* Correspondence: jorge@isec.pt

Abstract: NoSQL document databases emerged as an alternative to relational databases for managing large volumes of data. NoSQL document databases ensure big data storage and good query performance and are essential when the data scheme does not fit into the scheme of relational databases. They store their data in the form of documents and can handle unstructured, semi-structured, and structured data. This work evaluates the top three open-source NoSQL document databases: Couchbase, CouchDB, and MongoDB with Yahoo! Cloud Serving Benchmark (YCSB), which has become a standard for NoSQL database evaluation. The performance and scale-up of document databases are assessed using YCSB workloads with a different number of records and threads, where the runtime is measured for each database. In the experimental evaluation, we concluded that MongoDB is the database with the best runtime, except for the workload composed by scan operations. In addition, we identified CouchDB as the database with the best scale-up when varying the number of threads.

Keywords: NoSQL document databases; Yahoo! Cloud Serving Benchmark; Couchbase; CouchDB; MongoDB



Citation: Carvalho, I.; Sá, F.; Bernardino, J. Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB. *Algorithms* **2023**, *16*, 78. <https://doi.org/10.3390/a16020078>

Academic Editor: Angelo Montanari

Received: 12 December 2022

Revised: 12 January 2023

Accepted: 30 January 2023

Published: 1 February 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

NoSQL databases, also known as non-relational or not only SQL, emerged as an alternative to relational databases to deal with a large volume of data [1]. NoSQL offers high performance, schema flexibility, availability, data replication, and scalability, with the goal of solving the big data problems of volume, variety, and velocity [2].

There are four types of NoSQL databases: key-value, column-oriented, document, and graph [3]. NoSQL document databases have become very popular, with one of the main strengths being their flexibility regarding schema management. This paper is focused on the top three NoSQL document databases, according to the DB-Engine Ranking 2022: MongoDB, Couchbase, and CouchDB.

MongoDB is the most popular NoSQL database and is adopted by companies such as Vodafone, Bosch, SAP, CISCO, eBay, Google, Sage, and others. NoSQL document databases are important when the data scheme does not fit into the scheme of relational databases. They are a good choice for handling a large volume of data, enabling easy updates to schemas and fields. Besides understanding which document database is better, this experimental work also intends to help researchers and users to choose the suitable database according to their needs.

The performance of these databases was evaluated using Yahoo! Cloud Serving Benchmark (YCSB). The YCSB is an open-source benchmarking tool developed in 2010 that has become the *de facto* standard benchmark for NoSQL database comparison using CRUD operations. The YCSB is a benchmark tool whose goal is to evaluate different systems with the same workloads with a focus on performance and extensibility [4].

YCSB includes six standard workloads, and we added two more workloads to evaluate these three databases. These workloads are composed of reading, inserting, updating,

scanning, and read–modify–write operations. With this benchmark, we aim to know the behavior of the database engine when scale-up by varying the number of threads and records by using a single node. We evaluated the three databases using an ordinary and limited personal computer rather than a specialized computer.

Considering the runtime in the tests performed, MongoDB was the database with the best performance. However, MongoDB was the database with the worst runtime in the workload composed of scan operations. On the other hand, CouchDB was the database that showed the best scale-up with a variation in the number of threads, and it was the database with the best runtime with the scanning operation.

The following points are the main contributions of this paper:

- Analysis of the main characteristics of NoSQL document databases;
- Performance and scale-up evaluation of NoSQL document databases using YCSB benchmark;
- Study the impact of query execution time on the databases using a different number of threads and various records size.

The rest of this paper is organized as follows: Section 2 reviews related work about NoSQL database evaluation. Section 3 describes the three NoSQL document databases used in this work. Section 4 explains the Yahoo! Cloud Serving Benchmark. Section 5 presents the experimental results of YCSB using the databases. Finally, Section 6 presents the conclusions and future work.

2. Related Work

NoSQL document databases have become more popular and are widely used in different areas. Several studies compare and evaluate the performance of NoSQL databases, including some NoSQL document databases. The research on NoSQL databases is still growing, and only a few academic papers have been published. Thus, just a few studies compare and evaluate the performance of only the NoSQL document database type.

The NoSQL databases, ArangoDB, HBase, MongoDB, PostgreSQL, and an ORDBMS (object-relational database management system) were compared in [5]. In this study, a performance evaluation was conducted using YCSB with only three workloads and with 100% insert, 100% read, and 100% update, respectively. They varied the number of records by 10,000, 100,000, and 1,000,000 records and used just one and four threads. For each workload, ten runs were performed, and the average of these values was used as the final result of the experiment. The tests were performed on a Linux Ubuntu environment. The authors concluded that ArangoDB and MongoDB have similar performance when using a load of 10,000 records. However, when the workload increases, MongoDB performs better than ArangoDB. Unlike the study presented, we carried out more types of operations, such as read–modify–write and scanning operations. The number of records used and the number of threads was also different. We used 100,000, 1,000,000, and 10,000,000 records. We varied the number of threads, using one, three, and six threads. For each workload, we ran it three times, and the average of these values was used as the final result of the experiment, and the tests were performed on a Windows environment.

In [6], the authors evaluated the performance of four NoSQL document databases MongoDB, ArangoDB, OrientDB, and Elasticsearch. The authors used the standard YCSB workloads, except workload E, which includes operations such as scan. One million records were used, and the number of threads was varied by 1, 3, 6, 10, 20, 32, 50, 64, and 128 threads. The tests were performed in an Ubuntu environment, and it was concluded that the database with the best performance was MongoDB, followed by ArangoDB, OrientDB, and, finally, Elasticsearch.

The authors in [7] carried out a study to evaluate the performance of three document-type NoSQL databases: MongoDB, Couchbase, and RethinkDB. Apache JMeter was used, which is an open-source tool that performs tests and measures their performance. These tests are made up of operations such as delete, get, patch, and post. In this evaluation, a single thread and several threads were used. In contrast to the results obtained in the

present paper, these authors obtained a better performance with Couchbase in most tests, either with one or multiple threads. The exception was in the post-operation that obtained better results with MongoDB.

In [8], the authors evaluated two NoSQL databases: Cassandra and MongoDB. Cassandra is a column database, and MongoDB is a document database. The authors compared and analyzed the two databases using the YCSB, where they added two more workloads, G and H, that focus on update operations. In this work, we not only added two more workloads to focus on update operations, but we also used all the standard YCSB workloads, from A to F. These authors did not use workloads D and E because they tended to focus on update and read operations. Additionally, they defined the size per record with ten fields, each of 100 bytes, meaning 1 KB per record. The total records used were 100 thousand, 280 thousand, and 700 thousand. In this paper, we defined twenty fields each of 500 bytes, meaning 10 KB per record, and we used 100 thousand, 1 million, and 10 million records. We evaluate the databases in a Windows environment, and these authors evaluate them in an Ubuntu environment. Just like us, these authors also ran all workloads three times, and the values shown represent the average value of the three executions. They also concluded that MongoDB started to reduce performance with increasing data size and that Cassandra is faster than MongoDB, providing lower execution time regardless of database size.

The authors in [9] compared two NoSQL databases: HBase, which is a NoSQL column database, and MongoDB, which is a document database. The two databases were compared and evaluated using YCSB with two more workloads, G and H, that focus on update operations. The workloads D and E were removed from the experiments because they contain insert operations. The main purpose of this work was only the evaluation of read, write, and load operations. The databases were evaluated in an Ubuntu environment using 600,000 records. It was concluded that MongoDB was more efficient in read operations, and HBase performed better in write operations.

In [10], three NoSQL document databases were compared: Couchbase, CouchDB, and MongoDB. The databases were characterized, taking into consideration their architecture, main advantages, and limitations. Furthermore, the authors did not perform an experimental evaluation and only used the OSSPal methodology to compare the databases. OSSPal methodology combines quantitative and qualitative measures to assess open-source software, where seven categories are defined: functionality, operational software characteristics, software technology attributes, documentation, support and service, community and adoption, and development process. In the end, a score that identifies the best NoSQL document database was obtained. In this evaluation, the best score was obtained by MongoDB, followed by Couchbase and CouchDB.

In [11], two databases were compared: MySQL and MongoDB. The databases were evaluated using YCSB, but only workloads A, C, and F were used. These workloads are composed of the reading, updating, and read–modify–write operations. The number of records also varied between 5 hundred, 12 thousand, 500 thousand, 100 thousand, 150 thousand, and 200 thousand. The number of threads used also varied between 4, 8, 16, and 32. It was concluded that MongoDB has better performance than MySQL in all operations used. Unlike this author, we used 100 thousand, 1 million, and 10 million records and 1, 3, and 6 threads.

Unlike most works, we used a Windows environment rather than an Ubuntu environment because, according to Statista and Stat Counter Global Stats websites, “Microsoft Windows is the dominant desktop operating system worldwide”.

3. NoSQL Document Databases

NoSQL databases were created to scale easily as data grow. Their main advantage is dealing with unstructured data such as text files, email, and multimedia files. NoSQL databases are known to have good horizontal scalability [12]. On the contrary, relational databases are known for having a predefined structure for their tables and for using SQL (Structured Query Language) as their standard query language. NoSQL databases do not

have a predefined structure or a standard query language since each one usually provides its own query language [13].

NoSQL document databases store and organize their data in the form of document collections. Collections are equivalent to tables in relational databases, and documents can be seen as records in relational tables. The difference is that the documents do not have a defined structure, and each document may have different fields and attributes. Documents from the same collection must be similar, although the database system does not require it. Documents do not have a predefined structure and do not depend on each other, and when they present some structure, they are called semi-structured documents [1,2].

We selected the top three document databases according to the DB-Engines Ranking 2022: MongoDB, Couchbase, and CouchDB [14]. In the following subsections, we describe each one of these NoSQL document databases.

3.1. Couchbase

Couchbase is an open-source database document-oriented, developed in C++ by Couchbase Inc. in 2011. Couchbase is designed for web and mobile applications, where their documents are in JSON format and are stored in buckets. A bucket is a collection of documents that are made up of a unique key. Couchbase has its own query language, the N1QL. It provides nine programming languages and supports Linux, macOS, and Windows as operating systems. Couchbase can be downloaded at <https://www.couchbase.com> (accessed on 10 March 2022) and has three versions: Couchbase Server, Couchbase Capella, and Couchbase Lite.

Figure 1 illustrates a web interface for managing the Couchbase Server, including a dashboard that provides a graphical summary of the current server status from the statistics. A bucket tab is included, where it is possible to see the active buckets in the system, and allows the configuration of services and indexes. It is possible to add and remove cluster nodes and configure replication across data centers. It also creates and shows collections, documents, queries, and views [14,15].

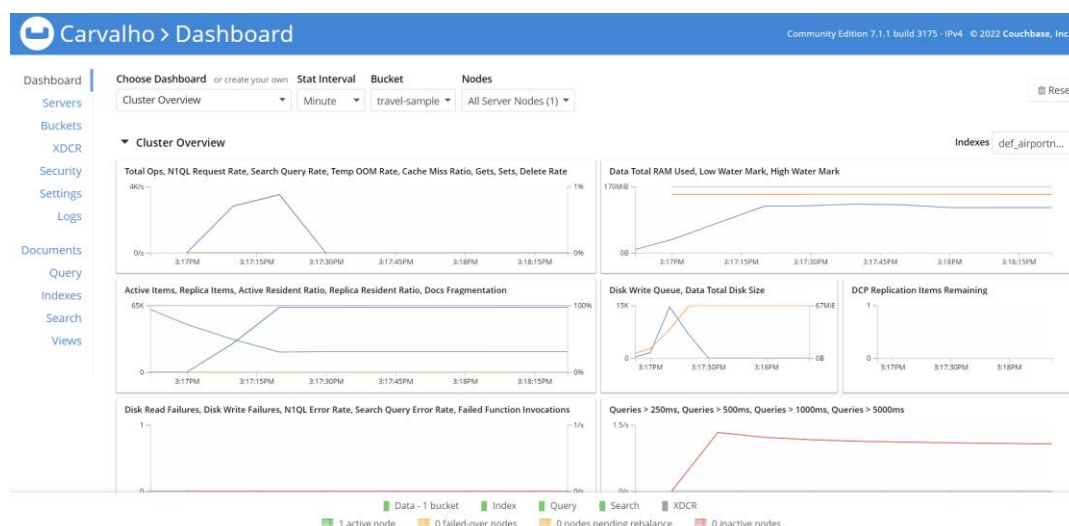


Figure 1. Couchbase web interface.

Couchbase Server has a set of services managed by the Couchbase Server cluster. These can be implemented, maintained, or deleted independently of each other. The services are data, query, eventing, indexing, full-text search, analytics, and backup.

Couchbase has a clustered architecture, where the cluster consists of a group of nodes that uses a peer-to-peer topology. The services are established per node, where each node can run on at most one instance of a service, and the data service must run on at least one node. The components of a Couchbase node include the cluster manager and the

other services provided by the Couchbase Server, also including the storage and cache management components [15].

The main characteristics and advantages of Couchbase are:

- Provides triggers, CRUD (create, read, update, and delete) operations, ad hoc queries, MapReduce, and indexes;
- It uses MVCC (MultiVersion Concurrency Control);
- It supports master–master replication and master–slave replication;
- It supports horizontal and vertical scaling.

Some of the limitations of Couchbase are as follows:

- It is not compatible with the four key properties of a transaction: atomicity, consistency, isolation, and durability (ACID properties);
- Indexing takes up too much RAM.

3.2. CouchDB

CouchDB is an open-source document-oriented database developed in Erlang by Apache Software Foundation in 2005. CouchDB documents are in JSON format and use JavaScript as a query language. CouchDB is compatible with the well-established operating systems Linux, macOS, Windows, Android, Solaris, and BSD and supports 17 programming languages [14,16].

CouchDB is a schema-less database designed for web and mobile applications suitable for CRM (customer relationship management) and CMS (content management system) systems. It provides a web application called Fauxton, as shown in Figure 2. In the Fauxton interface, it is possible to visualize and create collections, documents, indexes, and views. It also allows security management, such as adding users, assigning permissions, and configuring authentication mechanisms. CouchDB can be downloaded from <https://couchdb.apache.org/> (accessed on 8 March 2022).

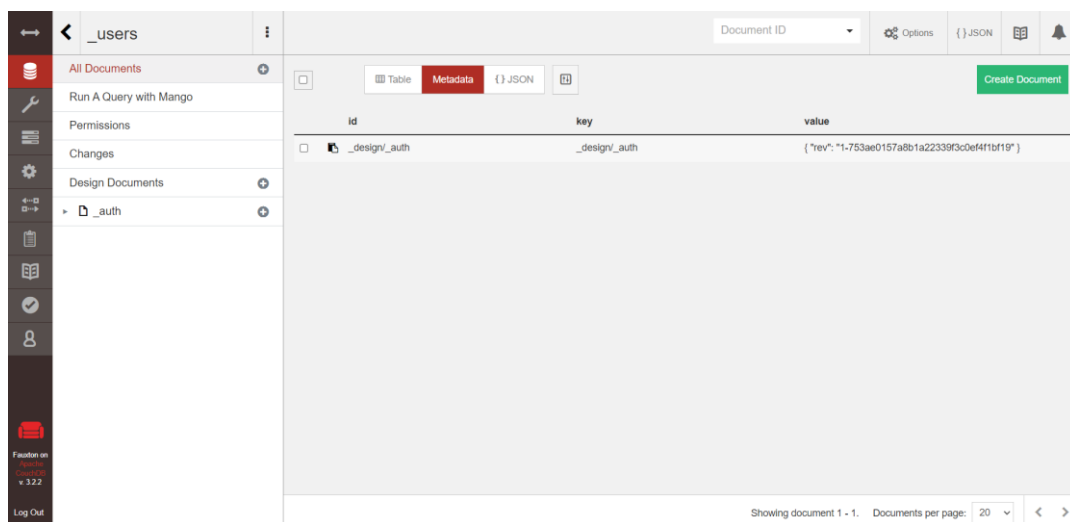


Figure 2. CouchDB web interface: Fauxton.

The main characteristics and advantages of CouchDB are:

- Provides triggers, CRUD operations, and MapReduce;
- It uses master–master replication;
- It supports horizontal scaling;
- It uses MVCC.

Some of the limitations of CouchDB are as follows:

- Views are temporary;
- Does not have support for ad hoc queries.

3.3. MongoDB

MongoDB is an open-source document-oriented database developed in C++ by MongoDB Inc. in 2009. MongoDB is designed for web and mobile applications and IoT. MongoDB uses documents in JSON format but stores them in BSON format and has its own query language, the MongoDB query language. MongoDB is compatible with the well-established operating systems Linux, macOS, Windows, and Solaris, and it supports 17 programming languages [14,17].

MongoDB offers three software versions: Atlas, Enterprise Advanced, and Community Edition. MongoDB includes the GUI (graphical user interface), the MongoDB Compass, a web application called Mongo Express, and a shell called mongo. MongoDB can be downloaded from <https://www.mongodb.com> (accessed on 9 March 2022).

Figure 3 illustrates the MongoDB Compass interface. It is possible to change and delete collections and documents. In addition, we can explore and manipulate data, create queries, make pipeline aggregations, visualize documents, create indexes, and build schema validation rules. It has a dashboard that provides a graphical summary of the server's current state [18].

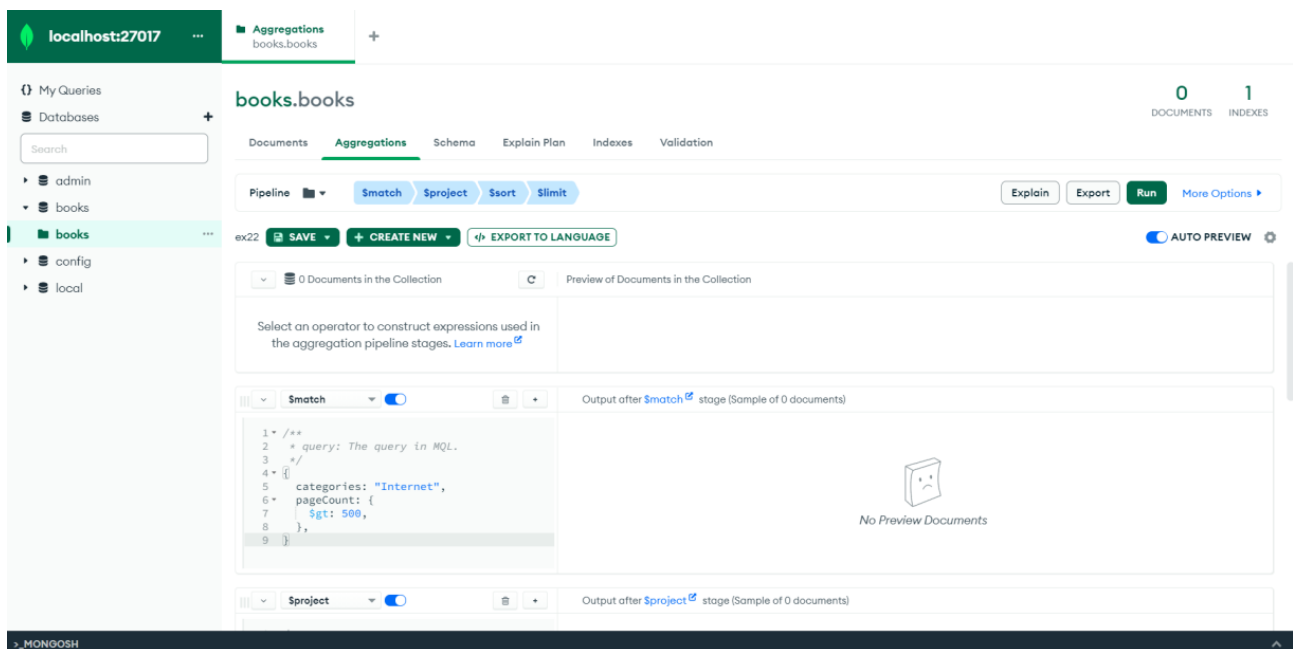


Figure 3. MongoDB web interface: MongoDB Compass.

In MongoDB, when we have a document in BSON format that exceeds 16 MB, it uses the GridFS, a distributed storage engine, to store large binary documents. As we can see in Figure 4, these documents are divided into several pieces or parts of 255 KB, except for the last piece that takes up the necessary space. The GridFS engine uses two collections. The first collection stores the document fragments, and the second stores the document metadata. GridFS is also used when we only want to obtain a part or a piece of the document without having to load the whole document into memory [19,20].

The main characteristics and advantages of MongoDB are:

- It provides indexing, ad hoc queries, CRUD operations, and MapReduce;
- It is ACID-compliant;
- It provides native drivers for programming languages and frameworks;
- It supports master–slave replication;
- It supports horizontal, vertical, and tiered scaling;
- It uses MVCC.

Some of the limitations of MongoDB are as follows:

- Data can easily be eliminated by mistake due to the lack of relations;
- Indexing takes up too much RAM;
- It just supports triggers on MongoDB Atlas.

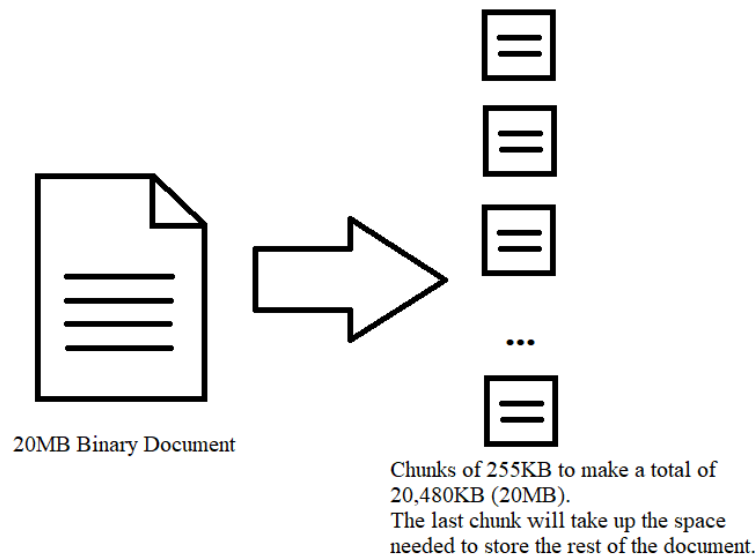


Figure 4. GridFS mechanism (based on [19]).

4. Yahoo! Cloud Serving Benchmark (YCSB)

The YCSB benchmark performs an essential role in the process of developing and testing data management systems. The YCSB has become the *de facto* standard benchmark to compare and measure the performance of different NoSQL databases under various workloads.

The YCSB architecture is illustrated in Figure 5, which consists of the YCSB Client (composed of workload executor), the client threads, and the statistics module. YCSB Client is a Java application that generates the data to be loaded into the target databases and the operations representing the workloads. In basic operations, the workload executor triggers several client threads. Each thread performs a series of operations that make calls to the database interface layer to load the database, known as the load phase, and to execute the workloads, known as the transaction phase. The database interface layer converts simple requests, such as read requests, from client threads into database calls. Threads measure latency and throughput achieved by their operations and pass these measurements to the statistics module. At the end of the workload execution, the statistics module aggregates these measurements and returns the execution time, the average latency, the latency in percentage, and a histogram or a time series of latencies [21,22].

The client receives two properties (name/value pairs) that define the operation to be performed [23]:

- Workload properties: define the workload; for example, the combination of reading and writing, the distribution to be used, and the size and number of fields in a record;
- Runtime properties: define the specific properties of a workload execution; for example, the database interface layer uses the properties used to initialize this layer, such as the database service hostname and the number of client threads.

In this work, we used the standard workloads provided by YCSB, called YCSB Core Workloads, which are described in Table 1. The standard workloads are from workload A to workload F. We defined two new workloads, G and H, which are similar to workloads B and C. However, workloads B and C focus on the read operation, and workloads G and H focus on the update operations.

YCSB Client

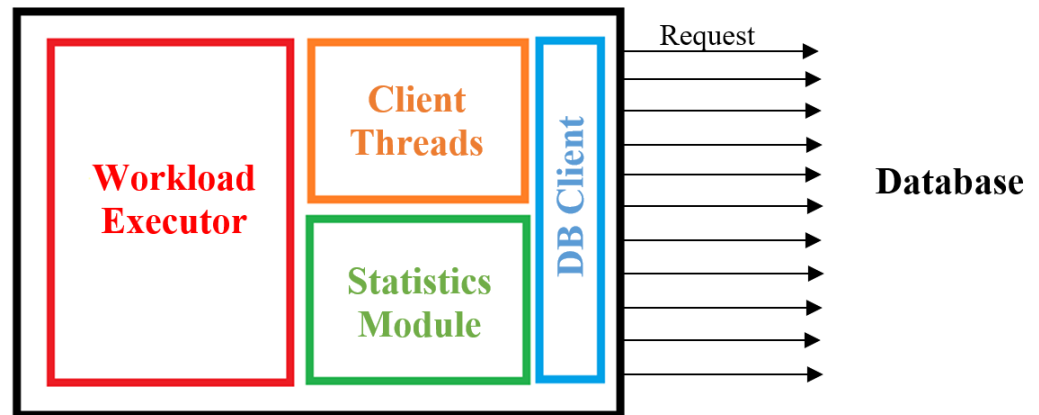


Figure 5. YCSB Client Architecture.

Table 1. Workloads used to evaluate the NoSQL databases.

Workloads	Operations	Distribution	Records	Threads	Data Size
A—Update Heavy	Read: 50% Update: 50%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
B—Read Mostly	Read: 95% Update: 5%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
C—Read Only	Read: 100%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
D—Read Latest	Read: 95% Insert: 5%	Latest	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
E—Short Ranges	Scan: 95% Insert: 5%	Zipfian Uniform	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
F—Read–Modify–Write	Read: 50% Read–Modify–Write: 50%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
G—Update Mostly	Update: 95% Read: 5%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB
H—Update Only	Update: 100%	Zipfian	100,000	1	Field size = 500 bytes
			1,000,000	3	Field number = 20
			10,000,000	6	500 bytes × 20 = 10 KB

In Table 1, we can see the distributions of operations and the type of distribution used. In all workloads, the data size was always 10 KB because we defined a number of twenty fields and each field has 500 bytes. The numbers of records were defined by stipulating a tenfold increase, starting with 100,000, 1 million, and 10 million records. The number of threads also varied between 1, 3, and 6.

When running a workload, the operations, distribution, and data size parameters always remain constant. The parameters that vary are the number of records and the number of threads. For example, a workload with one hundred thousand records and one thread, or one hundred thousand records and three threads, or one hundred thousand records and six threads, etc., for the remaining records. For each workload, three executions

were performed, and the average of these runtime values was used as the final result of the experiment.

The experimental setup for the tests was an HP Pavilion Laptop 15t-eg000 computer, with 476 GB of SSD disk, 15.4 GB of memory, an AMD Ryzen 5 4500U processor, with Radeon Graphics, 2375 MHz, six cores with six logical processors in a Windows 10 environment.

The test plan was designed to analyze the behavior of the databases when they scale up and their parallelism capacity. To this end, we used a reasonable number of records to simulate big data (100,000, 1 million, and 10 million records). Because of computer resources limitation, we did not use more than 10 million records, which occupy approximately 95 GB on disk. According to our tenfold increment, if we added the 100 million records, they would occupy approximately 953 GB on disk, and our computer can support only 476 GB.

We used threads to analyze the database engine's capability for parallelism. The environment used supports only six cores, so we varied the number of threads to one, three, and six.

5. Experimental Results

The following subsections present the results obtained by applying YCSB on NoSQL document databases: Couchbase, CouchDB, and MongoDB. In the experiments are used the workloads previously defined in Table 1, with 100,000, 1 million, and 10 million records, and the number of threads varies between one, three, and six. We conclude this section by discussing the results obtained with the three databases.

All the results obtained by applying YCSB on NoSQL document databases: Couchbase, CouchDB, and MongoDB are publicly available at GitHub: <https://github.com/isofiaC/Performance-Evaluation-of-NoSQL-Document-Databases> (accessed on 5 January 2023).

5.1. Couchbase

Table 2 presents the runtime of Couchbase, for all workloads, with 100,000 records, 1,000,000 records, and 10,000,000 records. Due to the lack of space, only the running time of the Couchbase with three threads is presented. Using Couchbase, workload E, with 100,000 records and 1,000,000 records, is the workload with the longest runtime because it is mostly composed of scan operations. On the other hand, with 10,000,000 records, the workloads with the longest execution time were A and F. Workload A is composed of read and update operations, and workload F is composed of read–modify–write operations. For all records used, workload C had the shortest runtime because this workload is composed only of read operations.

Table 2. Runtime of Couchbase with 3 threads.

Couchbase	100,000 Records	1,000,000 Records	10,000,000 Records
Workload A	10.28 s	1037.79 s	85,910.67 s
Workload B	9.27 s	687.31 s	33,770.34 s
Workload C	6.72 s	80.34 s	3423.30 s
Workload D	12.01 s	201.87 s	29,839.63 s
Workload E	373.08 s	3288.45 s	34,028.25 s
Workload F	33.16 s	2822.24 s	47,416.49 s
Workload G	20.63 s	523.50 s	6905.86 s
Workload H	22.88 s	83.78 s	1078.91 s
Total	488.03 s (8.13 min)	8725.28 s (145.42 min)	242,373.45 s (4039.56 min)

Table 3 presents all the runtime values (in seconds) for all the workloads used in Couchbase (Workload A, . . . , Workload H). From these results, it was possible to conclude that:

- Considering 100,000 records, we see that increasing the number of threads to three reduced the runtime almost in half;

- Considering the experiments with 1,000,000 records, we verified that by increasing the number of threads from one to three, the runtime decreased by more than half. However, when we modify from three to six threads, the runtime does not halve, only decreasing by 40%. This occurs because the computer used supports only six threads and has other processes running in addition to the experimental process. Thus, Couchbase does not take full advantage of all available threads;
- Taking into account 10,000,000 records, we see that as the number of threads increased, the runtime decreased by almost 60% each time the number of threads was incremented;
- When increasing from 100,000 to 1,000,000 records, we expected the runtime increase to be 10 times; however, there was an increase of 18.3 times. This increment can be explained by the limitations of computer hardware with a single physical processor and also by the use of computer resources by other processes running at the same time;
- When moving from 1,000,000 records to 10,000,000 records, we expected the runtime to increase 10 times since the workload also increased 10 times; however, there was a 23 times increase. This 23-fold increase is due to the fact that the workload was higher; therefore, more disk accesses had to be made, which made runtime much slower;
- When increasing from 100,000 records to 10,000,000 records, the runtime increases 421.3 times, when the expected increase would be 100 times since the workload also increased 100 times. This growth is explained by the limitations of computer hardware.

Table 3. Runtime of Couchbase for all workloads.

Couchbase	100,000 Records	1,000,000 Records	10,000,000 Records
1 Thread	1094.68 s	18,951.37 s	384,101.63 s
3 Threads	488.03 s	8725.28 s	242,373.45 s
6 Threads	256.29 s	6066.70 s	148,339.99 s
Total	1839.00 s (30.65 min)	33,743.33 s (562.39 min)	774,815.06 s (12,913.58 min)

5.2. CouchDB

Table 4 shows the CouchDB runtime results for all workloads with three threads and using different numbers of records. The results for one and six threads are available at: <https://github.com/isofiaC/Performance-Evaluation-of-NoSQL-Document-Databases> (accessed on 5 January 2023).

Table 4. Runtime of CouchDB with 3 threads.

CouchDB	100,000 Records	1,000,000 Records	10,000,000 Records
Workload A	10.58 s	1278.84 s	18,513.14 s
Workload B	14.83 s	790.41 s	28,537.45 s
Workload C	17.20 s	145.05 s	4150.06 s
Workload D	19.00 s	194.48 s	9021.59 s
Workload E	181.30 s	2806.93 s	45,257.58 s
Workload F	22.58 s	734.97 s	8988.81 s
Workload G	26.41 s	789.59 s	12,043.40 s
Workload H	22.37 s	356.89 s	6058.88 s
Total	314.27 s (5.24 min)	7097.16 s (118.29 min)	132,570.91 s (2209.52 min)

For all records, workload E is the workload with the longest runtime because this workload is composed of 95% scanning operations and only 5% of insert operations.

Table 5 shows the runtime results in seconds for all the workloads used in CouchDB. From these results, it was possible to conclude that:

- Considering 100,000 records, we see that increasing the number of threads from one to three caused the runtime to be more than halved. Although the same did not happen when switching from three to six threads, with the runtime being reduced by only 30%. This can be explained by the management of the threads that is performed by the database engine;
- Considering 1,000,000 records, we can see that with the increase in the number of threads, the runtime has almost halved;
- Considering 10,000,000 records, the runtime from one to three threads decreases by almost 60%, and from three to six threads, the runtime decreases by 50%. This shows that CouchDB presents good scale-up when using more threads;
- When increasing from 100,000 records to 1,000,000 records, the expected runtime was 10 times longer; however, there was an increase of 17.9 times;
- When increasing from 1,000,000 records to 10,000,000 records, the expected runtime increase was 10 times since the workload also increased 10 times, but it was found to increase 18.1 times;
- From 100,000 records to 10,000,000 records, the runtime increases by 323.7 times, when the expected increase was only 100 times since the workload increase was also 100 times. As explained before, this increase is due to the limitations of computer hardware and the high workload, implying more disk access.

Table 5. Runtime of CouchDB.

CouchDB	100,000 Records	1,000,000 Records	10,000,000 Records
1 Thread	742.38 s	12,233.93 s	222,046.97 s
3 Threads	314.27 s	7097.16 s	132,570.91 s
6 Threads	243.09 s	3937.96 s	66,041.70 s
Total	1299.74 s (20.50 min)	23,269.05 s (387.82 min)	420,659.55 s (7010.99 min)

5.3. MongoDB

Table 6 presents the MongoDB runtime for all workloads with all the records used and with three threads. Due to the lack of space, the runtime results with one and six threads are available at: <https://github.com/isofiaC/Performance-Evaluation-of-NoSQL-Document-Databases> (accessed on 5 January 2023).

Table 6. Runtime of MongoDB with 3 threads.

MongoDB	100,000 Records	1,000,000 Records	10,000,000 Records
Workload A	10.65 s	297.41 s	4950.83 s
Workload B	7.15 s	122.37 s	2564.01 s
Workload C	6.61 s	111.58 s	2305.59 s
Workload D	7.31 s	81.03 s	1427.47 s
Workload E	78.82 s	4491.09 s	126,310.74 s
Workload F	13.52 s	296.36 s	4756.72 s
Workload G	12.22 s	541.20 s	6765.96 s
Workload H	12.22 s	463.96 s	7034.36 s
Total	137.85 s (2.30 min)	6107.59 s (101.79 min)	151,164.85 s (2519.41 min)

In MongoDB, workload E has the worst runtime result, considering all the records used. This workload is mainly composed of scan operations, and since MongoDB performs many disk accesses, this causes an increase in runtime.

On the other hand, with 100,000 records, workload C has the best runtime because this workload is composed only of read operations. However, with 1,000,000 records, and

10,000,000 records, workload D had the best runtime results. This workload is composed of 95% read operations, and just 5% insert operations.

The workloads B, C, and D are mainly composed of read operations, which are faster than the other operations. This makes workloads B, C, and D show better runtime, even varying the number of records.

Table 7 shows the runtime results for all the workloads used in MongoDB. From these results, it was possible to conclude that:

- Considering 100,000 records, we verify that the runtime decreases more than halved when increasing from one to three threads. The same does not happen when increasing from three to six threads, as it drops only 20% of the runtime, representing poor scaling-up. This 20% can also be explained by the fact that the computer used only supports six threads and uses them for other processes that run simultaneously;
- Considering 1,000,000 records, we verify that the runtime was reduced by 50% when it switches from one to three threads. When it was changed from three to six threads, the decrease in runtime was even worse, decreasing by only 10%. This can happen because there is much processing, thus using the disk more frequently, which slows down the runtime;
- Considering 10,000,000 records, we verify that, once again, its performance has worsened. This is because it has more data to process and consumes more computer resources. When increasing from one to three threads, the runtime only decreases by 30%. It is worse when it goes from three to six threads, in which it only decreases by 10%. These results show that MongoDB scales up poorly when using more threads;
- When moving from 100,000 records to 1,000,000 records, the runtime was expected to increase by 10 times; however, there was a 37.4 times increase. This increment is much higher than Couchbase and CouchDB, with an increase of 18.3 times and 17.9 times, respectively. This growth can be explained by the limitations of computer hardware and the use of computer resources by other processes running at the same time;
- When moving from 1,000,000 records to 10,000,000 records, the increase expected was 10 times; however, there was an increase in a runtime of 23.8 times. This increment of 23 times is due to the fact that the workload is higher, and therefore, more disk accesses need to be made, which slows down the runtime;
- When moving from 100,000 records to 10,000,000 records, the runtime increases by 890 times, a much higher value than Couchbase and CouchDB, with an increase of 421.3 times and 323.7 times, respectively. This growth is also due to the justifications already presented in the previous points.

Table 7. Runtime of MongoDB.

MongoDB	100,000 Records	1,000,000 Records	10,000,000 Records
1 Thread	297.30 s	8883.50 s	197,138.39 s
3 Threads	137.85 s	6107.59 s	151,164.85 s
6 Threads	115.41 s	5609.53 s	141,708.75 s
Total	550.56 s (9.18 min)	20,600.62 s (343.34 min)	490,011.99 s (8166.87 min)

5.4. Discussion of the Results

Figure 6 shows, in logarithmic scale, the total values for 100,000, 1,000,000, and 10,000,000 records used in the three document databases. From this, we can conclude that:

- With 100,000 and 1,000,000 records, MongoDB was the database that had the shortest runtime. However, when using 10,000,000 records, CouchDB had the shortest runtime, followed by MongoDB and Couchbase;
- MongoDB had the worst runtime in workload E. On the other hand, Couchbase had the worst runtime using workloads A, E, and F. The operations that compose these workloads are scan in workload E, and read–modify–write in workload F;

- It was verified that the three document databases present good scale-up when moving from one to three threads. When moving from three to six threads, MongoDB presents the worst scale-up, especially when increasing the workload size;
- In some cases, CouchDB performed better than MongoDB, particularly when switching from three to six threads and also when using a large number of records, where the difference between runtime was greater.

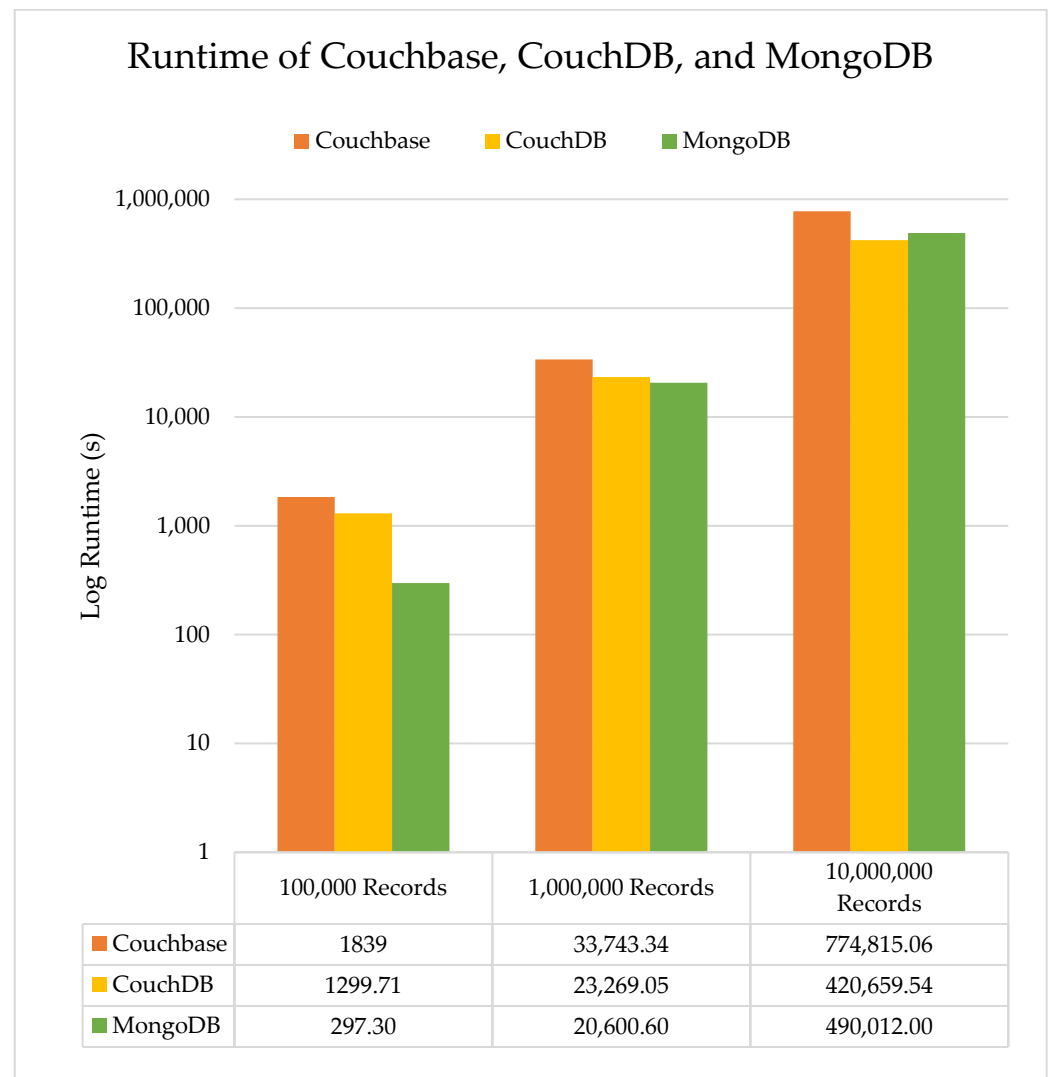


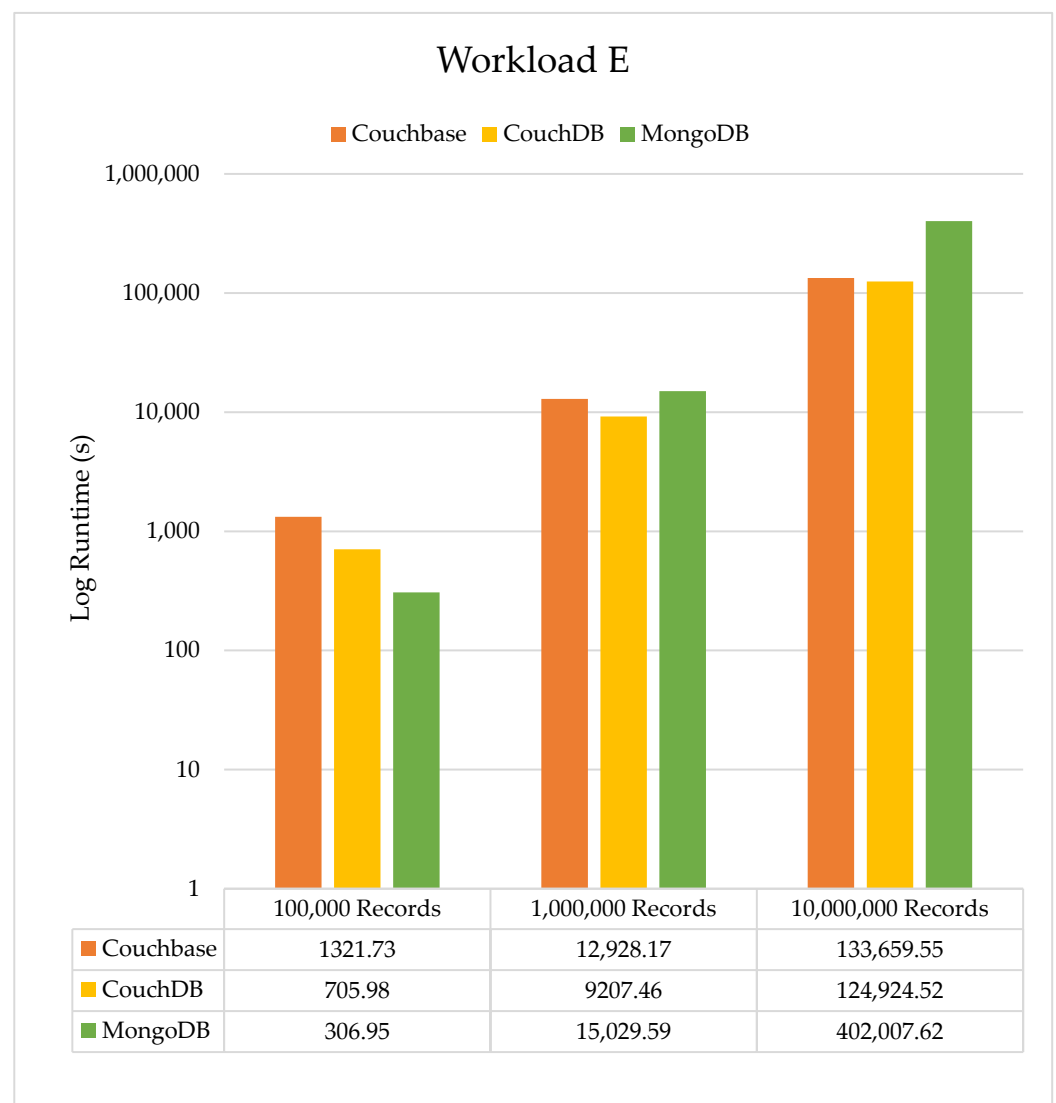
Figure 6. Log runtime representation, in seconds, of Couchbase, CouchDB, and MongoDB.

Table 8 shows the total values for all records used in the three document databases. CouchDB always had worse runtime using 100,000 and 1,000,000 records compared to MongoDB. When using 10,000,000 records, CouchDB had better runtime than MongoDB. This is because CouchDB shows better scale-up when increasing from three to six threads, unlike MongoDB. As we increased the number of records and the number of threads, CouchDB continued to maintain its performance, while MongoDB decreased its performance, causing the runtime to be lower on CouchDB than on MongoDB. Even though initially, with one thread, MongoDB had a better runtime than CouchDB.

Table 8. Runtime of all records used of Couchbase, CouchDB, and MongoDB.

	Couchbase	CouchDB	MongoDB
Total	810,397.40 s (13,506.62 min)	445,228.30 s (7420.47 min)	510,909.90 s (8515.16 min)

Another important factor that contributed to this outcome was workload E, which results are presented in Figure 7, using a logarithmic scale. Table 9 presents the total runtime of workload E employing the three databases. These results show that MongoDB took a long runtime in all workload E experiments, with a total of 6955.74 min. Moreover, CouchDB took 2247.30 min. This means that CouchDB had a runtime of 3.10 times faster than MongoDB, running workload E. MongoDB took approximately 5 days to run workload E. In contrast, CouchDB and Couchbase took about a day and a half to run the workload E.

**Figure 7.** Log runtime representation, in seconds, of workload E.**Table 9.** Total runtime of workload E.

Workload E	Couchbase	CouchDB	MongoDB
Total	147,909.45 s (2465.16 min)	134,837.95 s (2247.30 min)	417,344.15 s (6955.74 min)

Figure 8 shows the runtime using a logarithmic scale of all the workloads except workload E for the three sets of records. Table 10 presents the total runtime of all workloads except workload E. We removed workload E data because it is the only workload that has scan operations. With this selection, it is now possible to have an overview of all remaining operations, such as reading, updating, and inserting. As can be seen, MongoDB is the database with the lowest runtime, followed by CouchDB and Couchbase. This led to the conclusion that:

- MongoDB has a 3.28 times faster runtime than CouchDB;
- MongoDB has a 7.00 times faster runtime than Couchbase;
- CouchDB has a 2.13 times faster runtime than Couchbase.



Figure 8. Log runtime representation, in seconds, of all workloads except workload E.

Table 10. Runtime, in seconds, of all workloads except workload E.

	Couchbase	CouchDB	MongoDB
Total	662,487.95 s (11,041.47 min)	310,390.35 s (5173.17 min)	94,575.80 s (1576.26 min)

The workload E is composed of 95% of scan operations, and MongoDB uses many disk accesses to run the tests with this workload, which results in a longer runtime. On the contrary, CouchDB does not use as many disk accesses, and therefore its runtime is shorter.

This allows us to conclude that, when considering all the workloads, the document database with the best runtime is CouchDB, followed by MongoDB and Couchbase. However, when considering all performed workloads, with the exception of scan workload E,

the document database with the best runtime is already MongoDB, followed by CouchDB and Couchbase.

6. Conclusions and Future Work

NoSQL document databases store data in the form of documents and can handle unstructured and semi-structured data ensuring good query performance. In this paper, we selected the top three NoSQL document-oriented databases according to the DB-Engines Ranking of 2022 and assessed their performance using the YCSB benchmark. The YCSB benchmark was very useful as it allowed us to experimentally study the impact on the query execution time of using different types of operations, different numbers of threads, and various records size.

Compared to other scientific papers, we used a larger size per record, 10 KB, and used more records, 100,000 records, 1,000,000 records, and 10,000,000 records. In many papers, the authors did not use all types of operations because they did not use all the workloads included in YCSB. In the experimental evaluation, we used all the workloads available and implemented two more workloads (G and H). Unlike most works, we employed the more commonly used Windows environment rather than an Ubuntu environment.

This work allows us to conclude that CouchDB is the database with the best execution time. However, considering all the tests performed without the scan-denominated workload, which consists of scan operations, MongoDB is the database with the best execution time.

In future work, we intend to evaluate more NoSQL document databases with bigger workloads and compare them with relational databases in cluster and cloud environments.

Author Contributions: Conceptualization, J.B. and F.S.; Methodology, I.C. and J.B.; Software, I.C.; Validation, I.C., F.S. and J.B.; Formal analysis, I.C., F.S. and J.B.; Investigation, I.C.; Resources, I.C.; Data curation, I.C.; Writing—original draft preparation, I.C.; Writing—review and editing, J.B. and F.S.; Supervision, J.B. and F.S.; Project administration, J.B. and F.S.; Funding acquisition, J.B.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available in <https://github.com/isofiaC/Performance-Evaluation-of-NoSQL-Document-Databases>, accessed on 1 December 2022.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tannir, K. *RavenDB 2.x*; PACKT Publishing: Birmingham, UK, 2013.
2. Elmasri, R.; Navathe, S.B. *Fundamentals of Database Systems*, 7th ed.; Pearson Publishing: London, UK, 2016.
3. Dave, M. SQL and NoSQL Databases. 2012. Available online: <https://www.researchgate.net/publication/303856633> (accessed on 5 March 2022).
4. Reniers, V.; van Landuyt, D.; Rafique, A.; Joosen, W. On the state of NoSQL benchmarks. In Proceedings of the ICPE 2017—Companion of the 2017 ACM/SPEC International Conference on Performance Engineering, L'Aquila, Italy, 22–26 April 2017; pp. 107–112.
5. Dalström, I.; Ericsson, P. Performance Comparison between PostgreSQL, MongoDB, ArangoDB and HBase. Bachelor's Thesis, Information Technology. University of Skövde, Sweden, 2022.
6. Mishra, O.; Lodhi, P.; Mehta, S. Document Oriented NoSQL Databases: An Empirical Study. In *Data Science and Analytics*; Springer Nature: Singapore, 2018; pp. 126–136.
7. Augusto, D.; Morais, W.; Freitas, E. NoSQL real-time database performance comparison. *Int. J. Parallel Emergent Distrib. Syst.* **2018**, *33*, 144–156.
8. Abramova, V.; Bernardino, J. NoSQL Databases: MongoDB vs Cassandra. In Proceedings of the International Conference on Computer Science & Software Engineering, Porto, Portugal, 10–12 July 2013.
9. Matallah, H.; Belalem, G.; Bouamrane, K. Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB. *Int. J. Comput. Syst. Sci. Eng.* **2017**, *32*, 307–317.
10. Carvalho, I.; Sá, F.; Bernardino, J. NoSQL Document Databases Assessment: Couchbase, CouchDB, and MongoDB. In Proceedings of the 11th International Conference on Data Science, Technology and Applications, Lisbon, Portugal, 11–13 July 2022; pp. 557–564.

11. Pandey, R. *Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) vs. MySQL (Relational) Using YCSB*; National College of Ireland: Dublin, Ireland, 2020.
12. Leavitt, N. Will NoSQL Databases Live Up to Their Promise? 2010. Available online: www.leavcom (accessed on 5 March 2022).
13. Nayak, A.; Poriya, A.; Poojary, D. Type of NOSQL Databases and its Comparison with Relational Databases. *Int. J. Appl. Inf. Syst.* **2013**, *5*, 16–19.
14. DB-Engines. DB-Engines Ranking. 2022. Available online: <https://db-engines.com/en/ranking> (accessed on 27 March 2022).
15. Couchbase Inc. Couchbase Documentation. 2022. Available online: <https://docs.couchbase.com/home/index.html> (accessed on 24 April 2022).
16. CouchDB. Apache CouchDB®3.2.0 Documentation. 2022. Available online: <https://docs.couchdb.org/en/stable/> (accessed on 23 April 2022).
17. MongoDB. MongoDB Architecture Guide. 2021. Available online: <https://www.mongodb.com/collateral/mongodb-architecture-guide> (accessed on 29 March 2022).
18. MongoDB. Welcome to the MongoDB Documentation. 2021. Available online: <https://www.mongodb.com/docs/> (accessed on 25 March 2022).
19. MongoDB. GridFS. 2021. Available online: <https://docs.mongodb.com/manual/core/gridfs/> (accessed on 9 March 2022).
20. Wang, S.; Li, G.; Yao, X.; Zeng, Y.; Pang, L.; Zhang, L. A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 533. [CrossRef]
21. Kashyap, S.; Zamwar, S.; Bhavsar, T.; Singh, S. Benchmarking and Analysis of NoSQL Technologies. *Int. J. Emerg. Technol. Adv. Eng.* **2013**, *3*, 422–426.
22. Cooper, B. Yahoo! Cloud Serving Benchmark. 2010. Available online: <https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/> (accessed on 18 May 2022).
23. Hellerstein, J.M.; Chaudhuri, S.; Rosenblum, M.; Association for Computing Machinery; Special Interest Group on Management of Data; ACM Special Interest Group in Operating Systems. Benchmarking Cloud Serving Systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing, Indianapolis, IN, USA, 10–11 June 2010; p. 252.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.