A Performance Analysis of Different MongoDB Consistency Levels

45

Caio Lazarini Morceli, Valeria Cesario Times, and Ricardo Rodrigues Ciferri

Abstract

MongoDB DBMS is a NoSQL database management system that is known for its good performance in clusters of computers. In addition, the MongoDB has distinct levels of consistency that can be defined at the moment that a database operation is performed. The work described in this paper aims at investigating the MongoDB's performance using different levels of consistency. More specifically, the throughput and runtime performance of MongoDB operations are evaluated based on some workloads provided by the YCSB+T benchmark, which is an extension of the YCSB benchmark. The performance tests were performed according to two of the consistency level settings offered by MongoDB, w:1, j:true and readConcern (rc):local and w:majority, j:true and rc:local. The conclusion reached is that better performance is achieved when using the w:1, j:true and readConcern:local configuration. As a result, if there is a need for higher performance and data consistency is not an essential requirement, a write concern configuration that writes only to the primary node is a good solution.

Keywords

MongoDB · NoSQL · Performance · Benchmark · YCSB · YCSB+T · Consistency · WriteConcern · ReadConcern · ReadPreference

C. L. Morceli (✉) · R. R. Ciferri
Department of Computing, Federal University of Sao Carlos, São, Carlos, Brazil
e-mail: morcelicaio@estudante.ufscar.br; rrc@ufscar.br

V. C. Times
Center for Informatics, Federal University of Pernambuco Recife, Brazil
e-mail: vct@cin.ufpe.br

45.1 Introduction

The MongoDB is the most used NoSQL Database Management System (DBMS) in the world, as shown in Fig. 45.1, which shows the ranking of the most used DBMS according to a survey carried out on the StackOverflow (https://survey.stackoverflow.co/2022/) site, which is an environment that has its main content related to systems and database programming.

Several developers use the MongoDB for the most diverse purposes, because they can take advantage of the availability and scalability that the MongoDB provides in addition to the ACID guarantees that are offered by default for unique operations within the same collection [1].

45.2 Background

The main issue investigated and described in this paper is related to MongoDB's performance using different levels of consistency. More specifically, the throughput and runtime performance of MongoDB operations are evaluated based on some workloads provided by the YCSB+T benchmark, which is an extension of the YCSB benchmark. Therefore, this paper uses the YCSB+T benchmark to carry out tests and the MongoDB DBMS as a reference. In this way, its concepts are addressed with more clarity in detail.

This paper is organized as follows: In the first section, we provide a general introduction to MongoDB and its usefulness. In the second section, we present the tools used to write this document. We explain the levels of consistency found in MongoDB and its replica set architecture. We also report the YCSB benchmark along with its YCSB+T extension, which we use to perform data collection. In the third section, we show the different experimental results of this comparative study and analyze its results. Finally, the last section ends the

S. Latifi (ed.), *ITNG 2023 20th International Conference on Information Technology-New Generations*, Advances in Intelligent Systems and Computing 1445,
https://doi.org/10.1007/978-3-031-28332-1_45

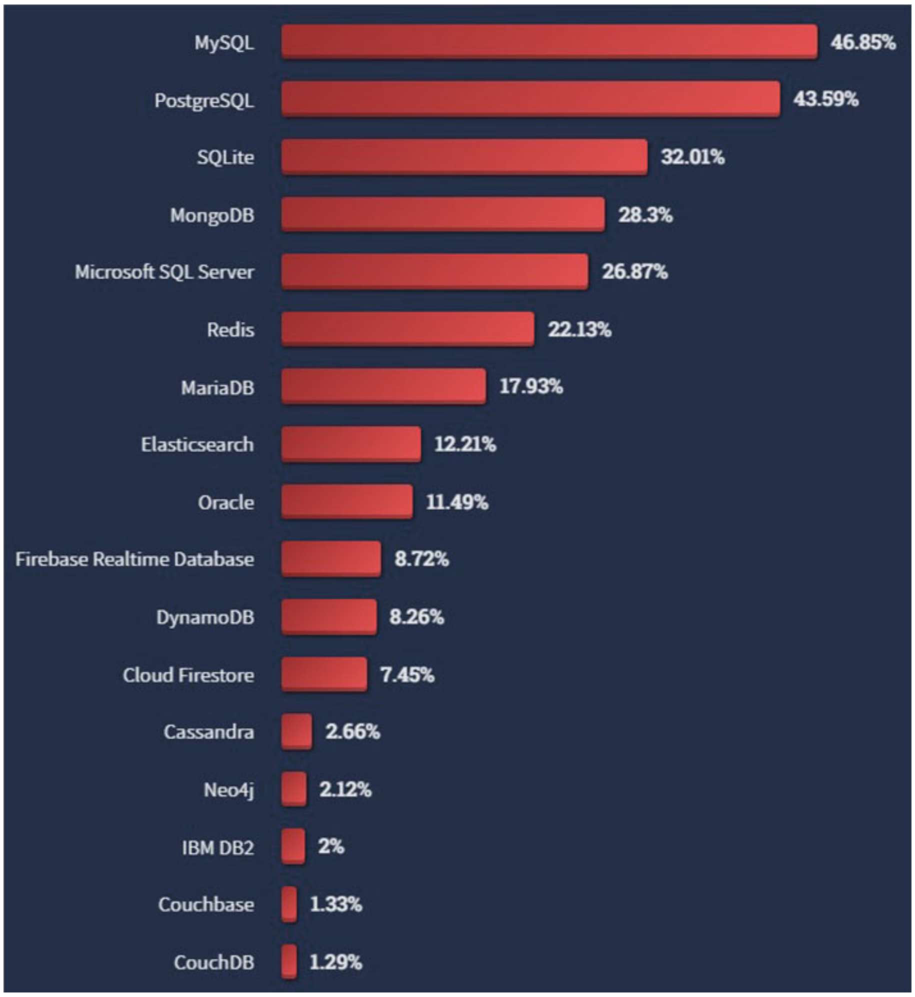

Fig. 45.1 Ranking of the mostly used DBMSs according to the StackOverflow website

document with a summary and some estimates for our future work.

45.2.1 MongoDB

MongoDB is a distributed document-based DBMS that lies within the NoSQL ecosystem. Data storage takes place in a binary form of JSON known as BSON [2], which is designed to represent JSON data more compactly and efficiently, using better encoding for handling numbers and other data types [3].

Its datasets are made up of collections and documents. In this way, MongoDB has collections and a set of documents that belong to a collection, which, in an analogous way, resembles the tables of a relational database [4].

When used in a cluster of computers, MongoDB has the replica set mode by default to allow greater availability of the stored data and can pass some parameters in the database operations to define the read and write levels of the operations.

45.2.1.1 Data Replication Process

To achieve high availability, MongoDB allows running the database as a group of nodes that work together, called *replica set*, acting as a consensus set in which each node participating in the *replica set* maintains a logical replica of the database state [5]. These *replica sets* use a single-leader-based consensus protocol, in which one node is used as the leader (i.e., primary node) and receives writes while the follower nodes (i.e., child nodes) are left with copies of backup [6]. The recordings coming from the clients to this primary node are placed in a replication *log* called *oplog*. This *oplog* is a collection that is stored in the primary node's local database and contains information regarding how to apply a database operation. In other words, the changes made to the database are recorded in *oplog*. The *oplog* is used by the secondary nodes so that the data replication takes place, since the secondary nodes execute the instructions contained in the *oplog* to maintain the similarity with the primary node. Therefore, the main existence of *replica sets* is for MongoDB to support high availability, allowing a *cluster* to have fault

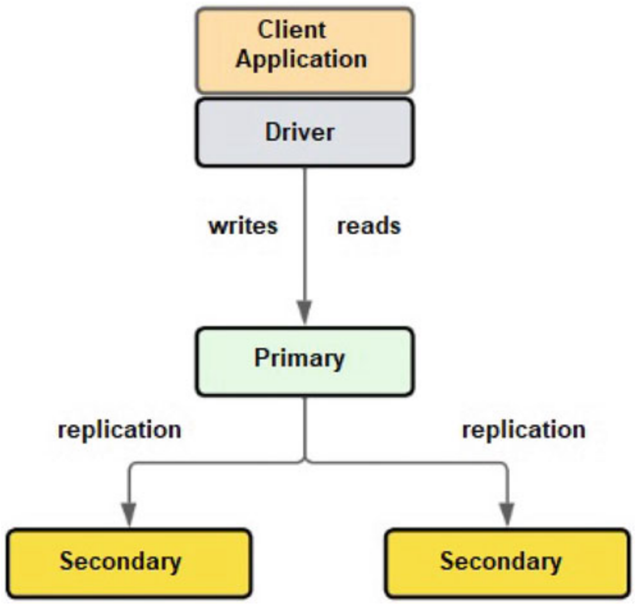

Fig. 45.2 MongoDB Replica Set

tolerance. In Fig. 45.2 the architecture of a *replica set* is shown.

45.2.1.2 MongoDB Consistency Levels

MongoDB allows developers to make consistency adjustments to operations performed through some parameters that are passed at the transaction time. Such parameters can be combined to reflect stronger or weaker integrity in the database, making it possible to configure some consistency levels.

MongoDB consistency levels are found in the parameters *write concern* (set to *w*) and *read concern*. Controling how data is written and read from cluster nodes is handled by these two parameters. In addition, parameters like *journal* and *read preference* can also be used to perform a consistency adjustment combination. All these parameters can be set for each database operation.

45.2.1.3 Write Concern and Journal

For *write concern*, the value set can be numeric (e.g., w: 1) or *majority*. The *write concern* parameter is responsible for controlling the number of nodes that must confirm the operation in a cluster before returning the response to the user. When *write concern* is numeric, it means that a certain number of nodes in the cluster must recognize the write. Regarding the *majority* value, it means that most nodes should recognize the recording. That is, if a cluster has three nodes, then two of them must confirm the write operation before returning to the user in a *w: majority* configuration.

Within the *write concern* setting, so that a commit is recognized, by default MongoDB has a setting called *journal*, which is a journal file on the primary node's disk. When *journal* is set to *j: true* (its default setting), writes are required to be persisted in the journal before completing. If it is set to *false*, the recording is recognized only when it reaches the server, that is, without writing it in the journal. In this configuration, data loss may occur while writing takes place on the primary node and a failure may occur. In this way, from these definitions, part of the consistency adjustment is achieved, as it configures the DBMS to guarantee the writing operation to a certain number of nodes in the cluster.

45.2.1.4 Read Concern

The *read concern* parameter establishes the data durability in addition to, on certain occasions, also determining the consistency of what is returned by the server.

As per the official MongoDB documentation, for *read concern*, the value set can be one of the following: *local, available, majority, snapshot* and *linearizable* [7]. Once the transaction of an operation is committed (i.e., *committed*), it becomes a locally committed operation. That is, it is confirmed only on the local node. Once the recording is in *oplog* and in the database, its replication can be performed to the secondary nodes. Thus, when it reaches the necessary condition defined by a *write concern* (e.g., *w: majority*) and performs the local *commit* on most nodes, it becomes mostly confirmed, meaning permanent durability in the *replica set*.

When using *read concern available*, the data returned from the server is not guaranteed to have been written to most nodes of the *replica set*. The *snapshot* option is only applied when dealing with multi-document transactions and ensures that customers see an intact *snapshot* of the data. Finally, the *linearizable* configuration remains. Such an option, when combined with *write concern w: majority*, it guarantees to return the effect of the most up-to-date majority write before the read operation starts. It has a linearization effect on operations, reflecting a stronger consistency guarantee.

When compared to the isolation levels reported in [8], the levels *read concern local* and *majority* can be considered analogous to the SQL isolation levels *READ UNCOMMITTED* and *READ COMMITTED*, respectively. This is due to the fact that the *read concern local* level can, in certain cases, visualize data that has not yet been confirmed by the majority. An operation using *read concern majority* is more similar to a *commit* path operation in standard SQL isolation.

45.2.1.5 Read Preference

The *read preference* parameter setting is responsible for indicating where a client sends read requests. In MongoDB, such read requests are sent to the primary node by default. In addition to this option, the MongoDB application developer may configure other settings for this parameter, such as directing read requests only to a secondary node, to a secondary node only when the primary node is unavailable, or finally to the closest server. The *read preference* setting also influences the DBMS by changing the magnitude of the consistency offered, considering that when reads are directed to the primary node, a higher consistency is achieved and when reads are sent to a secondary node, the result produces an eventual consistency.

Table 45.1 illustrates the settings for *read preference*.

45.2.2 YCSB

YCSB is a benchmark developed in the Java programming language that is used to generate the data to be loaded in a specific database chosen by the user and generate the operations that make up a specific workload chosen by the user [9]. In addition, one of its key features is that it is extensible, supporting the definition of new workloads. It was created with the purpose of enabling the conduction of performance comparisons of the new era of DBMS that focuses on working with cloud services. Many of these systems are referred to as key-value storage or recognized as NoSQL systems as well. These systems have common goals, such as having a strong adaptability of data on demand, also called elasticity, and horizontal scalability.

It is complex to evaluate different systems when trying to understand the performance of several systems, as some systems aim at optimizing writes while others aim at optimizing reads. This becomes a challenge as developers end up reporting specific performance numbers according to specific workloads for each of their systems. Another difficulty is related to performing an exact comparison when different numbers are derived based on different workloads. Thus, developers have the need to manually perform the comparison between different systems. Considering this prerogative, the YCSB benchmark was created with the idea of having a standard benchmark structure to help in the evaluation of different systems designed to run in the cloud.

The YCSB consists of two benchmark layers that aim to assess the performance and scalability of cloud service systems. One of the layers is the performance layer, which focuses on the latency of requests while the DB is under workload. According to Cooper et al. [9], latency is an important metric to be evaluated, considering that you usually have an impatient user waiting for a Web page to load, for example. This layer has the objective of verifying the compensation existing in the DBMS by measuring the latency as the throughput is increased until the moment when the DBMS becomes saturated and the performance stops increasing. That is, the hardware is kept constant while the workload size increases.

To drive the performance layer of the benchmark, it is necessary to have the workload generator that is shown in Fig. 45.3.

The performance layer of the benchmark serves the purposes of defining and loading the dataset and also performing operations on top of these datasets while performing the performance measurement that is handled by the statistics layer.

The other layer is the scalability layer, which is intended to check the scalability aspect of the DBMS. In this way, this metric is used to verify the behavior of the application as the data load increases. The scalability layer checks the performance impact as the number of machines in the system increases. That is, it allows knowing how the DB behaves as the number of machines increases. For this purpose, a given number of servers are loaded with data and then the workload

Table 45.1 Read Preference Levels in MongoDB

Read preference	Behavior
primary	All reads for the primary node
primaryPrefered	When possible, reads to the primary node
secondary	All reads for the secondary node
secondaryPrefered	When possible, reads to the secondary node
nearest	Directs reads to the nearest node

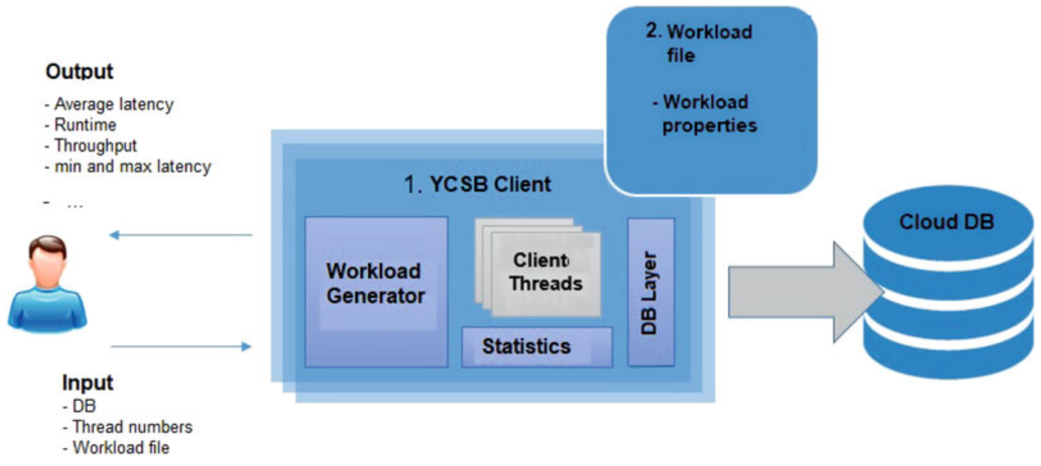

Fig. 45.3 YCSB client architecture

is executed. Next, the same task is performed by adding more machines to the cluster.

The YCSB benchmark, by default, uses some workloads to assess different performance aspects of a system. Each workload configures a specific combination of read/write operations and data sizes. These characteristics are used to evaluate systems at a specific point in the performance space.

45.2.3 YCSB+T

Problems regarding transaction support in NoSQL DBMS were identified and thus, the YCSB benchmark extension was created. From a new workload class, database operations performed within the scope of transactions can also be evaluated [10]. In this YCSB+T extension, a validation stage was included, which has the purpose of detecting and quantifying database anomalies that result from the user-defined workload.

Thus, the YCSB+T benchmark aims at keeping flexibility existing in the YCSB, and also allowing the user to choose the database interface to be implemented. Furthermore, it allows additional operations beyond the standard read, write, update, delete and scan operations and, most importantly, allows such operations to be grouped in the transaction scope. There is also a validation stage that is used to specify data consistency assessments conducted on the DBMS after a workload is completed. This validation stage is responsible for detecting and quantifying transaction anomalies.

For the experimental tests reported in this paper, the YCSB+T benchmark is being used to measure the throughput and runtime performance of operations, but in future work, transactional metrics will also be evaluated.

45.3 Experimental Results

The experimental evaluation was performed using the YCSB client on a physical machine: Core i7 9750H, 2.60GHz CPU, 16GB RAM, 1TB HD and SSD128GB. The software environment used was Windows 10 Home 64-bit. For the execution of the performance tests, a cluster of computers was used on MongoDB's free cloud platform, MongoDB Atlas [11], using a cluster in Replica Set mode with 3 nodes, being a primary node and two secondary nodes. The benchmark used is the YCSB+T which extends the 0.7.0 version of the YCSB. The benchmark package provides a set of standard workloads that can run as follows:

- Workload A (Update heavy): Consisting of a ratio of 50% Read and 50% Update;
- Workload B (Read heavy): Consisting of a ratio of 95% Read and 5% Update;
- Workload C (Read only): Its proportion is 100% read;
- Workload D (Read latest): Consisting of a ratio of 95% Read and 5% Inserts;
- Workload E (Short ranges): Consisting of a rate of 95% Scan and 5% Inserts;

In this way, 1000 records generated by the YCSB+T default properties file were loaded, each of which consists of 10 randomly generated fields of 100 bytes in the record identification key, which gives approximately 1 KB per write. The execution times of the workloads obtained during the read, write and update operations are also presented. All workloads perform 1000 operations and compute average execution times. Subsequently, an evaluation of the performance of some of the workloads chosen for the test is described.

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

Figure 45.4 shows the results obtained after running the workload A.

In this test, when w:1, j:true and readConcern:local, the performance is better than w:1, j:true and readConcern:majority. This reports that getting the write acknowledgment only on the primary node without replication acknowledgment to the secondary nodes translates to a shorter execution time and a higher rate of operations per second.

45.3.2 Workload B (Read Heavy): 95% Read and 5% Update

Figure 45.5 shows the results obtained after running the workload B.

In this test, when w:1, j:true and readConcern:local, the performance is better than w:1, j:true and readConcern:majority. This means that even when write operations are in the minority, an impact caused by ensuring data durability is perceived. Regarding the rate of operations per second, the result is very similar, since most operations are read and the readConcern level is the same for both configurations, which ends up not having a clearer impact on the test.

45.3.3 Workload D (Read Latest): 95% Read and 5% Inserts

Figure 45.6 shows the performance results obtained after running the workload D.

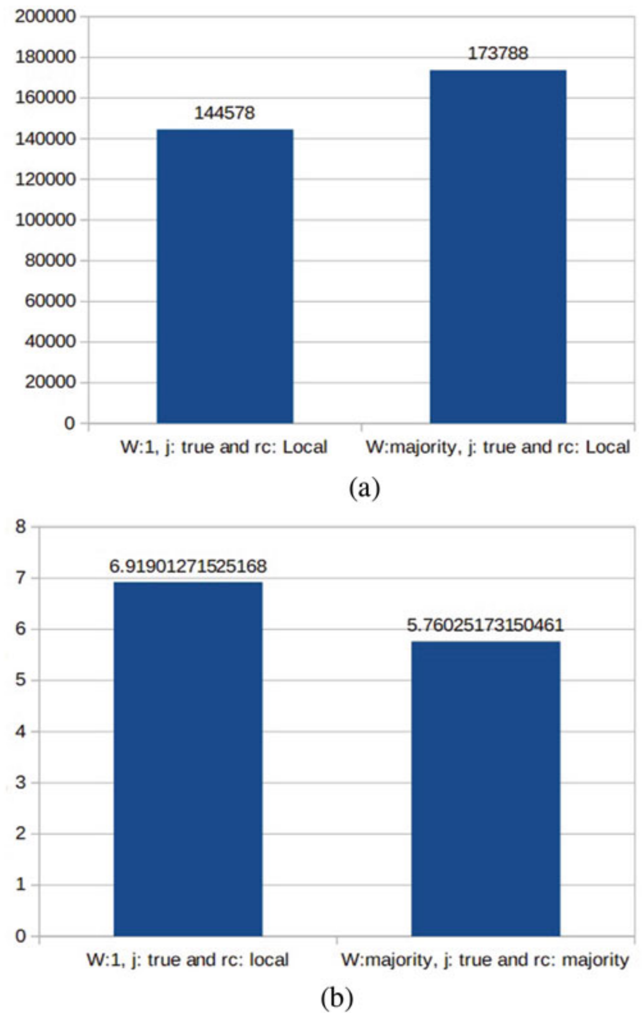

Fig. 45.4 YCSB performance comparison on workload A. (**a**) Run-Time (ms). (**b**) Throughput (ops/sec)

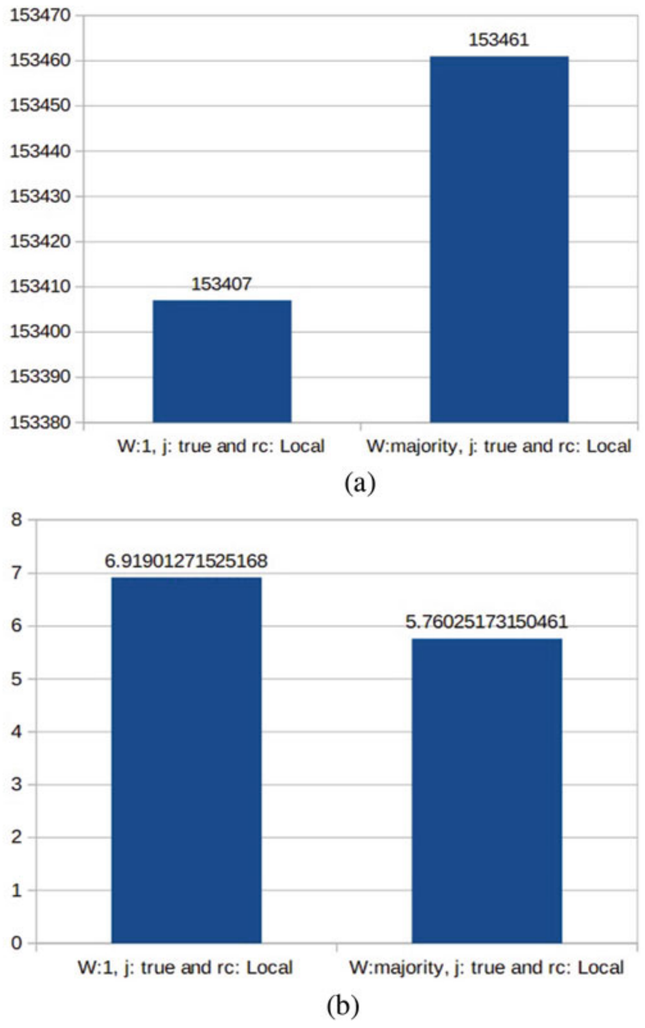

Fig. 45.5 YCSB performance comparison on workload B. (**a**) Run-Time (ms). (**b**) Throughput (ops/sec)

In this test, it is possible to observe that when w:1, j:true and readConcern:local, the performance is better than w:1, j:true and readConcern:majority. As with workload B, despite write operations being the minority, an impact caused by ensuring data durability is perceived. Regarding the rate of operations per second, the result is very similar, since most operations are read and the readConcern level is the same for both configurations, which ends up not having a clearer impact on the test.

In this work, tests were performed with the settings w:1, j:true and readConcern:local and w:majority, j:true and read-Concern:local of MongoDB. In addition, for data generation and analysis, the YCSB, YCSB+T benchmark extension was used. The performance differences between the configurable consistency levels that MongoDB offers allow the system developer to choose the most suitable configuration for the target application. Among the comparisons made, it is evident that data replication is mandatory. If you need higher performance and data consistency is not so much of a concern, a write concern configuration that writes only to the primary node is good solution. On the other hand, if durability guarantee is important for the other nodes, this incurs a performance loss, but guarantees more consistent data, as the database waits for the confirmation to be performed on most nodes in the cluster.

45.4 Future Works

Regarding future work, the main idea is to carry out new performance tests with other configurations that MongoDB offers between the existing levels of consistency. In addition,

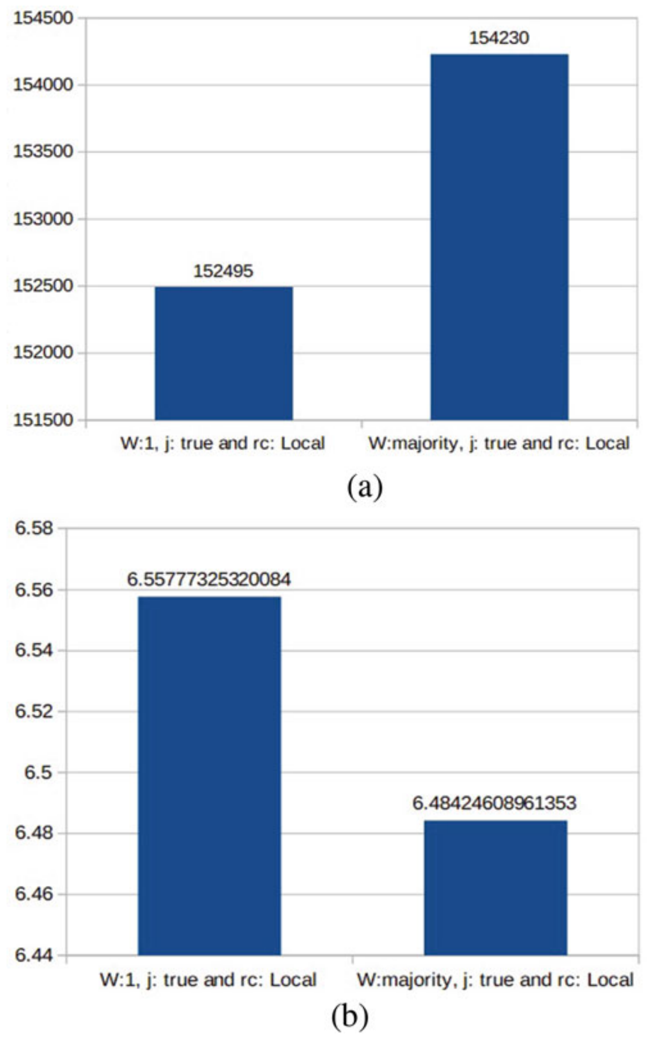

Fig. 45.6 YCSB performance comparison on workload D. (**a**) Run-Time (ms). (**b**) Throughput (ops/sec)

we plan to use all the other workloads that were not used in this experiment, such as *Workload C (Read only)* and *Workload E (Short ranges)*. Another point to consider, in addition to the performance of each tested level, will be to test the transactional layer of operations using the transactional evaluation layer, called CoreWorkload (CEW), that YCSB+T offers. We will seek to analyze the guarantees of consistency that these levels offered by MongoDB bring and we are also interested in evaluating the number of anomalies generated by the existing workloads in YCSB+T.

References

1. MongoDB Transactions. https://www.mongodb.com/docs/manual/core/transactions/
2. BSON Project. https://bsonspec.org
3. G. Harrison, M. Harrison, *MongoDB Performance Tuning* (Apress, Berkeley, 2021)
4. H. Matallah, G. Belalem, K. Bouamrane, Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB. Int. J. Comput. Syst. Sci. Eng. **32**(4), 307–317 (2017)
5. W. Schultz, T. Avitabile, A. Cabral, Tunable consistency in mongodb. Proc. VLDB Endow. **12**(12), 2071–2081 (2019)
6. KLEPPMANN. *Martin, Designing Data-Intensive Applications* (O'Reilly, 2017)
7. MongoDB Read Concern. https://www.mongodb.com/docs/manual/reference/read-concern/
8. H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil, A critique of ANSI SQL isolation levels, in *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1995), pp. 1–10
9. B. Cooper et al., Benchmarking cloud serving systems with YCSB, in *ACM Symposium on Cloud Computing (SoCC), Indianapolis, Indiana, June* (2010)
10. A. Dey, A. Fekete, R. Nambiar, U. Röhm, YCSB+T: Benchmarking web-scale transactional databases, in *2014 IEEE 30th International Conference on Data Engineering Workshops* (2014), pp. 223–230
11. MongoDB Atlas. https://www.mongodb.com/cloud/atlas