STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.02

PERFORMANCE BENCHMARKING FOR NOSQL DATABASE MANAGEMENT SYSTEMS

CAMELIA-FLORINA ANDOR

Abstract. NoSQL database management systems are very diverse and are known to evolve very fast. With so many NoSQL database options available nowadays, it is getting harder to make the right choice for certain use cases. Also, even for a given NoSQL database management system, performance may vary significantly between versions. Database performance benchmarking shows the actual performance for different scenarios on different hardware configurations in a straightforward and precise manner. This paper presents a NoSQL database performance study in which two of the most popular NoSQL database management systems (MongoDB and Apache Cassandra) are compared, and the analyzed metric is throughput. Results show that Apache Cassandra outperformes MongoDB in an update heavy scenario only when the number of operations is high. Also, for a read intensive scenario, Apache Cassandra outperformes MongoDB only when both number of operations and degree of parallelism are high.

1. Introduction

Big data came along with big challenges regarding how to store, manage and distribute a huge quantity of data, generated in short time and from diverse sources. NoSQL databases were the response to these challenges, specialized in solving specific big data problems. NoSQL database management systems are diverse and it is harder to choose the best fit for specific use cases than it is in the case of relational database management systems. Of course, relational database management systems present differences from one product to another, but those differences are less significant than the differences between NoSQL database management systems. Relational database management systems are based on the relational model, and the query language used is SQL,

Received by the editors: 18 April 2021.

2010 *Mathematics Subject Classification.* 68P15, 68P99.

1998 *CR Categories and Descriptors.* H.2.1 [**Database Management**]: Logical design – *Data models*; H.2.4 [**Database Management**]: Systems – *Distributed databases, Parallel databases*.

Key words and phrases. NoSQL database, performance benchmarking, MongoDB, Cassandra.

but NoSQL database management systems do not share the same data model or query language. It's quite common to see a different query language for each NoSQL implementation, and a specific data model, usually other than relational. Figuring out which NoSQL database management system fits best your use case is far more difficult than it seems at first, and it requires a thorough study of several NoSQL technical documentations and fine tuning. The hardware configuration is also important, and performance benchmarking is a good solution in this case. As NoSQL database management systems have a fast evolution, observing how their performance evolves between versions can offer meaningful knowledge.

This paper presents a performance benchmarking study which involves two of the most popular NoSQL database management systems, MongoDB (version 4.4.2) and Apache Cassandra (version 3.11.9). The benchmarking experiments were performed with YCSB, a free and open source benchmarking framework, which was also used to generate the data sets involved in the experiments.

2. Background

2.1. **NoSQL Data Models.** The NoSQL data models considered for this case study are column-family and document, which are two of the four main NoSQL data models. The remaining NoSQL data models are key-value and graph.

The key-value model is the least complex model, and NoSQL database management systems that use it have a very limited query language, but very fast operations. Both column-family and document model derive from the key-value model.

The graph model is the most complex NoSQL data model, and while it's a good fit for highly interconnected data, it has some drawbacks regarding horizontal scalability. NoSQL database management systems that use the graph data model have expressive query languages and constant read performance.

The document data model has much more in common with the column-family data model than it has with the other two main NoSQL data models. Both document and column-family data models support high availability, horizontal scalability, flexible schema and reasonable expressive query languages. Yet document NoSQL database management systems tend to offer more schema flexibility and richer query languages than column-family NoSQL database management systems. Also, column-family NoSQL database management systems tend to support faster write operations, even at scale.

2.2. **NoSQL database management systems.** MongoDB[8] and Cassandra[2] are two open source NoSQL database management systems. MongoDB is

based on the document data model and Cassandra is based on the column-family data model. MongoDB has a flexible schema, that can be easily modified, as the application's requirements evolve. It is more difficult to adapt the database schema in Cassandra, where data modeling is query driven (the application's queries must be known from the start). When designing database schema in Cassandra, the structure of the tables must optimize the application's queries. It's not uncommon to have several versions of a table, with minor structure changes in order to optimize different queries on the same data (data duplication is common in NoSQL databases). From the query language perspective, MongoDB is by far superior to Cassandra. While Cassandra has a query language somehow similar to SQL (but far more limited), MongoDB's query language is JavaScript based, rich and expressive. Also, MongoDB has support for many types of secondary indexes (text, geospatial, hidden, etc.), that are not available in Cassandra. High availability and horizontal scalability are well supported in MongoDB and Cassandra, but the distribution models are different.

2.3. **NoSQL performance benchmarking.** Performance benchmarking is quite handy when working with NoSQL databases. There are benchmarking tools that can be used only for a specific database management system (DBMS), like *cassandra-stress*[11] for Cassandra or *cbc-pillowfight*[10] for Couchbase. These types of tools are useful to test a given NoSQL DBMS in certain scenarios, but they don't help much when a comparison between several NoSQL DBMSs is the goal of the benchmarking experiment. For a fair comparison between several NoSQL DBMSs, a benchmarking tool which has support for all options considered is necessary. Unfortunately, there are not many benchmarking tools of this kind available in the open source section. *YCSB*[4] is an open source benchmarking framework aimed at cloud systems and NoSQL DBMSs. YCSB is a popular benchmarking framework, relatively easy to understand and use. It supports many NoSQL DBMSs and can be used on both Windows and Linux operating systems. Also, YCSB can be used to generate both the data set involved in testing and the database requests according to the chosen workload type. YCSB was also used by *MongoDB Inc.* for performance testing of MongoDB, see [12]. Other organizations and researchers used YCSB as well, for benchmarking NoSQL DBMSs. All NoSQL benchmarking experiments presented in [3], [5], [6] and [7] used YCSB as benchmarking tool. Other NoSQL benchmarking tools emerge, like NoSQLBench[9] (used by DataStax), but are still in early stages of development.

3. Case study

The case study presented in [1] analyzes the database performance metric called *throughput*, measured in number of operations/second. Other important and useful database performance metrics are *latency* (measured in number of microseconds/operation) and *total runtime* (the time necessary to run a certain number of database operations). The case study presented in this paper also analyzes the *throughput*. As NoSQL DBMSs evolve fast and change a lot from one version to another, it is important to see how those changes affect performance.

3.1. **Experimental setting.** I reproduced the experimental study presented in [1] using newer versions of database management systems and operating system without changing the hardware configuration. That experimental study involved three physical servers with the same hardware configuration. Windows 7 Professional 64-bit was the operating system installed on all servers. YCSB version 0.12.0, MongoDB version 3.4.4 and Apache Cassandra 3.11.0 were each installed on its dedicated server. I followed the same benchmarking methodology and I performed all benchmarking tests in the same order and under the same conditions as those performed in the experimental study presented in [1]. The YCSB, MongoDB and Cassandra versions used in my experimental study were newer. MongoDB version 4.4.2 was installed with default settings and the default storage engine, Wired Tiger. Apache Cassandra version 3.11.9 was installed with default settings and the settings necessary to avoid write timeouts:

- `counter_write_request_timeout_in_ms` set to 100000
- `write_request_timeout_in_ms` set to 100000
- `read_request_timeout_in_ms` set to 50000
- `range_request_timeout_in_ms` set to 100000.

YCSB version 0.17.0 was used to generate the data set and the database requests involved in tests.
Each application involved (Cassandra, MongoDB and YCSB) ran on its own server. The server configuration is as follows:

- OS: Windows 10 Professional 64-bit
- RAM: 16 GB
- CPU: Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 4 cores, 8 logical processors
- HDD: 500 GB.

I used the *YCSB client* to generate a data set having the same size and schema as the one used in the experimental study I reproduced (4 million records, each record made of 10 fields, each field contains a 100 byte string

value that was randomly generated). The same predefined YCSB workloads, Workload A (50% read operations, 50% update operations) and Workload B (95% read operations, 5% update operations) were involved, and the asynchronous version of Java Driver was used for both DBMSs. When a benchmarking test is run using YCSB, the workload type, the total number of operations to be executed and the number of client threads must be specified. After the test run, YCSB outputs a file that contains the measured results. For each *workload* considered (Workload A and Workload B), the *number of operations* parameter was set to 1000, 10000, 100000 and 1000000. For each workload and number of operations considered, the *number of client threads* parameter was set to 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512. Each test having a certain combination of values for DBMS, workload, number of operations and number of client threads was repeated three times. I will consider a set of tests all tests run for a combination of DBMS, workload and number of operations. Before and after the execution of each set of tests, database server information was captured. The database server was restarted before the execution of every set of tests. When all tests for the first workload were executed, the data set was deleted and a new data set with the same characteristics (schema and number of records) corresponding to the second workload was generated and loaded into the database.

3.2. **Results.** Each test was repeated three times for every combination of DBMS, workload, number of operations and number of client threads. As a consequence, a throughput average was computed for every combination of DBMS, workload, number of operations and number of client threads. This throughput average was used to create the following charts. A comparison between Cassandra and MongoDB for each combination of number of operations and workload is displayed in the first eight charts (Figures 1 to 8).
In case of Workload A (50% update operations, 50% read operations), Figures 1 and 2 show that MongoDB outperforms Cassandra by far, when the number of operations is relatively small (1000, 10000). When the number of operations is set to 100000, Cassandra's performance is almost as good as MongoDB's, as shown in Figure 3. However, Figure 4 shows that when the number of operations is set to 1000000, Cassandra outperforms MongoDB by far when the number of client threads is greater than or equal to 64.
In case of Workload B (5% update operations, 95% read operations), Figures 5, 6 and 7 show that MongoDB outperforms Cassandra by far when the number of operations is set to 1000, 10000 and 100000 operations. Figure 8 shows that Cassandra outperforms MongoDB only when the number of operations is set to 1000000 and the number of client threads is greater than or equal to 64.

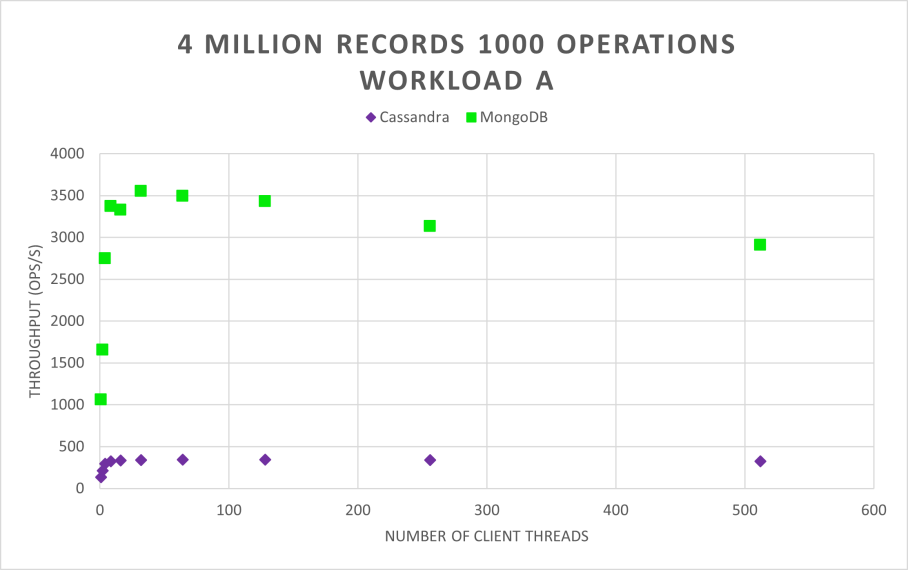

FIGURE 1. 4 Million Records Workload A 1000 Operations - Throughput

When compared to the results presented in [1], it can be observed that in the case of Workload A and number of operations set to 1000000, Cassandra outperformed MongoDB when the number of client threads was greater than or equal to 32, but the throughput does not exceed 45000 operations/second. My experimental study shows that, for the same scenario, while Cassandra outperfomes MongoDB only when the number of client threads is greater than or equal to 64, the throughput is significantly higher, with the maximum value around 76000 operations/second.
In the case of Workload B (5% update operations, 95% read operations), the results presented in [1] show that MongoDB outperformed Cassandra in all scenarios, and achieved high throughput values (between 68000 and 80000 operations/second) when the number of operations was high (100000 and 1000000 operations). My experimental study shows that, in case of Workload B and number of operations set to 1000000, Cassandra outperformes MongoDB when the number of client threads is greater than or equal to 64. Also, Cassandra presents a maximum throughput value around 62000 operations/second. Compared to MongoDB's throughput values presented in [1], MongoDB's throughput values observed in my study do not exceed 40000 operations/second.

The last four charts (Figures 9 to 12) display the individual performance of each DBMS, for each workload. Figure 9 presents the evolution of throughput on Workload A for Cassandra. When the number of operations is low

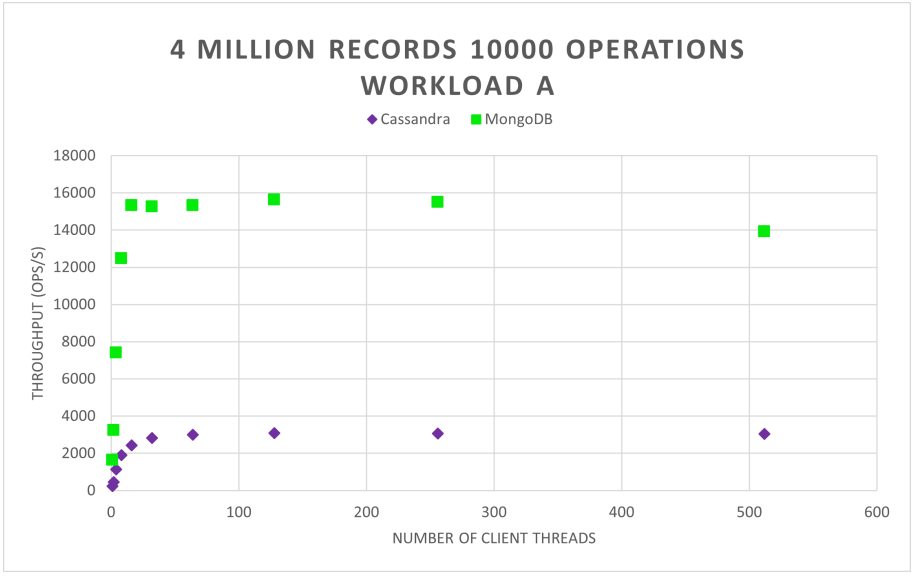

FIGURE 2. 4 Million Records Workload A 10000 Operations - Throughput

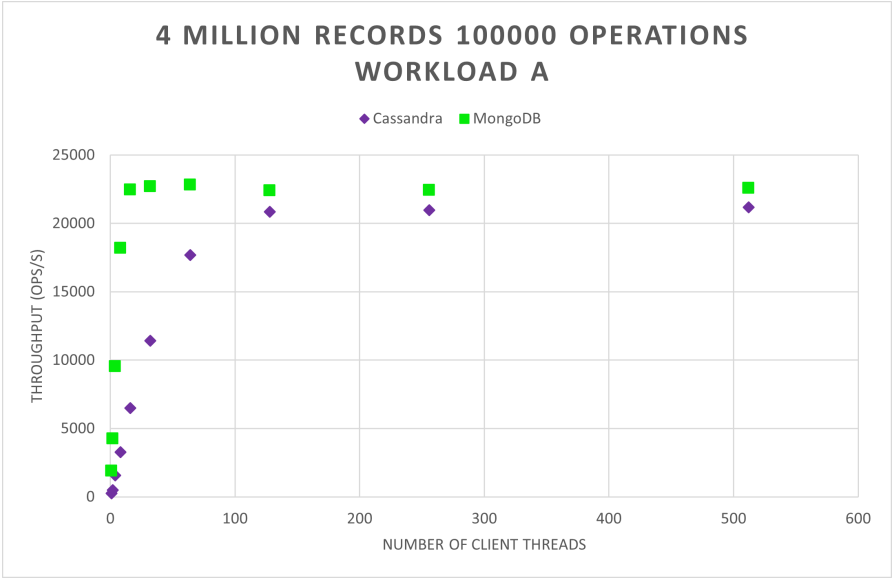

FIGURE 3. 4 Million Records Workload A 100000 Operations - Throughput

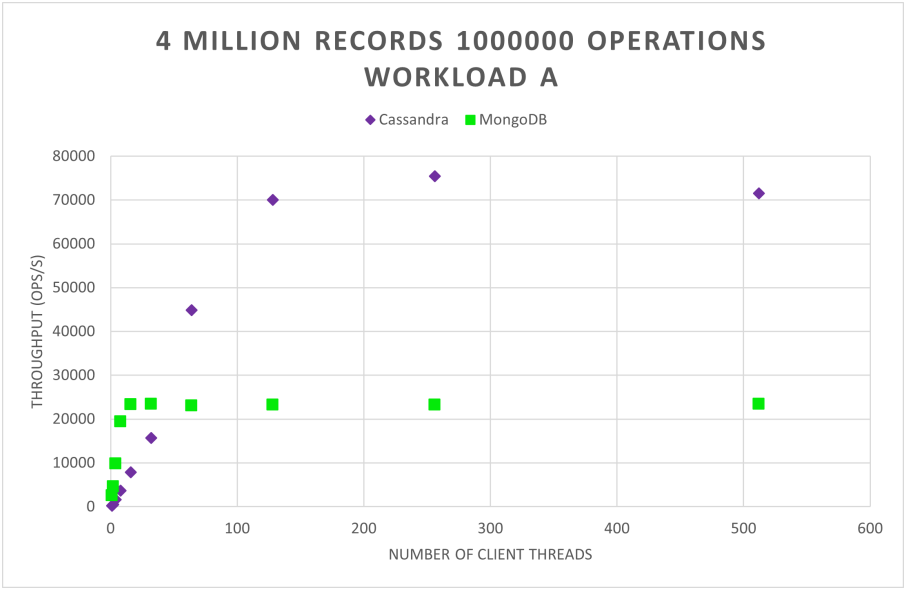

FIGURE 4. 4 Million Records Workload A 1000000 Operations - Throughput

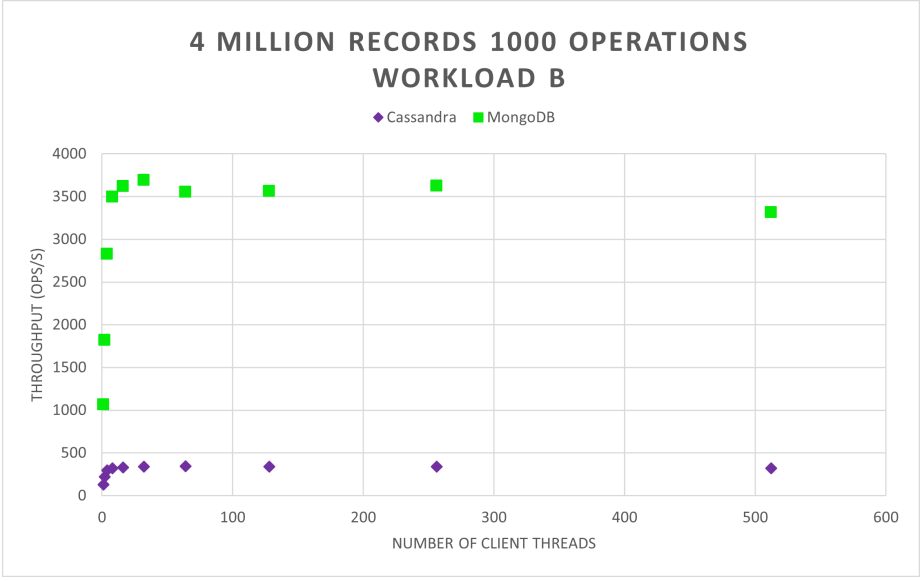

FIGURE 5. 4 Million Records Workload B 1000 Operations - Throughput

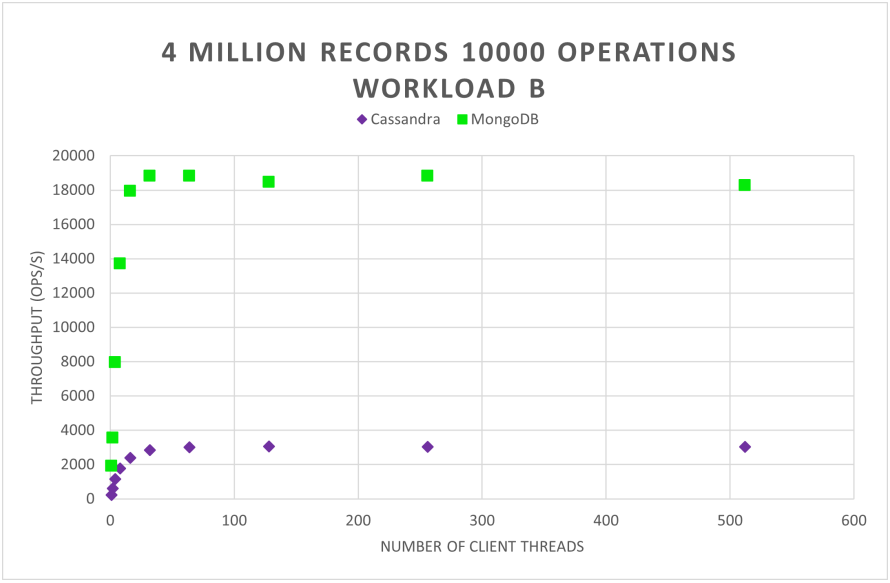

FIGURE 6. 4 Million Records Workload B 10000 Operations - Throughput

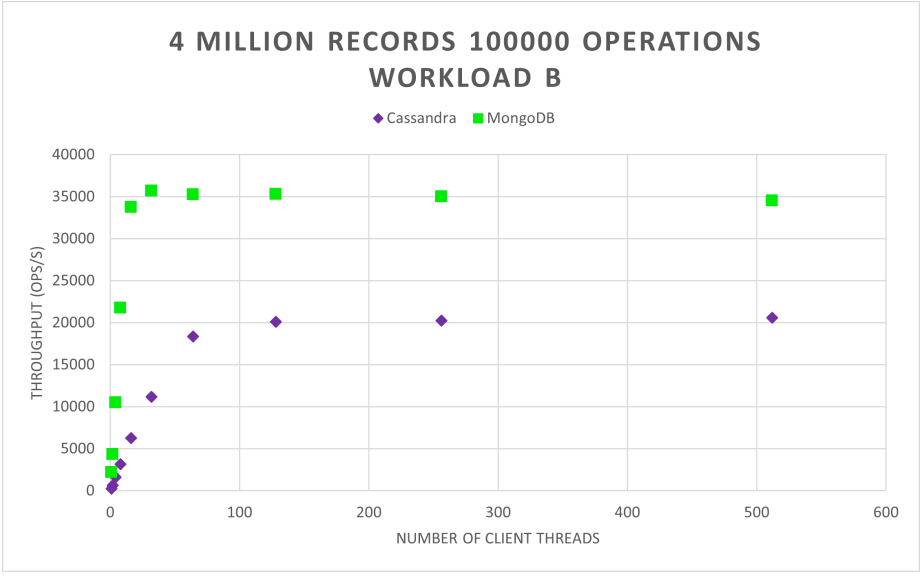

FIGURE 7. 4 Million Records Workload B 100000 Operations - Throughput

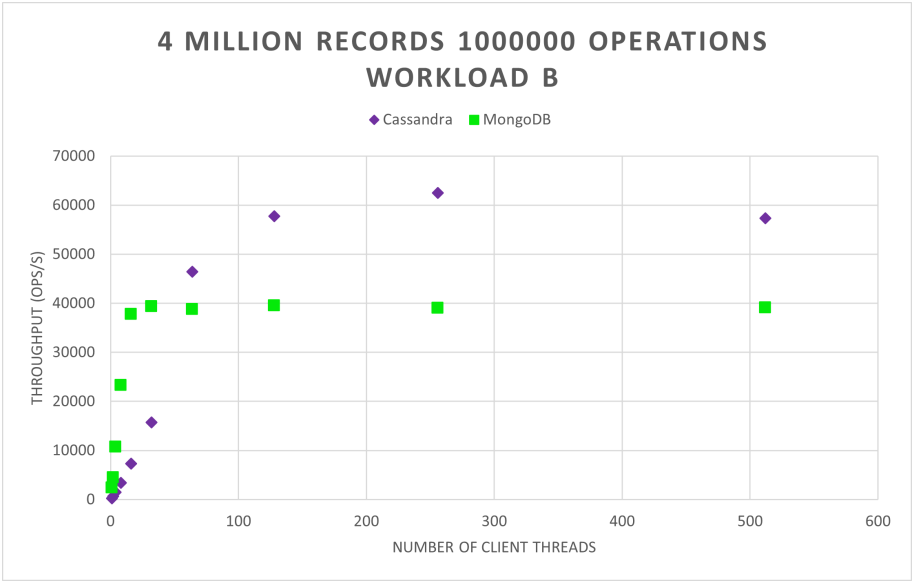

FIGURE 8. 4 Million Records Workload B 1000000 Operations - Throughput

(1000, 10000 operations), Cassandra's maximum throughput value does not exceed 3100 operations/second. When the number of operations grows at 100000, we can observe a significant increase of throughput (up to 22000 operations/second), especially as the number of client threads grows. The highest throughput values (up to 76000 operations/second) can be observed when the number of operations is set to 1000000, with a slightly decrease when the number of threads is set to 512.

In Figure 10, the evolution of throughput on Workload A for MongoDB is displayed. As the number of operations increases, the MongoDB throughput values increase as well, but remain almost constant when the number of operations is greater than or equal to 100000. The maximum throughput value does not exceed 23500 operations/second.

Figure 11 presents the evolution of throughput on Workload B for Cassandra. Cassandra's throughput patterns observed for Workload A are preserved, but the throughput values observed when the number of operations is set to 1000000 are lower and do not exceed 63000 operations/second.

Figure 12 displays the evolution of throughput on Workload B for MongoDB. The throughput remains the same as observed for Workload A, when the number of operations is set to 1000, but increases when the number of operations is greater than or equal to 10000. When the number of operations is

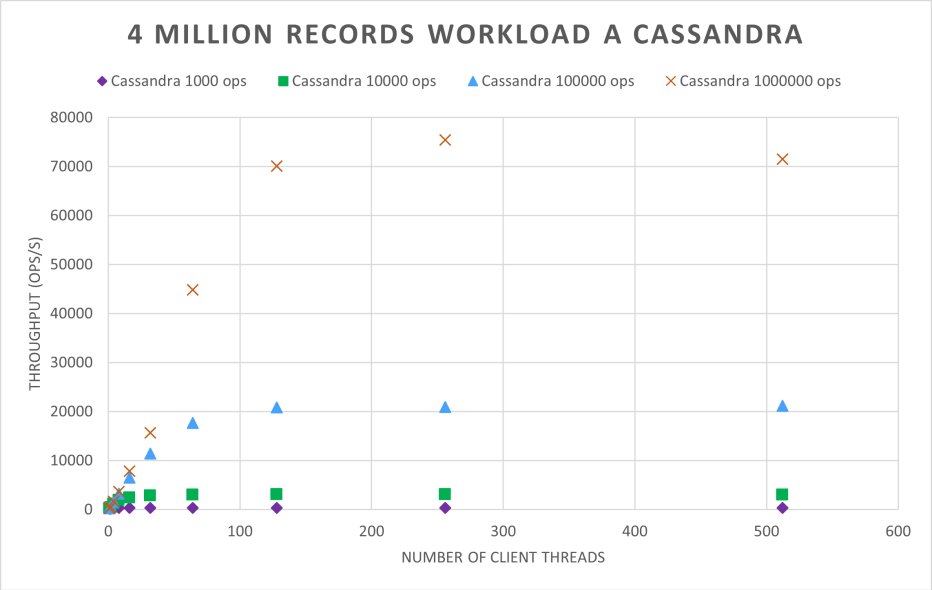

FIGURE 9. 4 Million Records Workload A Cassandra - Throughput

set to 100000, throughput values rise significantly (compared to those observed for number of operations set to 10000). Throughput values observed when the number of operations is increased at 1000000 are slightly higher than those observed for number of operations set to 100000, but do not exceed 45000 operations/second.

Cassandra version 3.11.9 presents great throughput improvements for both workloads, especially when the number of operations and number of client threads are high. MongoDB version 4.4.2 presents significantly lower throughput values for Workload B, a read intensive workload.

4. Conclusions and future work

NoSQL database management systems have a fast evolution, with significant changes between versions. Database performance benchmarking offers a good overview of how these changes impact application workloads. The experimental study presented in this paper reveals that the newer Cassandra version has important throughput improvements, especially when the number of operations and degree of parallelism are high, for both read and update operations. This is significant, as Cassandra is generally known to offer fast write operations, but not as fast read operations. Also, the newer MongoDB version presents decreased throughput values when the number of operations

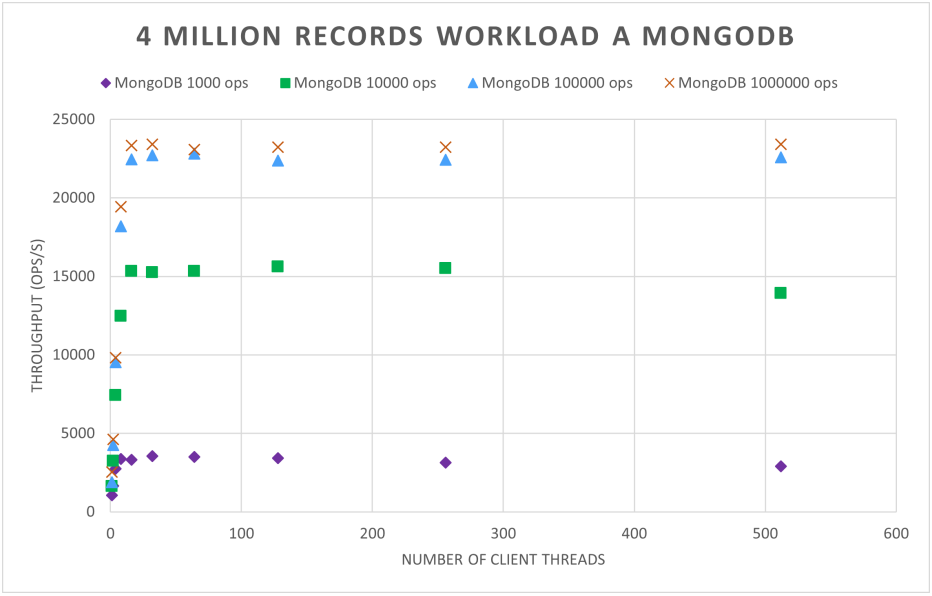

FIGURE 10. 4 Million Records Workload A MongoDB - Throughput

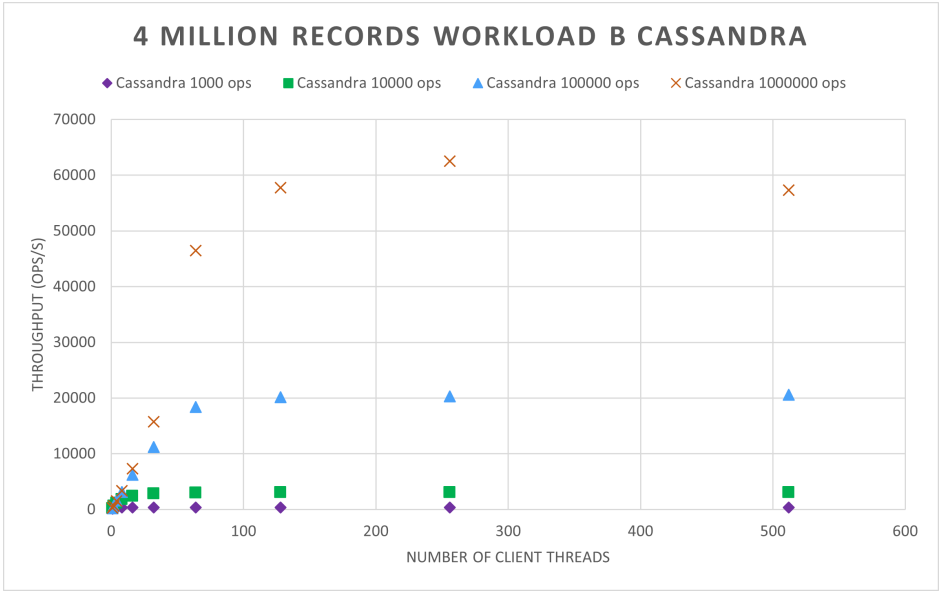

FIGURE 11. 4 Million Records Workload B Cassandra - Throughput

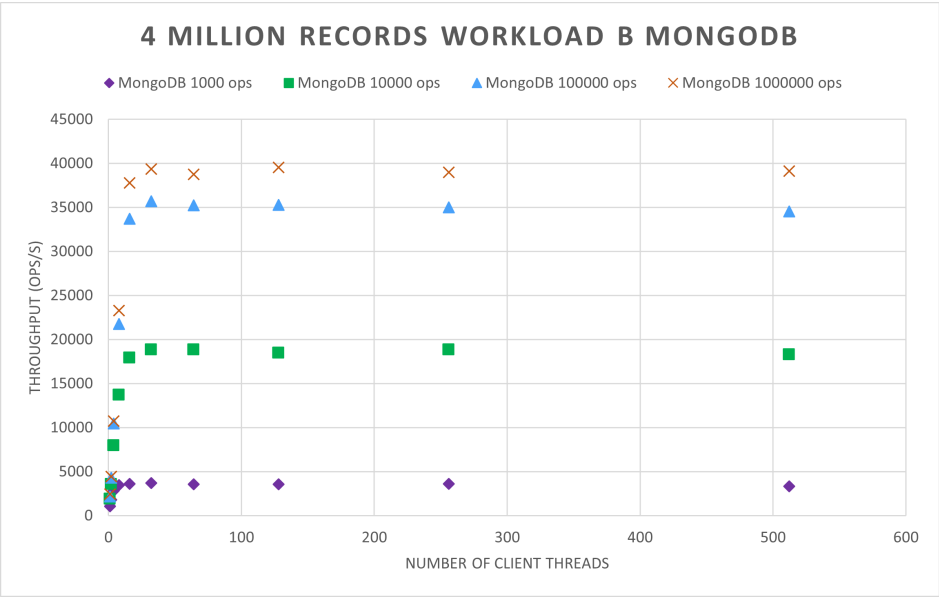

FIGURE 12. 4 Million Records Workload B MongoDB - Throughput

and degree of parallelism are high in case of Workload B (95% read operations, 5% update operations). In this case, we can observe that the newer MongoDB version does not handle a high number of operations with a high degree of parallelism as well as the newer version of Cassandra does. Also, surprisingly, the older MongoDB version (3.4.4) performed better than the newer version (4.4.2) in case of Workload B, under the same experimental conditions. Generally, one would expect performance improvements from newer versions, but it is not always the case. This further shows that monitoring the evolution of performance between versions is important and worth doing.

In the future, I intend to replicate other database performance experimental studies and to analyze other database performance metrics as well. I plan to focus on analyzing the *latency* performance metric, which comes in several variants: average latency, maximum latency, minimum latency, 95th percentile latency and 99th percentile latency. As latency reveals how much time is needed to execute a database operation, it certainly is a performance metric worth analyzing for all application use cases that require very short response times.

Acknowledgments

Parts of this work were supported through the MADECIP project *Disaster Management Research Infrastructure Based on HPC*. This project was granted to Babeş-Bolyai University, its funding being provided by the Sectoral Operational Programme *Increase of Economic Competitiveness*, Priority Axis 2, co-financed by the European Union through the European Regional Development Fund *Investments in Your Future* (POSCEE COD SMIS CSNR 488061862).

References

[1] C.-F. Andor and B. Pârv. NoSQL Database Performance Benchmarking - A Case Study. *Studia Informatica*, LXIII(1):80–93, 2018.
[2] Apache Cassandra. `http://cassandra.apache.org/`. Accessed: 2021-02-14.
[3] Performance Analysis: Benchmarking a NoSQL Database on Bare-Metal and Virtualized Public Cloud - Aerospike NoSQL Database on Internap Bare Metal, Amazon EC2 and Rackspace Cloud. `http://pages.aerospike.com/rs/229-XUE-318/images/Internap_CloudSpectatorAerospike.pdf`. Accessed: 2021-03-24.
[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.
[5] Fixstars. GridDB and Cassandra Performance and Scalability. A YCSB Performance Comparison on Microsoft Azure. Technical report, Fixstars Solutions, 2016.
[6] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla. Performance Evaluation of NoSQL Databases. *EPEW 2014: Computer Performance Engineering, Lecture Notes in Computer Science*, 8721:16–29, 2014.
[7] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser. Performance Evaluation of NoSQL Databases: A Case Study. *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, pages 5–10, 2015.
[8] MongoDB. `https://www.mongodb.com/`. Accessed: 2021-02-14.
[9] NoSQLBench. `https://github.com/nosqlbench/nosqlbench`. Accessed: 2021-03-24.
[10] Stress Test for Couchbase Client and Cluster. `https://docs.couchbase.com/sdk-api/couchbase-c-client/md_doc_cbc-pillowfight.html`. Accessed: 2021-03-21.
[11] The cassandra-stress tool. `https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/tools/toolsCStress.html`. Accessed: 2021-03-21.
[12] YCSB MongoDB Performance Testing. `https://www.mongodb.com/blog/post/performance-testing-mongodb-30-part-1-throughput-improvements-measured-ycsb`. Accessed: 2021-03-24.

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: `camelia.andor@ubbcluj.ro`