STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIII**, Number 1, 2018
DOI: 10.24193/subbi.2018.1.06

NOSQL DATABASE PERFORMANCE BENCHMARKING - A CASE STUDY

CAMELIA-FLORINA ANDOR AND BAZIL PÂRV

Abstract. This paper describes an experimental study regarding NoSQL database performance. Two NoSQL databases were considered (MongoDB and Cassandra), two different workloads (update heavy and mostly read) and several degrees of parallelism. The results refer to throughput performance metric. Statistical analysis performed is referring to throughput results. Conclusions confirm that MongoDB performs better than Cassandra in the context of a mostly read workload, while Cassandra outperformed MongoDB in the context of an update heavy workload where the number of operations was high.

1. Introduction

It is hard to figure out what kind of database fits best a certain application nowadays. There are many NoSQL databases that are highly configurable and flexible, but to determine the right choice for a given application is a tedious task. NoSQL databases differ from one another on many levels, from data model to distribution model and it is not easy to make a fair performance comparison between them. Just reading the documentation of a certain NoSQL database is not enough to make sure you make the right decision for your application, but performance benchmarking gives you the opportunity to see that database in action, on your chosen hardware configuration.
In order to find out how NoSQL databases perform on a general performance benchmark, we ran performance benchmarking tests using the `YCSB` client against Cassandra and MongoDB database servers. We generated a dataset that fits in memory using the `YCSB` client and then ran benchmarking tests

Received by the editors: February 9, 2018.
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.

using various combinations of workload, number of operations and number of client threads on each database server.

2. Background

2.1. **NoSQL models.** NoSQL models appeared as a response for the need of big companies like Google or Amazon to store and manage huge amounts of data. The fact that the relational model was not built to offer horizontal scalability and the difference between in-memory data structures that are used in application programming and the relational model, known as impedance mismatch[16] are the key factors that contributed to the emergence of the NoSQL databases.

There are four main NoSQL data models: key-value, document, column-family and graph. For this paper, two data models were chosen: the document model and the column-family model. The document model and the column-family model are based on the key-value model. In a key-value database, data is stored as key-value pairs, with the key part of the pair as the unique identifier for the value that is stored in the value part of the pair. Complex values like objects or arrays can be stored as values for keys, but their structure remains invisible to the database.

In a document database, data is stored as documents. A document is similar to a key-value pair, but the difference is that the value part has a structure that is visible to the database. In a key-value database, the value part is not visible to the database. What used to be a record in a relational table becomes a document in a collection inside a document database. Still, there are some key differences between them. In a relational table, all the records have the same schema and every field can store only simple values. In a document collection, documents can have different schemas and complex values like arrays or embedded documents can be stored as values for a given field. The most popular document format is JSON[12], but there are others, like XML[21] or YAML[22]. Document databases simplify application development process. It is a lot easier to store objects as documents than to create the relational representation of an object and store it in more than one table. Also, document databases have flexible schema, so it is easier to modify it as the application evolves.

In column-family databases, data is stored as rows in column families. A column family is similar to a relational table, but it has a flexible schema. Rows in the same column family can have different columns. Each column is composed of a timestamp and a key-value pair, with the name of the column as key and the value for that column as value. Complex values like collections or arrays can be stored as values for a given column. Column-family databases are generally optimized for writes. When designing applications for column-family databases, it is a good practice to know in advance what kind of queries are needed in order to optimize read operations.

2.2. **NoSQL tools.** For each selected NoSQL model, a NoSQL database was chosen. In our benchmark, the document model is represented by MongoDB[14]. There are other document databases available on the market, like CouchDB[5] or OrientDB[15]. The column-family model is represented in our benchmark by Apache Cassandra[2]. Other column-family databases are Bigtable[3] and HBase[10].
MongoDB is an open source distributed database that was built to offer schema flexibility, horizontal scalability, replication and high availability. It has a rich query language and a good support for ad hoc queries. It was developed by 10Gen, known today as MongoDB Inc. In a MongoDB cluster there are shards or nodes that store data, config servers that store cluster metadata and query routers that route queries to the shards.
Cassandra is an open source distributed database that offers high availability, horizontal scalability and data replication, including multiple datacenter replication. It was initially developed at Facebook[13] and its data model was based on Bigtable and Dynamo[7]. In a Cassandra cluster every node is identical, which means that there is no master node and nodes can be added or removed from the cluster with no downtime.
Cassandra 3.11.0 and MongoDB 3.4.4 were the database versions installed on our servers.

2.3. **NoSQL benchmarking.** Benchmarking is very useful when evaluating NoSQL systems because it reveals the actual performance of a database on a given hardware configuration for a specific application use case. It is a difficult task to make a comparison between different NoSQL databases, and the lack of benchmarking tools for this category makes this task even harder. As a consequence of this fact, `Yahoo! Cloud Serving Benchmark`[4] (or `YCSB` for short) emerged as an open source benchmarking framework for cloud or NoSQL systems. `YCSB` was written in Java and it has two main components: the `YCSB` client, which is a workload generator and the Core workloads that represent a set of workload scenarios to be executed by the generator[23]. Both components are extensible. New workloads can be defined, so that specific application workloads can be run and database performance for those workloads can be evaluated. Other benchmarking tools are `cassandra-stress tool`[19], a tool for benchmarking Cassandra clusters and `cbc-pillowfight`[18], a tool for benchmarking Couchbase. In [6], `cbc-pillowfight` was used as a tool for workload generation, while the benchmarked database was MongoDB. Also, in a more general context, we can mention BigBench[1] tool. These tools were not used in our evaluation because (a) they cannot be used for all databases considered, and/or (b) we cannot find straight Windows implementations for them. There are benchmarking studies using YCSB discussed in the literature: [9], [8] and [11]. All these studies use a different testing environment, more precisely they employ a cloud-based infrastructure. By using virtual machines, cloud solutions are easier to manage because all the resources needed are available as Software-as-a-Service or Infrastructure-as-a-Service. Our solution, discussed in the next section, implied a big amount of work for installing and configuring all software applications needed. For our performance benchmark, we chose to use YCSB version 0.12.0 as benchmarking framework, because it is free, available, and can be used for evaluating Cassandra and MongoDB.

3. Case study

In database performance benchmarking, there are two important metrics: the throughput, measured in operations per second and the latency, measured in microseconds per operation. These two metrics are present in every test output we obtained using YCSB, but from lack of space only the throughput was analyzed in this paper.

3.1. Experimental setting.

A total of three servers having the same hardware configuration were used to run the experiment. The YCSB client ran on the first server, Apache Cassandra ran on the second server and MongoDB ran on the third server. Each server had the following hardware configuration:

- OS: Windows 7 Professional 64-bit
- CPU: Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 4 cores, 8 logical processors
- RAM: 16 GB
- HDD: 500 GB.

The data set used in our tests is composed of 4 million records and it was generated by the YCSB client. Every record has 10 fields and each field contains a random generated 100 byte string value. Because of its size, this data set could fit within memory entirely. Two YCSB core workloads were chosen: Workload A (50% update, 50% read), an update heavy workload[20] and Workload B (5% update, 95% read), a read mostly workload[20]. The number of operations parameter is in fact the number of operations performed in a test run. Each workload was tested with the following values for the number of operations: 1000, 10000, 100000 and 1000000. For every workload and number of operations combination, tests were run on 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 client threads, with every test repeated three times for

every number of client threads.
MongoDB was installed with default settings and the default storage engine for version 3.4.4, Wired Tiger.
Cassandra was installed with default settings, but in order to avoid write timeouts, we followed the setting recommendation found in [8], which is:

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

For both databases, the asynchronous Java driver was used.
A combination of workload, database and number of operations will be considered in this context a batch of tests. The database server was restarted before each execution of a batch of tests, and database server status information was captured before and after each run of batch of tests. When all combinations of tests were run for a certain workload, the data set for that workload was deleted and a data set with the same parameters corresponding to the next workload was loaded.

3.2. **Results.** Each test was repeated three times for every combination of database, workload, number of operations and number of client threads. In order to create the following charts, a throughput average was computed for every combination of database, workload, number of operations and number of threads. The first eight graphics (Figures 1 to 8) show a comparison between Cassandra and MongoDB for every combination of workload and number of operations. The last four graphics (Figures 9 to 12) show the evolution of throughput for every combination of workload and database considered in our experimental study.

Figures 1, 2, 3, and 4 show that MongoDB outperforms Cassandra when number of operations is small (1000 and 10000, respectively), for both workloads used.

Figure 5 shows that Cassandra's performance is closer to MongoDB's when the number of operations is increased to 100000, in the case of a update-heavy workload A. For the same number of operations, MongoDB still outperforms Cassandra when we use a read-heavy workload B, as Figure 6 shows.

Cassandra outperforms MongoDB only when the number of operations is 1000000 and the workload is update-heavy, as in Figure 7. For read-heavy workloads and the same number of operations, MongoDB's performance is better, as shown in Figure 8.

Figures 9 and 10 show the individual performance of the databases considered when using a heavy-update workload A, as function of the number of

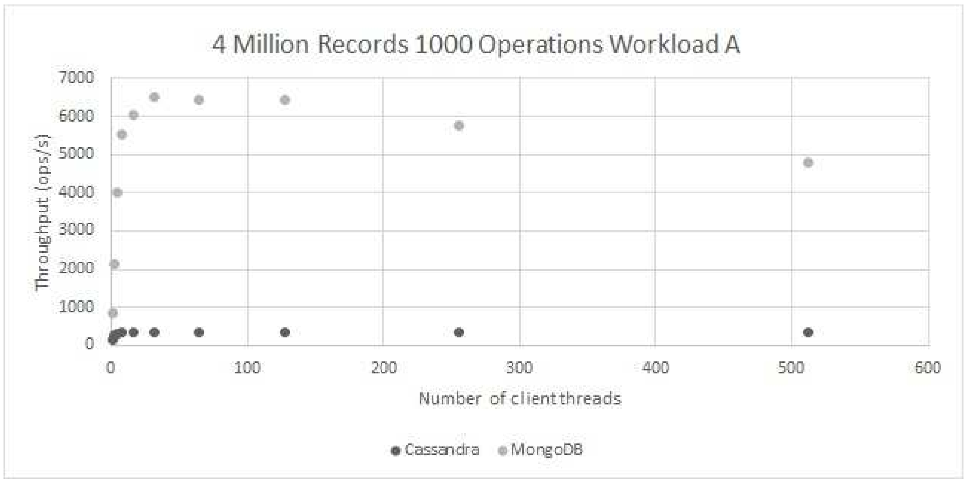

FIGURE 1. 4 Million Records 1000 Operations Workload A

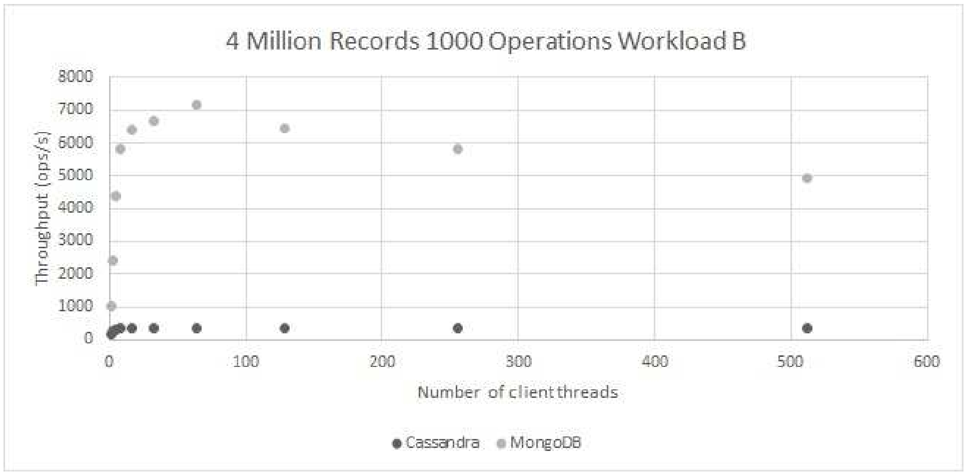

FIGURE 2. 4 Million Records 1000 Operations Workload B

threads used and number of operations involved. After the initial steep increase (128 threads for Cassandra, 32 for MongoDB), the performance flattens (with a very small decrease in the case of MongoDB). In the case of Cassandra, the performance (Figure 9) depends on the number of operations in a quasi-logarithmic fashion, while MongoDB's (Figure 10) throughput is almost the same when the number of operations is greater than or equal to 10000, with

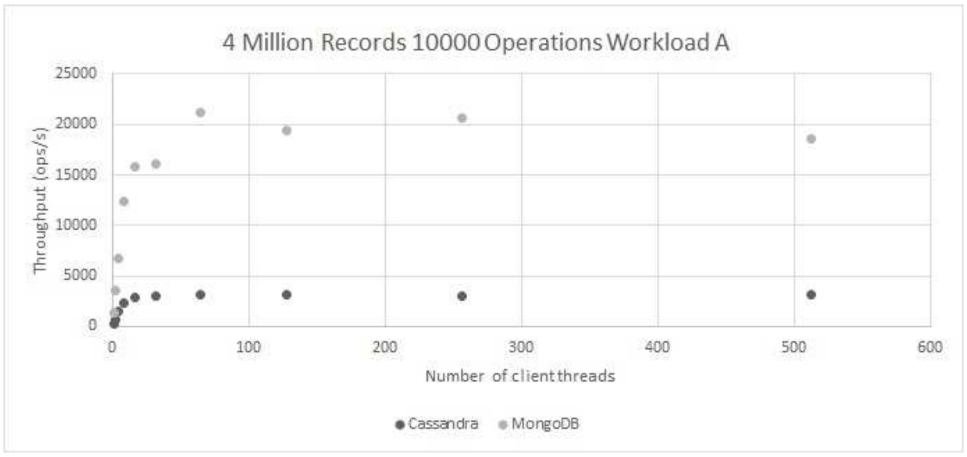

Figure 3. 4 Million Records 10000 Operations Workload A

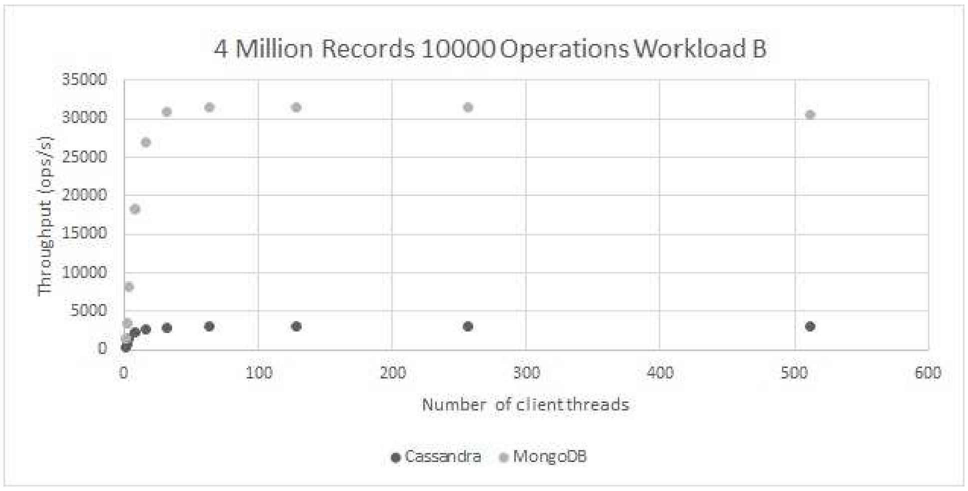

Figure 4. 4 Million Records 10000 Operations Workload B

the remark that it is slightly smaller when the number of operations increases from 100000 to 1000000.

The same comparison was performed in the Figures 11 and 12 to show the individual performance of the databases considered when using a heavy-read workload B. A first remark is that the performance decreases in the case of Cassandra (from 43000 to 30000, Figure 11) and increases in the case of MongoDB (from 23000 to 75000, Figure 12). After the initial steep increase (64

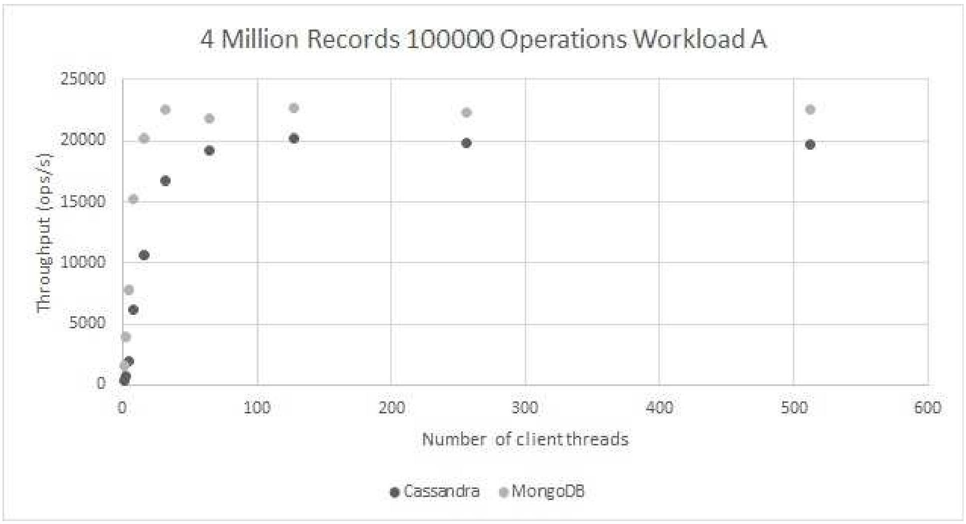

FIGURE 5. 4 Million Records 100000 Operations Workload A

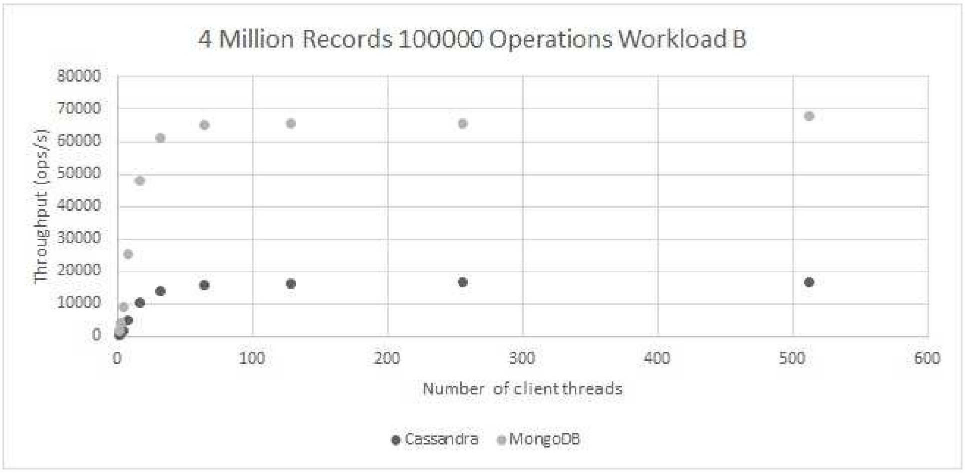

FIGURE 6. 4 Million Records 100000 Operations Workload B

threads for Cassandra, 32 for MongoDB), the performance flattens, following the same patterns.

3.3. **Statistical analysis.** Statistical analysis of the experimental results was performed using two-way ANOVA (Analysis of Variance) procedure from R Statistics Package[17]. A synthesis of the results is given in Table 1. For

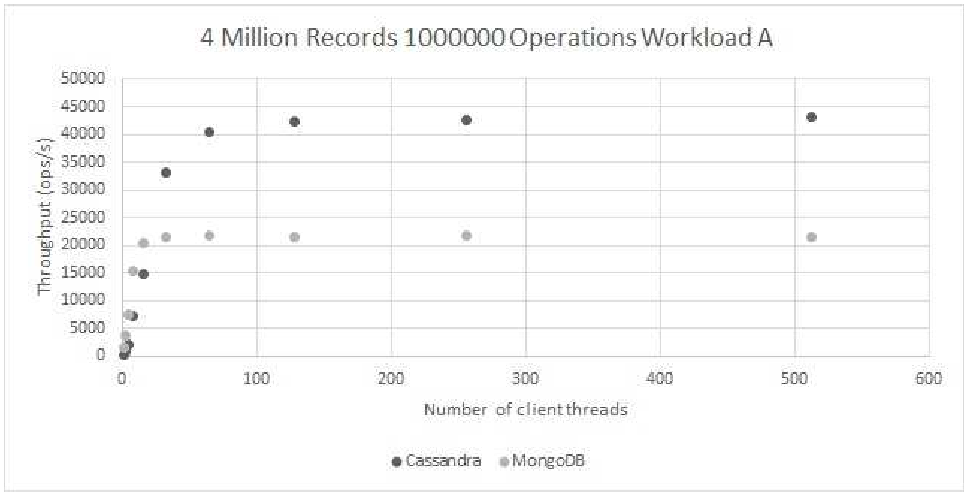

FIGURE 7. 4 Million Records 1000000 Operations Workload A

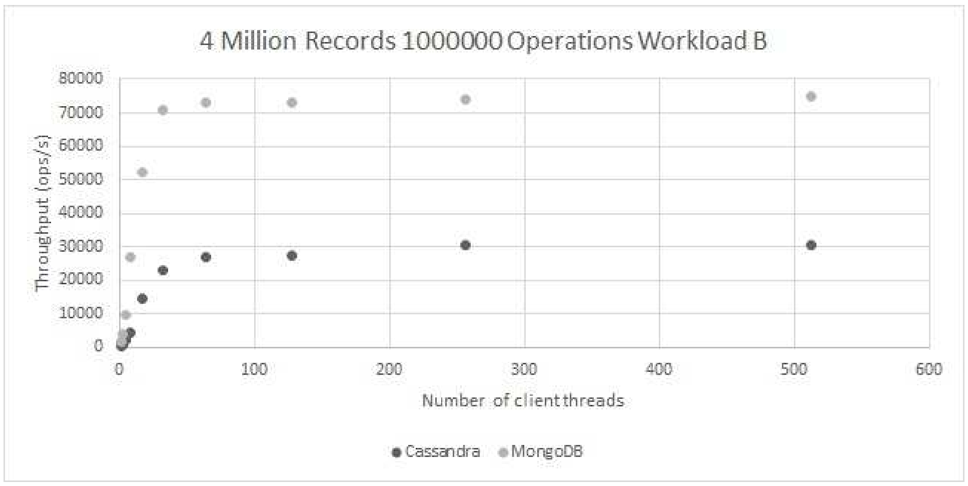

FIGURE 8. 4 Million Records 1000000 Operations Workload B

each experiment, two factors were considered: database (DB, with two levels: Cassandra and MongoDB), and the number of threads (NT, with ten levels: 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512). The interactions between DB and NT were also considered. The column labeled "Sgf" is referring to the P-value and describes textually the level of significance, 0.1%, 1%, 5%, and 10%, according to the following conventions: 0 $***$ 0.001 $**$ 0.01 $*$ 0.05 . 0.1 (*blank*) 1. In

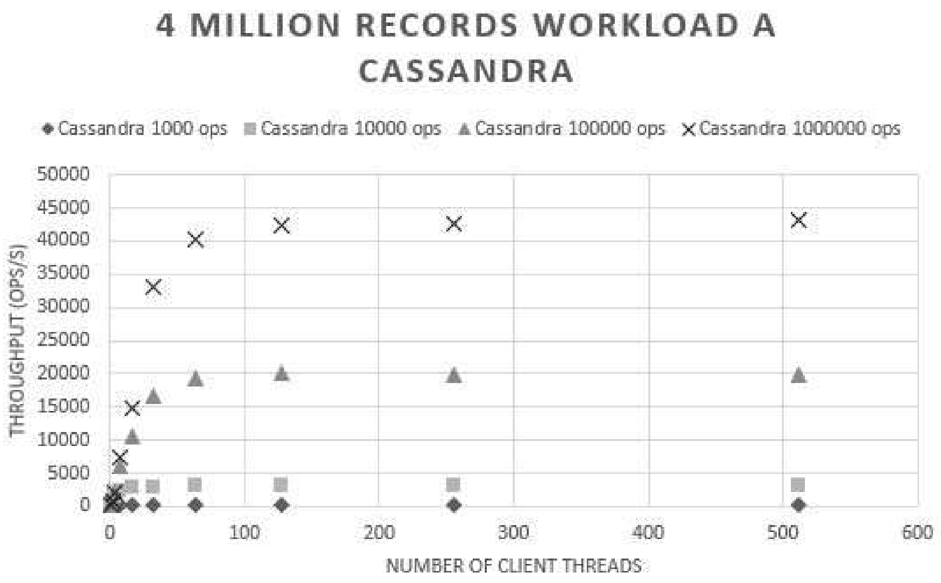

Figure 9. 4 Million Records Cassandra Workload A

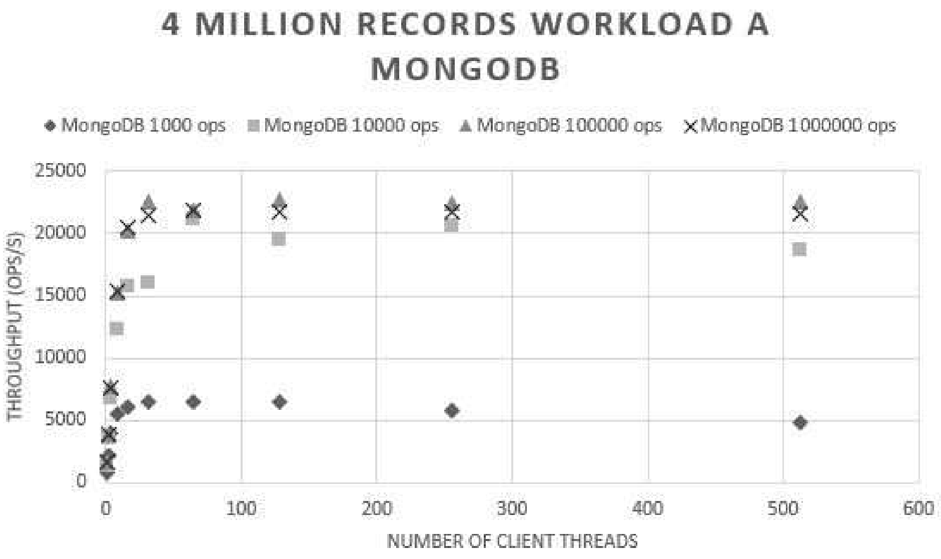

Figure 10. 4 Million Records MongoDB Workload A

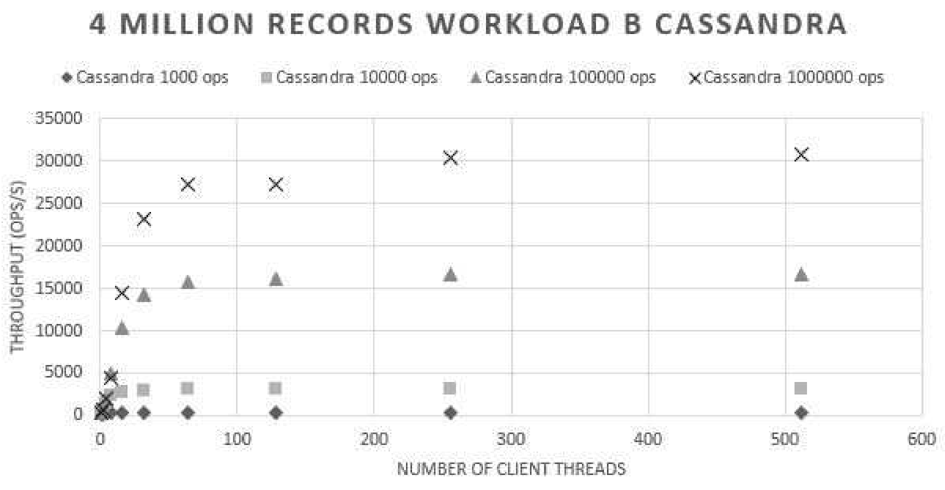

Figure 11. 4 Million Records Cassandra Workload B

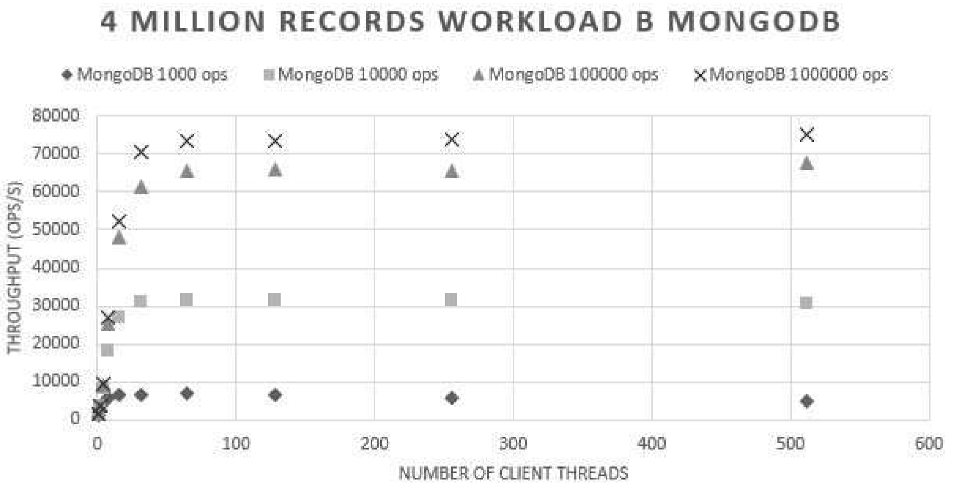

Figure 12. 4 Million Records MongoDB Workload B

other words, if P-value is $\leq 0.1\%$ (i.e. $***$ according to the legend), it means that the differences between means have a strongest statistical significance, while a P-value greater than 10% (i.e. blank space) shows that the differences between the means of the levels considered are within the experimental error.

TABLE 1. Analysis of variance - results

Wrk	No	Database			No of threads			DB:NT		
ld	ops	F-value	Pr(>F)	Sgf	F-value	Pr(>F)	Sgf	F-value	Pr(>F)	Sgf
A	1000	162.4446	<2E-16	***	1.1394	0.2904		0.9268	0.3398	
A	10000	94.3802	1.29E-13	***	12.521	0.0008174	***	7.1367	0.0098707	**
A	100000	6.3535	0.01459	*	26.268	3.82E-06	***	0.3309	0.56742	
A	1000000	5.9014	0.018362	*	31.2701	6.94E-07	***	8.7875	0.004449	**
B	1000	178.571	<2E-16	***	0.75	0.3902		0.5777	0.4504	
B	10000	96.271	9.06E-14	***	11.05	0.001568	**	7.963	0.006596	**
B	100000	56.322	5.07E-10	***	22.632	1.42E-05	***	7.61	0.007827	**
B	1000000	36.373	1.35E-07	***	27.8642	2.19E-06	***	3.4366	0.06904	.

4. CONCLUSIONS AND FURTHER WORK

After the results were analyzed, it became obvious that for a read-mostly workload (Workload B), MongoDB performed much better than Cassandra. MongoDB outperformed Cassandra in every test combination where the workload parameter was set to Workload B.
For an update-heavy workload (Workload A), Cassandra outperformed MongoDB when the number of operations was increased at 1000000 (Figure 7). In this update-heavy context, MongoDB performed much better than Cassandra in the first two test scenarios where the number of operations was set to 1000 (Figure 1), respectively 10000 (Figure 3). In the third test scenario, where the number of operations was set to 100000 (Figure 5), Cassandra's performance was comparable to MongoDB's, but not greater. After the number of operations was set at 100000, MongoDB's performance stopped growing, while Cassandra's one kept growing. Due to big differences in the infrastructure (cloud-based versus on-premises) and tool and database management systems versions used for benchmarking studies, the results of our work cannot be compared with the results reported by other studies. However, it is important to notice that the general trends are preserved, as they are mentioned by the technical documentation issued by providers.
As further work, we intend to analyze the latency metric results for this experiment, to perform post-hoc ANOVA tests and to run performance benchmarking using data sets that don't fit within memory on single server and cluster configurations. We also plan to run performance benchmarking on servers that use SSDs as disk storage and to enable replication for database servers to see how it affects performance. Moreover, the other variable in our future case studies will be the benchmarking tool, i.e. we'll try to use benchmarking tools available on Linux platforms.

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] M. H. F. R. M. P. A. C. H.-A. J. Ahmad Ghazal, Tilmann Rabl. Bigbench: towards an industry standard benchmark for big data analytics. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1197–1208, 2013.

[2] Apache cassandra. `http://cassandra.apache.org/`. Accessed: 2017-09-25.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *OSDI '06 Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, 7, 2006.

[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] Couchdb. `http://couchdb.apache.org/`. Accessed: 2017-09-25.

[6] Datagres. Perfaccel performance benchmark:nosql database mongodb. Technical report, Datagres Technologies Inc., 2015.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, oct 2007.

[8] Fixstars. Griddb and cassandra performance and scalability. a ycsb performance comparison on microsoft azure. Technical report, Fixstars Solutions, 2016.

[9] 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.

[10] Hbase. `https://hbase.apache.org/`. Accessed: 2017-09-25.

[11] N. E. P. D. K. P. C. M. John Klein, Ian Gorton. Performance evaluation of nosql databases: A case study. *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, pages 5–10, 2015.

[12] Json. `https://www.json.org/`. Accessed: 2018-03-16.

[13] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44:35–40, 2010.

[14] Mongodb. `https://www.mongodb.com/`. Accessed: 2017-09-25.

[15] Orientdb. `http://orientdb.com/`. Accessed: 2017-09-25.

[16] M. F. Pramod J. Sadalage. *NoSQL distilled : a brief guide to the emerging world of polyglot persistence*. Addison-Wesley Professional, 2012.

[17] R statistics package. `https://www.r-project.org/`. Accessed: 2017-09-25.

[18] Stress test for couchbase client and cluster. `http://docs.couchbase.com/sdk-api/couchbase-c-client-2.4.8/md_doc_cbc-pillowfight.html`. Accessed: 2017-09-25.

[19] The cassandra-stress tool. `https://docs.datastax.com/en/cassandra/2.1/cassandra/tools/toolsCStress_t.html`. Accessed: 2017-09-25.

[20] The ycsb core workloads. `https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads`. Accessed: 2017-09-25.

[21] Xml. `https://www.w3.org/TR/2008/REC-xml-20081126/`. Accessed: 2018-03-16.

[22] Yaml. `http://yaml.org/`. Accessed: 2018-03-16.

[23] Ycsb github wiki. `https://github.com/brianfrankcooper/YCSB/wiki`. Accessed: 2017-09-25.

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

Email address: {`andorcamelia, bparv`}`@cs.ubbcluj.ro`