STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 1, 2019
DOI: 10.24193/subbi.2019.1.07

RUNTIME PERFORMANCE BENCHMARKING FOR NOSQL DATABASES

CAMELIA-FLORINA ANDOR AND BAZIL PÂRV

Abstract. An experimental study regarding the performance of NoSQL database management systems is presented in this paper. In this study, two NoSQL database management systems (MongoDB and Cassandra) were compared, and the following factors were considered: degree of parallelism and workload. Two distinct workloads (mostly read and update heavy) were used, and various numbers of client threads. The measured results refer to total runtime and they confirm that MongoDB outperforms Cassandra in almost all tests considered.

1. Introduction

The relational model was considered for a long time the default data model to use, as it is very well known and extensively used by many database professionals and companies worldwide. As the necessities of software applications evolve, the quantity of data to be stored grows and the diversity of data formats increases, which makes it harder to store and manage all that data in a relational system. The need for horizontal scalability and high availability of newer software applications were the main reasons that led to the emergence of NoSQL models, as the relational model was not a good fit in these cases. The main NoSQL data models are: key-value, column-family, document and graph. While plenty of NoSQL database management systems are offered by different providers, it is quite difficult to make a choice. Besides the fact that NoSQL database management systems are very different from one another regarding data models, query languages and distribution models, they are also highly configurable and very flexible. Therefore, a fair comparison between NoSQL database management systems is not a trivial task, as

Received by the editors: February 6, 2019.

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.

it requires in-depth knowledge about each one of them. In order to choose an appropriate NoSQL database management system for an application, the designer of the application must study thoroughly the technical documentations of several NoSQL products. Even after reading and understanding properly every aspect of each NoSQL database management system considered, it can be quite hard to make a fair comparison and take the right decision. Performance benchmarks are a good way to overcome the lack of comparison criteria between NoSQL database management systems. These benchmarks help application designers see the candidate database management system in action, allowing them to choose the appropriate hardware and software configuration. The current paper refers to total runtime metric and it is the third in a series evaluating performance metrics of two NoSQL implementations: MongoDB and Cassandra. All benchmarking tests were executed using `Yahoo! Cloud Serving Benchmark` with different combinations of number of operations, number of client threads and workload on every database server.

2. Background

2.1. **NoSQL data models.** Huge volumes of data that are generated with a high velocity, also known as big data, cannot be handled effectively with relational databases. As a response to this problem, companies that work with big data have created new data storage systems which are more flexible, non relational, distributed and highly available. Inspired by Amazon's and Google's non relational database systems, more and more companies built their own non relational implementations, specialized on their needs. These new non relational database systems are known today as NoSQL database systems, and the data models they are based on are known as NoSQL data models. These data models offer a flexible schema and the ability to store related data in a single unit with a complex structure, thus removing the need for join operations and increasing performance. The most important NoSQL data models are the graph model, the key-value model, the document model and the column-family model.

The graph model is the best fit for highly interconnected data. In this model, data are stored as nodes and relationships between nodes. Each node and relationship has a type and multiple properties, similar to a property graph. Graph queries can be very expressive and fast, but the highly interconnected nature of this model limits the horizontal scalability.

The key-value model can be considered the simplest NoSQL data model, because it stores data as key-value pairs. Key-value pairs can be grouped into containers called buckets, as stated in [19]. The key is the unique identifier

of the value part that stores the actual data. The value can have a complex structure, but it is invisible at database level, therefore queries based on value are not supported. While query capabilities are quite limited, horizontal scalability is very well supported and database operations are fast.

The document model is somehow similar to the key-value model, because a document can be considered a key-value pair. Each document has a unique identifier and a value with a complex structure that is visible at the database level. Documents are organized in collections. Queries can be very expressive and horizontal scalability is easily supported, as there are no relationships defined between documents. In a document, objects or arrays are allowed to be stored as values for fields. Unlike the relational model, which imposes that all records stored in a table must have the same structure, the document model allows documents that have different structures to be stored in the same collection. One of the most used document formats is JSON[13], followed by XML[23] and YAML[24]. Related data can be grouped and stored in a single document that is similar to the object representation used at the application level, therefore the problem known as impedance mismatch[19] is removed and the application development process is significantly improved.

The model known as column-family organizes data as rows that belong to column families. There are some similarities between a relational table and a column family, but the latter allows arrays, lists or objects to be stored as values for columns and rows with different columns can be part of the same column family. A column is in fact a key-value pair stored together with a timestamp, and the name of the column represents the key part of the pair. Join operations are usually not supported by this model, and denormalization is common. Write operations are quite fast. A disadvantage of this model is that query capabilities are somehow limited and the most common queries must be taken into consideration in the design phase. Otherwise, horizontal scalability and high availability are easily supported.

2.2. **NoSQL tools.** The data models chosen for our benchmarking study are column-family and document. For each data model we chose a representative implementation: Apache Cassandra[1] as the column-family implementation, and MongoDB[16] as the document implementation. The column-family model is also implemented by HBase[11], Bigtable[3] and Hypertable[12]. For the document model, alternative implementations are: BaseX[2], CouchDB[6] and Couchbase[5].

Apache Cassandra is an open source column-family DBMS that was originally developed at Facebook[15]. Its data and distribution models are based on those used by Bigtable[3] and Dynamo[7]. Cassandra was designed to run in a distributed environment and has its own query language, called

CQL(Cassandra Query Language). The main features offered by Cassandra are automatic sharding, high availability, tunable consistency and data replication, including data replication across multiple data centers. In a Cassandra cluster, all nodes are equal and each one of them can accept both read and write operations. New nodes can be added to a cluster without downtime, and the cluster capacity for both read and write operations scales linearly. The version used in our study was Apache Cassandra 3.11.0.

MongoDB is an open source document DBMS developed by MongoDB Inc. MongoDB stores data as JSON-like documents and does not require a predefined schema. It has a rich query language based on JavaScript and ad hoc queries are well supported. The main features of MongoDB are automatic sharding, high availability, data replication (including multiple data center replication), tunable consistency and schema flexibility. MongoDB was designed to run in a distributed environment. In a MongoDB cluster, there are three types of nodes: shards, query routers and configuration servers. Shards store the actual data, while the cluster metadata is stored on the configuration servers. Query routers use cluster metadata stored on configuration servers to route queries to the corresponding shards. The version used in our study was MongoDB 3.4.4.

2.3. **NoSQL benchmarking.** When many NoSQL database management systems are available on the market, it can be hard to choose the right product that offers the best performance for a particular application use case on a specific hardware configuration. Benchmarking is a good approach in this case, as it measures the actual performance of a NoSQL implementation on a given hardware configuration. But the benchmarking process also requires software tools, and there are not so many open source options available.

A benchmarking tool has two main features: workload generation and performance measurement. A batch that contains all requests sent by an application to a database server during a working session represents the application's workload. The main metrics in database performance benchmarking are throughput, total runtime and latency. The number of operations completed in a time unit is known as throughput. The amount of time needed for a single operation to be completed is known as latency. The amount of time needed to complete a given number of operations is known as total runtime (RT). Throughput is measured in operations per second, while latency is measured in microseconds per operation. Higher throughput values are better from the performance viewpoint. Total runtime is measured in milliseconds and it represents the duration of a benchmarking test. Regarding performance, lower total runtime and lower latency values are better. This paper refers to total runtime.

Two types of NoSQL database benchmarking tools can be used: database-independent and database-specific. From the database-specific type we can remark `cbc-pillowfight`[20] for Couchbase and `cassandra-stress tool`[21] for Cassandra. Our study aims to compare two different NoSQL database servers by applying the same workload, so database-specific tools cannot be used. From the database-independent type we mention `BigBench`[10] and `YCSB`[4]. While `BigBench` runs only on Linux, `YCSB` runs on Linux and Windows. Our case study involves servers that have Windows operating system installed, which implies that `BigBench` cannot be utilized as benchmarking framework. Also, the resemblance between `BigBench` and TPC-DS[17] makes it less flexible and more oriented on traditional workloads instead of NoSQL workloads. On the other hand, `YCSB` focuses on big data and NoSQL workloads and offers a lot of flexibility in both workload definition and workload configuration. The fact that `YCSB` can be used to test many NoSQL DBMSs, including Cassandra and MongoDB, is another important aspect that makes it suitable for our study.

`Yahoo! Cloud Serving Benchmark`[4] (`YCSB`) appeared as a response to the necessity of a benchmarking tool that is suitable for cloud or NoSQL systems. It is an open source project developed initially at Yahoo! and written in Java which has two base components that are extensible. The first component is the workload generator known as the `YCSB` client. The second component, known as the Core workloads, consists of a set of workload scenarios that have to be executed by the `YCSB` client, as stated in [25]. Each workload used in `YCSB` has two main parts: a data set and a transaction set. The total number of records that need to be loaded into the database before any test is performed represents the data set. The mix of write and read operations to be performed in a test represents the transaction set. The main parameters of the transaction set are: the total number of operations to be applied in a test execution, the number of client threads and the ratio between write and read operations. If the workloads contained in the Core workloads set are not suitable for the needs of the user, new custom workloads can be created. In our benchmarking study, we used `YCSB` version 0.12.0.

`YCSB` is also used in other benchmarking studies that are discussed in the literature: [14], [9] and [8]. These benchmarking studies use a cloud-based infrastructure, while our benchmarking study uses physical machines and it is not cloud-based. The DBMS versions used in our study are newer than those used in [14], [8] and [9]. Also, the operating system installed on our servers is Windows. Other significant differences refer to the data sets used and their size, hardware configuration and workload types. Paper [14] presents a benchmarking study that involves custom workloads and a proprietary data set

belonging to a healthcare organization. The benchmarking studies presented in [9] and [8] use data sets generated by the `YCSB` client. In [9], the size of the data sets is not clearly stated, while the study discussed in [8] does not include MongoDB, even if it specifies the actual size of the data sets used.

3. Case study

Our benchmarking experiment involved three servers with the same hardware configuration. A different application ran on each server: `YCSB` client on the first server, Cassandra on the second server and MongoDB on the last server. The server configuration is as follows:

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

The data set utilized in our experiment was generated by the `YCSB` client. It contains 4 million records and it was used with all workloads. Each record contains 10 fields and every field holds a 100 byte string value generated randomly. The data set could fit in the internal memory due to its size. Two predefined `YCSB` workloads were chosen: Workload A (50% reads, 50% updates), which is considered an *update-heavy* workload[22], and Workload B (95% reads, 5% updates), that is considered a *read-mostly* workload[22]. An example of application for Workload A could be a session store that records recent actions, while photo tagging could be a corresponding example for Workload B, as stated in [22]. Both workloads were tested with the following number of operations: 1000, 10000, 100000 and 1000000. For each number of operations, tests were executed using 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 client threads. Each test with a certain combination of workload, number of operations and number of client threads was repeated three times.
We installed MongoDB with default settings. The default storage engine for MongoDB version 3.4.4 is Wired Tiger. We also installed Apache Cassandra with default settings, but in order to avoid timeouts, we followed the setting recommendation mentioned in [8]:

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

The asynchronous variant of Java driver was used for both database servers. A batch of tests includes all tests having the same workload, number of operations, and database server, but different number of client threads. We

TABLE 1. Runtime Results for 1000 Operations Workload A

NT	Cassandra	MongoDB
1	6085.666667	1232.666667
2	3702.333333	468
4	3088.666667	249.6666667
8	2959	182
16	2938.333333	166.6666667
32	2880.666667	156
64	2896.333333	157.6666667
128	2855	156
256	2865	174
512	2906.666667	209.3333333

TABLE 2. Runtime Results for 10000 Operations Workload A

NT	Cassandra	MongoDB
1	34757	7124
2	16884.33333	2792
4	6708	1477
8	4279.666667	811.3333333
16	3535.666667	634.3333333
32	3364.333333	623.6666667
64	3161.333333	478.6666667
128	3187.333333	519.6666667
256	3276	494
512	3229.333333	546.3333333

restarted the database server before each batch of tests was executed, and we captured database server status information before and after each execution of a batch of tests. When the execution of all combinations of tests for the first workload was finished, the data set corresponding to that workload was dropped. After that, the data set characterized by the same parameters that corresponds to the second workload was loaded.

3.1. **Results.** Every test was repeated three times for each combination of workload, database, number of operations and number of client threads. A total runtime (RT) average was calculated for each combination of workload, database, number of operations and number of threads (NT) in order to create the following charts. Figures 1 to 8 show a comparison of RT performance between Cassandra and MongoDB for each combination of workload and number of operations. It is worth to mention here that the cases NT = 1 and NT = 2 are not shown in figures because they produce by far greater RT values than the other cases considered. The runtime results are also presented in tables 1, 2, 3, 4, 5, 6, 7 and 8, including the cases NT = 1 and NT = 2.

Figures 1, 2, 5 and 6 show that in the case of a small number of operations (1000 and 10000, respectively), MongoDB outperforms Cassandra for both workloads used and all NT levels considered.

Figure 3 shows that the performance of Cassandra closes to MongoDB's in the case of an update-heavy workload A and for NT $\geq$ 64, when the number of operations is set to 100000. For the same number of operations, MongoDB produces better results than Cassandra when we use a read-heavy workload B, as shown in Figure 7.

When the number of operations is 1000000, the results differ: Cassandra outperforms MongoDB (as in Figure 4) when workload is update-heavy and NT $\geq$ 32, while MongoDB's performance is better for a read-heavy workload, as shown in Figure 8.

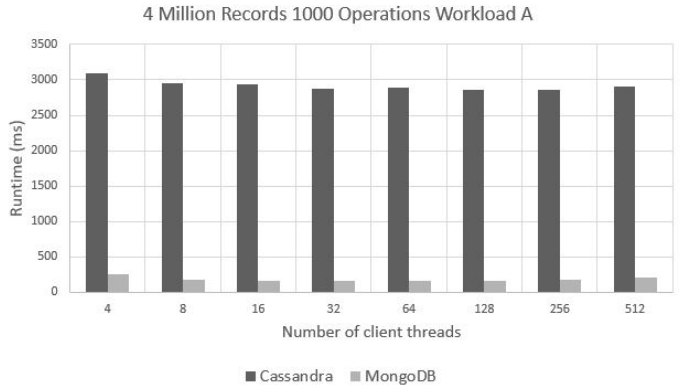

FIGURE 1. 4 Million Records 1000 Operations Workload A - Runtime

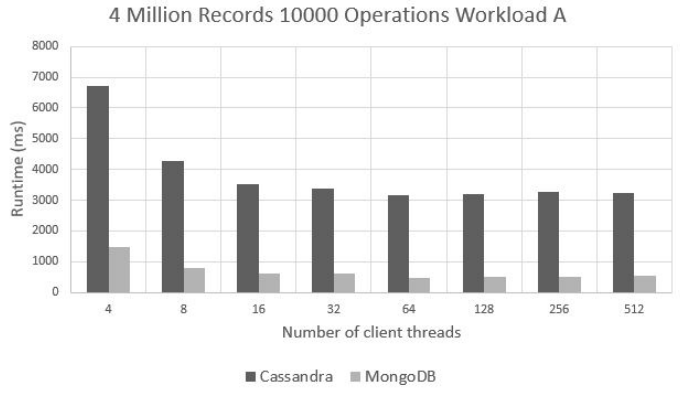

FIGURE 2. 4 Million Records 10000 Operations Workload A - Runtime

TABLE 3. Runtime Results for 100000 Operations Workload A

NT	Cassandra	MongoDB
1	303900	62698.66667
2	144909	25791.66667
4	50164.33333	12912
8	16301.66667	6557.333333
16	9344.333333	4950.666667
32	5954	4435.666667
64	5194.666667	4591.666667
128	4950.333333	4415
256	5033.666667	4477
512	5064.666667	4441

TABLE 4. Runtime Results for 1000000 Operations Workload A

NT	Cassandra	MongoDB
1	2995324.667	629249
2	1454355	259615.6667
4	501797	131332.6667
8	137265.3333	64933
16	67153.33333	48817.66667
32	30202	46567
64	24820	45854
128	23609	46166.33333
256	23431.66667	46108.66667
512	23197.66667	46254.66667

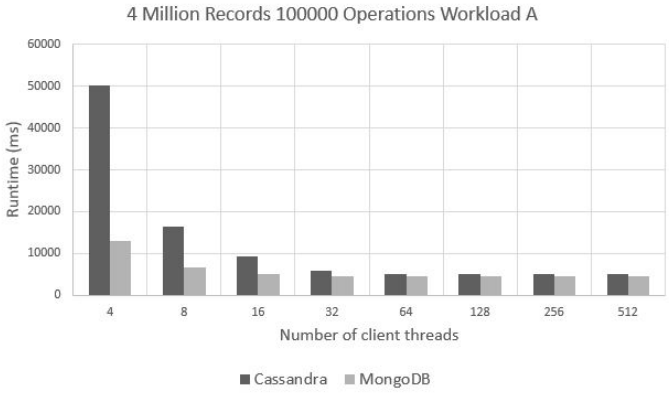

FIGURE 3. 4 Million Records 100000 Operations Workload A - Runtime

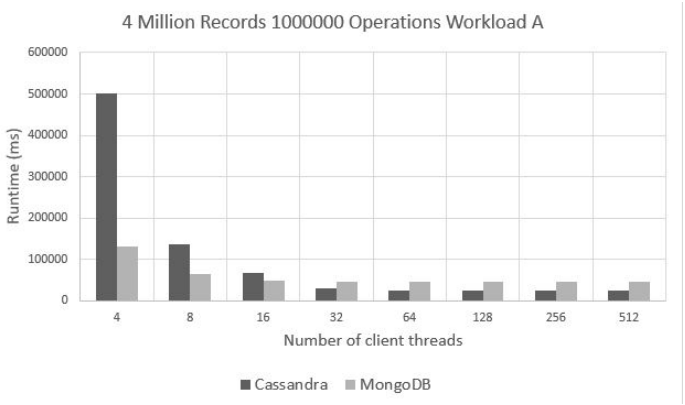

FIGURE 4. 4 Million Records 1000000 Operations Workload A - Runtime

TABLE 5. Runtime Results for 1000 Operations Workload B

NT	Cassandra	MongoDB
1	6635	1014
2	3832.333333	415.6666667
4	3104.333333	228.6666667
8	2943	171.3333333
16	2906.666667	156
32	2870.333333	151.3333333
64	2865	140.6666667
128	2875.333333	156
256	2875.666667	171.6666667
512	2922.333333	203

TABLE 6. Runtime Results for 10000 Operations Workload B

NT	Cassandra	MongoDB
1	35765.66667	6391
2	13884	2870.333333
4	6541.666667	1227.666667
8	4186.333333	546.3333333
16	3567	369.6666667
32	3317.666667	322.6666667
64	3219	317
128	3224	317.3333333
256	3224.333333	317.3333333
512	3244.666667	327.6666667

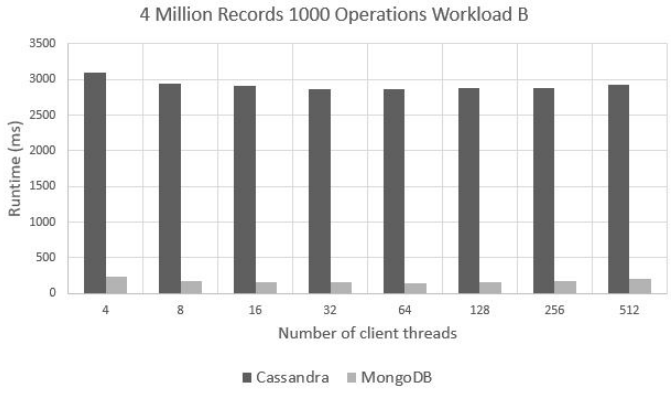

FIGURE 5. 4 Million Records 1000 Operations Workload B - Runtime

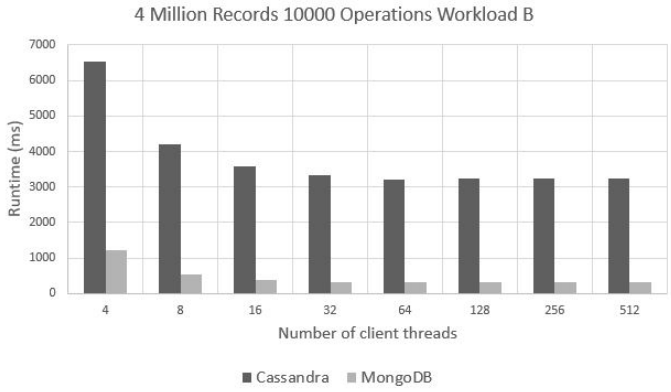

FIGURE 6. 4 Million Records 10000 Operations Workload B - Runtime

TABLE 7. Runtime Results for 100000 Operations Workload B

NT	Cassandra	MongoDB
1	309530.3333	57995.66667
2	134706.3333	26020.66667
4	52171.66667	11117.33333
8	19978.33333	3957.333333
16	9651.333333	2085
32	7051.333333	1632.666667
64	6354.333333	1528.666667
128	6188.333333	1518.333333
256	5964.333333	1523
512	5974.666667	1476.666667

TABLE 8. Runtime Results for 1000000 Operations Workload B

NT	Cassandra	MongoDB
1	3001223	614303
2	1306186	269886
4	500798.3333	105114
8	227412.3333	37201
16	68801	19208.66667
32	43238.66667	14154.33333
64	36936.33333	13645
128	36754	13645
256	32880	13509.66667
512	32579	13338.33333

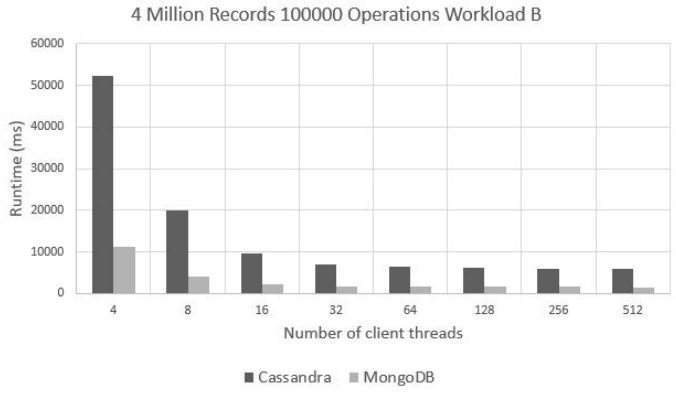

FIGURE 7. 4 Million Records 100000 Operations Workload B - Runtime

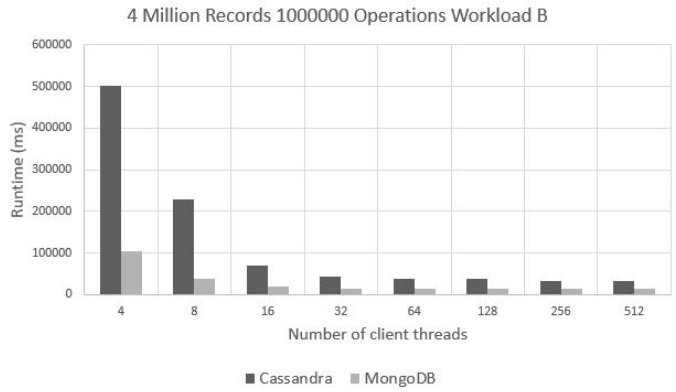

FIGURE 8. 4 Million Records 1000000 Operations Workload B - Runtime

3.2. **Statistical analysis.** Experimental data given in tables 1 thru 8 were processed using two-way ANOVA (Analysis of Variance) procedure from R Statistics Package[18]. Table 9 displays a synthesis of the results. The two factors considered for each experiment are: 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 studied. The column named "Sgf" (abbreviation for statistical significance) refers to the P-value, denoting the level of significance, 0.1%, 1%, 5%, and 10%, following the usual conventions: $0 \ *** \ 0.001 \ ** \ 0.01 \ * \ 0.05 \ . \ 0.1 \ (blank) \ 1$. In other words, if a P-value is $\leq 0.1\%$ (that is, $***$ according to the legend), it means that the differences between means of the factors considered have the

TABLE 9. Analysis of variance - Runtime Results

Wrk ld	No ops	Database			No of threads			DB:NT		
		F-value	Pr(>F)	Sgf	F-value	Pr(>F)	Sgf	F-value	Pr(>F)	Sgf
A	1000	269.6848	<2e-16	***	3.8486	0.05476	.	1.1328	0.29174	
A	10000	14.3015	0.0003805	***	4.7928	0.0327625	*	2.0936	0.1534923	
A	100000	6.1596	0.01610	*	5.1321	0.02737	*	2.4626	0.12222	
A	1000000	5.5011	0.02257	*	5.2303	0.02600	*	2.5335	0.11708	
B	1000	218.2569	<2e-16	***	3.1952	0.07927	.	1.4045	0.24098	
B	10000	14.1485	0.0004059	***	4.2981	0.0427683	*	1.8757	0.1762899	
B	100000	7.1493	0.009809	**	5.1082	0.027720	*	2.3786	0.128642	
B	1000000	6.4783	0.01370	*	5.3822	0.02401	*	2.4108	0.12614	

strongest statistical significance. The other end of spectrum, when a a P-value is greater than 10% (a blank space) shows that the differences between the means of the factors considered are within the range of experimental error.

When comparing RT averages for DB factor, Table 9 shows that there are several degrees of statistical significance between them in all combinations (workload, number of operations) considered. The same table shows less stronger statistical significance when comparing RT averages for NT factor, and even poorer one in the case of 1000 operations. The interactions DB:NT have no statistical impact on the RT.

4. Conclusion

In our benchmarking study, the performance of the two NoSQL database management systems was measured for two workloads: read-mostly (Workload B) and update-heavy (Workload A). The performance indicator was total runtime (RT). MongoDB outperforms Cassandra in all studies involving Workload B. For Workload A, the situation is the same, with some exceptions: the cases where the number of operations is equal to 1000000 and the number of client threads is greater than or equal to 32.

As further work, we plan to perform other experimental studies using data sets with different number of fields on single server and cluster configurations. Also, we intend to test other workload configurations with data sets that exceed the internal memory. Another direction in our experimental work will deal with database server replication and SSDs as disk storage, in order to measure the performance impact of these configurations. Lastly, the operating system will be another variable in our future case studies.

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] Apache Cassandra. `http://cassandra.apache.org/`. Accessed: 2017-09-25.
[2] BaseX. `http://basex.org/`. Accessed: 2018-11-27.
[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] Couchbase. `https://www.couchbase.com/`. Accessed: 2019-01-22.
[6] CouchDB. `http://couchdb.apache.org/`. Accessed: 2017-09-25.
[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 21st 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] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. 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.
[11] HBase. `https://hbase.apache.org/`. Accessed: 2017-09-25.
[12] Hypertable. `http://www.hypertable.org/`. Accessed: 2018-11-27.
[13] JSON. `https://www.json.org/`. Accessed: 2018-03-16.
[14] 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.
[15] A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44:35–40, 2010.
[16] MongoDB. `https://www.mongodb.com/`. Accessed: 2017-09-25.
[17] R. O. Nambiar and M. Poess. The Making of TPC-DS. *VLDB '06 Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 1049–1058, 2006.
[18] R Statistics Package. `https://www.r-project.org/`. Accessed: 2017-09-25.
[19] P. J. Sadalage and M. Fowler. *NoSQL distilled : a brief guide to the emerging world of polyglot persistence*. Addison-Wesley Professional, 2012.
[20] 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: 2019-01-03.
[21] The cassandra-stress tool. `https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsCStress.html`. Accessed: 2019-01-03.

[22] The YCSB Core Workloads. https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads. Accessed: 2017-09-25.
[23] XML. https://www.w3.org/TR/2008/REC-xml-20081126/. Accessed: 2018-03-16.
[24] YAML. http://yaml.org/. Accessed: 2018-03-16.
[25] YCSB Github Wiki. https://github.com/brianfrankcooper/YCSB/wiki. Accessed: 2017-09-25.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 Kogălniceanu, Cluj-Napoca, 400084, Romania
Email address: {andorcamelia, bparv}@cs.ubbcluj.ro