STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 1, 2019
DOI: 10.24193/subbi.2019.1.04

USING LATENCY METRICS IN NOSQL DATABASE PERFORMANCE BENCHMARKING

CAMELIA-FLORINA ANDOR, BAZIL PÂRV, AND DAN MIRCEA SUCIU

Abstract. This paper presents an experimental study evaluating the performance of NoSQL database management systems. The study compares two NoSQL database management systems (Cassandra and MongoDB) and considers the following parameters/factors: workload and degree of parallelism. Two different workloads (update heavy and mostly read) were used, and different numbers of threads. The measured results are related to average latency: update latency and read latency. Our study shows that with the only exception of 1000 operations, both latency indicators have a quasi-parabolic behavior, where the minimum (i.e. the best performance) depends mainly on the number of threads and slightly varies with the increase in the number of operations. In the case of 1000 operations, there is also a maximum point (i.e. worst performance) case, after which the latency decreases.

1. Introduction

NoSQL data models appeared from practical reasons, some industrial solutions becoming de facto standard solutions in the cases where the relational data model failed to provide acceptable performance in terms of horizontal scalability and high availability. A lot of different data models (key-value, document, column-family, graph, etc.) are collectively known today as NoSQL data models. Different providers offer today a large selection of highly configurable and flexible NoSQL database management systems. They differ in terms of the data models and distribution models they implement, as well as the real problems appropriate for their use. Consequently, it is very difficult to compare them. For the designer of an application using such a NoSQL

Received by the editors: January 8, 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.

database management system it is not enough to read the technical documentation in order to make the best design decisions. Instead, the usual approach is to use some performance benchmarks, which helps to see the actual behavior of the database and to choose the appropriate hardware configuration. This paper is the second describing a series of experimental studies, evaluating different performance indicators of two NoSQL database management systems: Cassandra and MongoDB. The current paper refers to two average latency measures, update latency and read latency, while the first one, [1], discussed the throughput averages. Latency indicators characterize the time needeed to perform a single operation/request (amount of time the request takes to complete), and throughput is related to the computational performance, measured in number of operations/requests performed per second. Both performance indicators have an equal importance: latency for describing the response time, and throughput for the server performance. The benchmarking tests for experimental studies were performed using the `Yahoo! Cloud Serving Benchmark` client, using varied pairs of number of operations, number of threads and workload on every database server.

2. Background

2.1. **NoSQL Data Models.** Large companies like Amazon and Google introduced NoSQL data models to collect and manage large quantities of data in a distributed environment: Google Bigtable[3] and Amazon Dynamo[6]. Their success gave a new research field - NoSQL data models. In the order of their complexity, they are: key-value, document, column-family and graph. There are also combinations/variations of these basic models.

In the key-value model, the key part uniquely identifies the value part, which is not visible to the database (you cannot directly perform queries on values). Key-value databases organize data as key-value pairs, allowing arrays or objects to be stored as values for keys, but their structure is not exposed at the database level. Amazon Dynamo uses this model.

The column-family and document models are based on the key-value model. The document model allows you to query the value: a document resembles a record that belongs to a relational table. Document DBMSs organize documents in collections. Unlike the relational model, in which all the records have the same schema and a field can store only simple values, here the documents can have different schemas and their fields can store complex values like arrays or embedded documents. The most popular document formats are JSON[11], XML[21], and YAML[22]. Document DBMSs simplify the application development process, as the document's structure is similar to that of an object used at the application level.

In the column-family model, data is organized as rows that belong to column families. A column family is like a relational table, but has a flexible schema. Each column contains a key-value pair and a timestamp. The key is in fact the name of the column and the value is the column itself. The value of a column can be complex, like a collection or a tuple. Rows that contain different columns are allowed to be part of the same column family. A good practice in application development that involves the use of column-family DBMSs is to know the queries in advance, in order to optimize the database schema around those queries. Column-family DBMSs are generally optimized for write operations. Google Bigtable was the first implementation of this data model.

The graph model is the most complex, being appropriate for heavy interconnected data. In such a property graph, both nodes and edges have properties. If data are heavy interconnected, this affects the horizontal scalability, because it is difficult to decide where to split the graph into several sub-graphs stored into a distributed environment.

2.2. **NoSQL tools.** For our benchmarking study, two data models were considered: the document model (implemented by MongoDB[14]) and the column-family model (implemented by Cassandra[2]). Among other DBMSs implementing these models and available on the market we mention CouchDB[5], OrientDB[16] for the document model, and Bigtable[3], HBase[10] for the column-family model. OrientDB also supports other data models.

MongoDB is an open source distributed database developed by 10Gen, known today as MongoDB Inc. Its main features are: horizontal scalability, flexible schema, high availability, replication and an expressive query language. Ad hoc queries are very well supported. A MongoDB cluster is made up of shards, configuration servers and query routers. Shards or nodes are used to store data, config servers store cluster metadata, while query routers route queries to the shards.

The other open source distributed database considered is Cassandra, initially developed at Facebook[13]. Its main features are: high availability, data replication and horizontal scalability. Unlike MongoDB, which performs better in the case of read operations, its core query language is not so rich, and it offers better support for write operations. Cassandra's data model is column-family, based on Bigtable[3] and Dynamo[6]. A Cassandra cluster is made up of identical servers, which means each node can accept both read and write operations. Also, there is no downtime when adding or removing a node from the cluster.

In our case study, Apache Cassandra 3.11.0 and MongoDB 3.4.4 versions were installed on our servers.

2.3. **NoSQL benchmarking.** If many options (i.e. in our case many open source NoSQL database management systems) are available to the application developer, it is a difficult decision to choose the right one, which gives the best performance on a given hardware configuration and for a specific application use case. In these situations, where it is difficult to make comparisons between the performance of different NoSQL database servers, benchmarking is very useful. As it is the case in other similar situations, the benchmarking process needs its own tools, and there are not so many options in the open source movement.

The functionality of a benchmarking tool covers two different areas: workload generation and performance measurement using different workload scenarios. A workload is the load that is put by a certain application on the database management system, i.e. a batch of all requests a given application is sending to the server during a working session. For benchmarking purposes, the workload definition needs some clarifications, as we'll see below.

In database performance benchmarking, there are three important metrics: throughput, measured in operations per second, latency, measured in microseconds per operation, and total runtime, also measured in milliseconds. Throughput measures the number of operations per time unit (second), while latency measures the duration of a single operation (expressed in microseconds). The total runtime expresses the entire duration of a test. Higher throughput and lower latency and lower total runtime values are better from the performance viewpoint.

There are two categories of NoSQL database benchmarking tools: database-specific and database-independent. In the database-specific category we can mention `cassandra-stress tool`[19] for Cassandra, and `cbc-pillowfight`[18] for Couchbase. These tools cannot be used in our study, which aims to compare performances of different databases on the same workload. In the database-independent category we mention `YCSB`[4] and `BigBench`[9]. `YCSB` runs on both Windows and Linux, while `BigBench` runs only on Linux. Moreover, `YCSB` offers more flexibility than `BigBench` in terms of data models and workload configurations. The fact that `BigBench` resembles `TPC-DS`[15] makes it less oriented on big data or NoSQL workloads, while `YCSB` focuses on NoSQL systems and their specific workload types. Also, `YCSB` can be used to test the performance of many NoSQL DBMSs, including MongoDB and Cassandra, which makes it a good choice for our study.

`Yahoo! Cloud Serving Benchmark`[4] (or `YCSB`) is an open source benchmarking framework for NoSQL and cloud systems. It was developed in Java and includes two main components: the workload generator known as the

`YCSB` client, and the Core workloads that represent a set of workload scenarios to be executed by the generator[23]. These components can be extended. In the context of `YCSB`, a workload has two components: a data set and a transaction set. The *data set* represents the set of records that are loaded into the database before any transaction is performed on it. The *transaction set* contains all read and write operations to be run on the database server. Main parameters of the transaction set are: the number of operations in it (i.e. the number of operations to be performed in a test run), the ratio between read and write operations, and the number of client threads.

Besides Core workloads, `YCSB` allows the user to define new workloads. For our performance benchmark, `YCSB` version 0.12.0 was used as benchmarking framework. In the literature, there are other benchmarking studies that use `YCSB`: [7], [8] and [12]. These studies employ a testing environment that involves a cloud-based infrastructure, which greatly differs of our approach. Our testing environment is not cloud-based, and it uses Windows operating system on both client and server machines. Other differences refer to the use of physical machines instead of virtual ones, DBMS versions, operating system, workload types, the size of the data sets, and hardware configurations. The case study presented in [12] uses a proprietary data set that belongs to a healthcare organization and custom workloads. The data sets used in [7] and [8] are generated by the `YCSB` client, but the actual size of the data sets used in [8] is not clearly specified. The study presented in [7] specifies the size of the data sets used, but it does not include MongoDB in the study, it only includes Datastax's variant of Cassandra. Also, the software versions of the DBMSs and `YCSB` used in [7], [8] and [12] are older than those used in our study and, especially in the case of MongoDB, the difference between version 2 and version 3 is very significant. Starting with version 3, MongoDB uses a different storage engine, called Wired Tiger, which greatly improves performance.

3. Case study

3.1. **Experimental setting.**

Our experiment used a total of three servers having the same hardware configuration, each of them running a different application: `YCSB` client on the first server, Apache Cassandra on the second one and MongoDB on the third. The configuration of each server is as follows:

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

All workloads used the same data set, composed of 4 million records. The `YCSB` client was used to generate the data set. Each record contains 10 fields and every field stores a 100 byte string value that was randomly generated. The data set could fit within internal memory due to its size. The chosen workloads belong to the `YCSB` Core workloads set: Workload A (the ratio between read and write operations in the transaction set is 1 - 50% updates, 50% reads), an *update-heavy* workload[20], and Workload B (the ratio between read and write operations in the transaction set is $\frac{1}{19}$ - 5% updates, 95% reads), a *read-mostly* workload[20]. An application that has a workload similar to Workload A could be a session store in which recent actions are recorded, while Workload B is similar to the workload of an application that involves photo tagging, as stated in [20]. Every workload was tested with the following values for the number of operations (NO): 1000, 10000, 100000 and 1000000, and with 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 client threads (NT). Each test was repeated three times.

MongoDB was installed with default settings, which implies that the storage engine used is Wired Tiger (the default storage engine for MongoDB version 3.4.4).

Cassandra was also installed with default settings, but we followed a setting recommendation found in [7], so that write timeouts can be avoided:

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

The asynchronous Java driver was used for both MongoDB and Cassandra. A batch of tests contains all tests with the same workload, number of operations, and database server, varying the number of client threads. Before the execution of every batch of tests, the database server was restarted. Information about database server status was captured before running a batch of tests and after. The data set corresponding to the first workload was deleted after all combinations of tests related to it were executed. A data set having the same parameters but corresponding to the second workload was loaded before executing the tests for the second workload.

3.2. **Results.** Every batch of tests was repeated three times, and the average values of average update latency (AUL) and average read latency (ARL) were computed. Figures 1 and 2 show the AUL results for Workload A and Workload B, respectively. Figures 3 and 4 refer to ARL. Here the x-axis represents $log_2(NT)$.

When testing Workload A (Figure 1), the evolution of AUL with respect to the number of threads has a quasi-parabolic shape, with the exception of the case with 1000 operations. There is an optimal number of threads thr_{min} for which AUL has a minimum value. This point is slightly shifting to the right as the number of operations grows. This leads us to the conclusion that for a fixed number of operations there is a threshold thr_{min} producing the minimum AUL value. When the number of threads exceeds this thr_{min}, AUL grows: in the case of Cassandra, the slope of the curve decreases with the number of operations; in the case of MongoDB, the slope is increasing as the number of threads grows. In the case with 1000 operations, the minimum AUL value is obtained for $thr_{min} = 2^2$, and the growth at some maximum value ($thr_{max} = 2^7$ for Cassandra, $thr_{max} = 2^8$ for MongoDB) after which it starts diminishing. MongoDB and Cassandra produce almost the same AUL value for 10000 operations when NT $\geq 2^5$. For a number of threads less than 2^8, AUL for 10000 operations is greater than AUL for 100000 operations in the case of MongoDB. The maximum AUL value is obtained for 10000 operations (Cassandra), respectively 1000000 operations (MongoDB).

For Workload B tests (Figure 2), the quasi-parabolic shape is preserved, again with the only exception of 1000 operations test case. The threshold thr_{min} has a slower shift to the right than in the case of Workload A. The way AUL values grow is the same for both workloads in the case of Cassandra, while in the case of MongoDB the AUL growth rate is slower for Workload B than in the case of Workload A, and AUL slightly diminishes with the number of operations. In the case with 1000 operations, the minimum AUL is obtained for $thr_{min} = 2^1$ (MongoDB), respectively $thr_{min} = 2^2$ (Cassandra), and the growth stops at $thr_{max} = 2^8$ (for both databases). In the case of MongoDB and Workload B, almost identical AUL values were obtained when number of operations was 100000, respectively 1000000. The maximum AUL value is obtained for 1000 operations and 2^8 client threads for both databases (AUL = 20588 for Cassandra, and AUL = 11854 for MongoDB). The worst performance in terms of AUL for Cassandra is almost double the similar MongoDB's one.

Figures 3 and 4 show the variation of ARL for workloads A and B. At first sight, the shapes are almost identical with those referring to corresponding AUL values. For 1000 operations and Workload A, the maximum ARL value is obtained for $thr_{max} = 2^7$ (Cassandra), respectively $thr_{max} = 2^8$ (MongoDB), while in the case of Workload B $thr_{max} = 2^8$ for both databases. For all cases considered when testing Workload A, maximum ARL value is obtained for $thr_{max} = 2^9$ with 1000000 operations (MongoDB), respectively 10000 operations (Cassandra). For Workload B, global ARL maximum values are obtained in the case $thr_{max} = 2^8$ and 1000 operations for both databases.

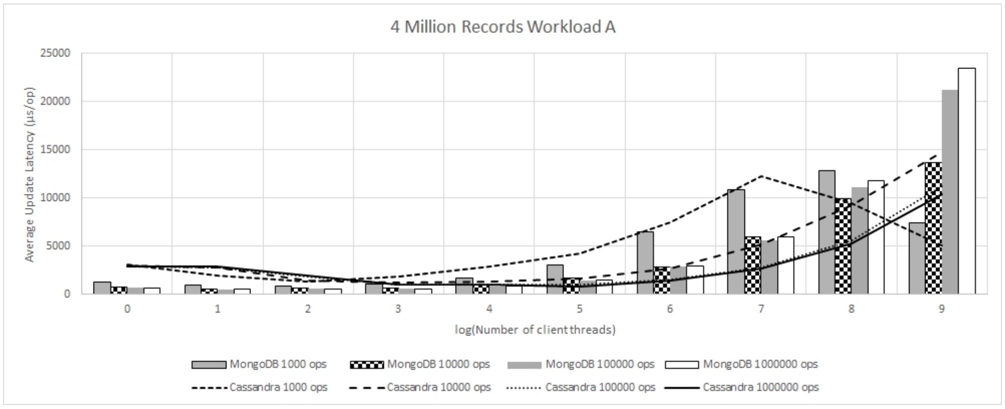

FIGURE 1. 4 Million Records Workload A - Average Update Latency

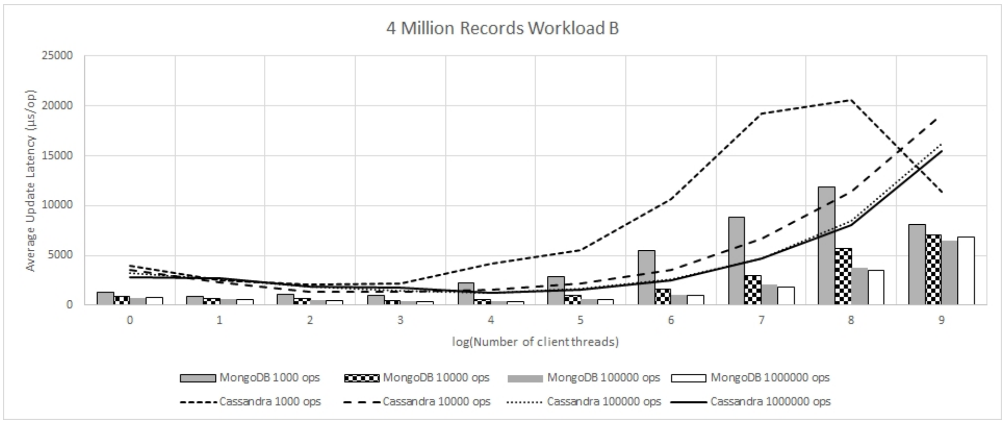

FIGURE 2. 4 Million Records Workload B - Average Update Latency

TABLE 1. Analysis of variance - update latency 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	0.1131	0.7378		20.0319	3.784e-05	***	2.0400	0.1588	
A	10000	3.8686	0.05416	.	851.0068	<2e-16	***	0.3264	0.57005	
A	100000	59.212	2.46e-10	***	2700.088	<2.2e-16	***	402.539	<2.2e-16	***
A	1000000	109.56	8.514e-15	***	2960.81	<2.2e-16	***	633.90	<2.2e-16	***
B	1000	8.6920	0.004654	**	26.1476	3.984e-06	***	0.7462	0.391370	
B	10000	196.81	<2.2e-16	***	1141.68	<2.2e-16	***	210.28	<2.2e-16	***
B	100000	315.01	<2.2e-16	***	1623.29	<2.2e-16	***	254.34	<2.2e-16	***
B	1000000	352.39	<2.2e-16	***	1879.47	<2.2e-16	***	242.42	<2.2e-16	***

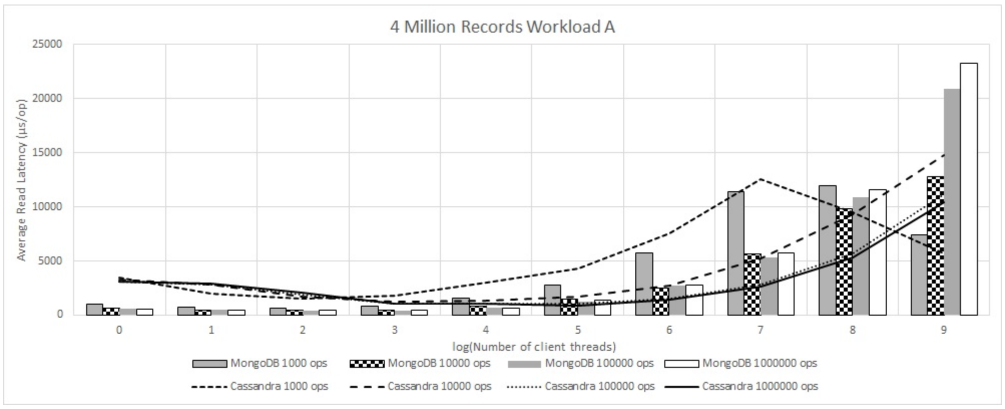

FIGURE 3. 4 Million Records Workload A - Average Read Latency

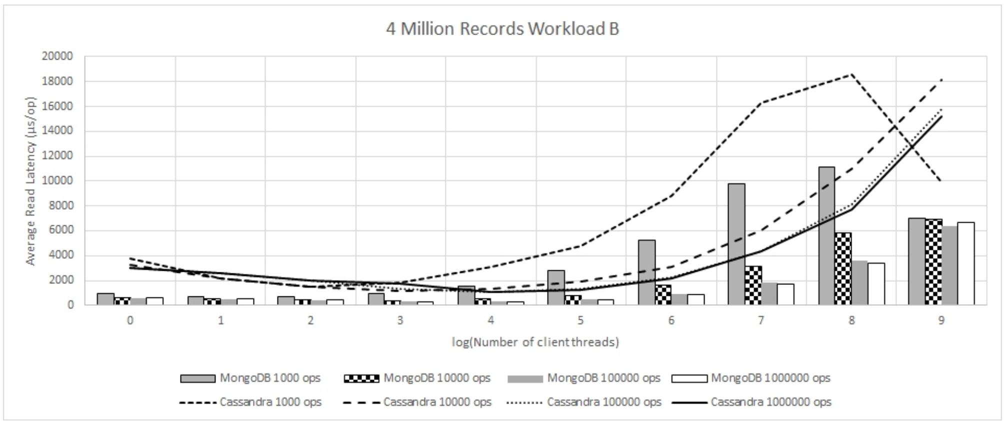

FIGURE 4. 4 Million Records Workload B - Average Read Latency

TABLE 2. Analysis of variance - read latency 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	0.6449	0.4253		21.0577	2.558e-05	***	1.7397	0.1925	
A	10000	9.1106	0.003821	**	727.9405	<2.2e-16	***	0.0037	0.952034	
A	100000	39.508	5.196e-08	***	2496.036	<2.2e-16	***	369.484	<2.2e-16	***
A	1000000	81.258	1.71e-12	***	2692.374	<2.2e-16	***	578.117	<2.2e-16	***
B	1000	6.3605	0.01454	*	25.9703	4.241e-06	***	0.6839	0.41175	
B	10000	160.54	<2.2e-16	***	1045.30	<2.2e-16	***	172.17	<2.2e-16	***
B	100000	267.69	<2.2e-16	***	1422.88	<2.2e-16	***	211.16	<2.2e-16	***
B	1000000	257.98	<2.2e-16	***	1381.54	<2.2e-16	***	170.57	<2.2e-16	***

3.3. **Statistical analysis.** The two-way ANOVA (Analysis of Variance) procedure from R Statistics Package[17] was used to perform the statistical analysis of the experimental results. Table 1 (for AUL) and Table 2 (for ARL) present a synthesis of the results. For every experiment, two factors were taken into consideration: database and number of threads. The interactions between database and number of threads (DB:NT) were considered as well. The database factor (DB) has two levels: Cassandra and MongoDB. The number of threads (NT) has ten levels: 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512. The column labeled "Sgf" refers to the P-value and describes textually the level of significance, $0.1\%, 1\%, 5\%$, and 10%, following the usual conventions: 0 $***$ 0.001 $**$ 0.01 $*$ 0.05 . 0.1 (*blank space*) 1. A P-value less than or equal to 0.1% (i.e. $***$ conforming to the legend) shows that the differences between means have a strongest statistical significance, while a P-value greater than 10% (i.e. blank space) indicates that the differences between the means of the levels considered are within the experimental error.

With respect to the number of threads, the differences in variation of AUL and ARL have the strongest statistical significance for both workloads. In the same time, for 1000 operations and Workload A, the differences of means with respect to DB levels for AUL and ARL are within the experimental error. All interactions DB:NT lead to strongest significance between means, except for Workload A with 1000 and 10000 operations and Workload B with 1000 operations, for which there is no statistical significance.

4. Conclusions and further work

As we stated above, the performance of Cassandra and MongoDB database servers was measured for two different workloads: update-heavy (Workload A) and read-mostly (Workload B) and two performance indicators were measured and analyzed, average update latency (AUL) and average read latency (ARL). All test cases with NO $\geq$ 10000 proved a quasi-parabolic behavior of AUL and ARL with respect to NT. This means that the best performance is achieved with a right combination of NO and NT. As NT exceeds these optimal values, the performance in terms of latency diminishes. In the case NO = 1000, the minimum AUL is obtained at $thr_{min} = 2^2$ for Workload A and both databases, respectively at $thr_{min} = 2^1$ (MongoDB) and $thr_{min} = 2^2$ (Cassandra) for Workload B. There is also a global maximum AUL value obtained for Workload A at $thr_{max} = 2^9$ and NO = 10000 for Cassandra, respectively at $thr_{max} = 2^9$ and NO = 1000000 for MongoDB. For Workload B, maximum AUL values were obtained at $thr_{max} = 2^8$ and NO = 1000 for both databases. These figures are almost the same for ARL, basic trends being preserved. Global maximum

ARL values are obtained for the same combination of (DB, workload, NO, NT).

As further work, we intend to analyze other metrics obtained from this experiment, and to perform post-hoc ANOVA tests. Also, we plan to perform other experimental studies using data sets that do not fit within the internal memory on cluster and single server configurations. Another direction in the experimental work will deal with the use of SSDs as disk storage and the replication for database servers, in order to measure how these configurations affect performance. Finally, another variable in our future case studies will be the operating system, so that we can use other NoSQL DBMSs that are not available on Windows.

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: 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] 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.

[7] Fixstars. GridDB and Cassandra Performance and Scalability. A YCSB Performance Comparison on Microsoft Azure. Technical report, Fixstars Solutions, 2016.

[8] 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.

[9] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Big-Bench: 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.

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

[11] JSON. `https://www.json.org/`. Accessed: 2018-03-16.

[12] 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.

[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] 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.

[16] OrientDB. `http://orientdb.com/`. Accessed: 2017-09-25.

[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: 2019-01-03.

[19] The cassandra-stress tool. `https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsCStress.html`. Accessed: 2019-01-03.

[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.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania

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