

EXPERIMENTAL EVALUATION OF NOSQL DATABASES

Veronika Abramova¹, Jorge Bernardino^{1,2} and Pedro Furtado²

¹Polytechnic Institute of Coimbra - ISEC / CISUC, Coimbra, Portugal

²University of Coimbra – DEI / CISUC, Coimbra, Portugal

ABSTRACT

Relational databases are a technology used universally that enables storage, management and retrieval of varied data schemas. However, execution of requests can become a lengthy and inefficient process for some large databases. Moreover, storing large amounts of data requires servers with larger capacities and scalability capabilities. Relational databases have limitations to deal with scalability for large volumes of data. On the other hand, non-relational database technologies, also known as NoSQL, were developed to better meet the needs of key-value storage of large amounts of records. But there is a large amount of NoSQL candidates, and most have not been compared thoroughly yet. The purpose of this paper is to compare different NoSQL databases, to evaluate their performance according to the typical use for storing and retrieving data. We tested 10 NoSQL databases with Yahoo! Cloud Serving Benchmark using a mix of operations to better understand the capability of non-relational databases for handling different requests, and to understand how performance is affected by each database type and their internal mechanisms.

KEYWORDS

NoSQL databases, SQL databases, performance evaluation, database models, YCSB

1. INTRODUCTION

Relational databases were developed in 70's as a technology to store structured data organized as tables and with its own query language model – the Structured Query Language (SQL) [1]. Soon enough, databases became a vital part of organizations and became used all over the globe. However, with the constant growth of stored and analysed data, relational databases exhibit a variety of limitations. Data querying loses efficiency due to the large volumes of data, as well as storage and management of larger databases becomes challenging. NoSQL databases were developed to provide a set of new data management features while overcoming some limitations of currently used relational databases [2, 3]. In comparison to relational databases, NoSQL databases are more flexible and horizontally scalable. One of the main advantages of non-relational databases is represented by the non-existence of a rigid data structure that, in relational databases, must be defined before data can be stored. Although it is not possible to query stored records using SQL, this approach allows easy storing and retrieving of data, regardless of structure and content. NoSQL enables easier application management and removes the necessity of application modification or database schema change. Besides, with the increase of data, NoSQL databases exhibit better and easier horizontal scalability [4]. Those databases are capable of taking advantage of new clusters and nodes transparently, without requiring database

administrator management or manual distribution of information. Horizontal scalability allows hundreds of low-cost servers to satisfy requests, while company's costs are reduced. It is more profitable to use such a distributed system instead of building mainframes with high-end capacities. In comparison to relational models, the cost per request or amount of stored data is reduced. Since database administration may be a difficult task with such amounts of data, NoSQL databases are projected to automatically manage and distribute data, recover from faults and repair the whole system automatically [5].

Therefore, non-relational databases emerged as disruptive technologies that can be used sole or as a complement to relational databases, increasing its performance while bringing different new characteristics and advantages. Currently, there are over 150 NoSQL databases with diverse features and optimizations [2]. With the increase of popularity of non-relational databases, some features and system characteristics started to evolve. When new database technologies started to emerge, NoSQL databases were known and characterized by lack of consistency of its data. For some type of companies and systems that could be a limitation, cases in which strong consistency was important and a necessary characteristic. However, today, there are different NoSQL databases that provide all new features and advantages while keeping data consistent or even eventually consistent, depending on the system needs. Also, in order to increase speed of requests, non-relational databases began to use volatile memory. Since I/O over hard disks is slower, mapping databases or just parts of those into volatile memory increases performance and reduces execution time for requests. Yet, although use of non-relational databases has increased over past years, their capabilities have not been disclosed. Since there are different types of NoSQL databases, in order to choose a database that would be more appropriate for a specific business, it is important to understand its main characteristics. Similarly to relational databases, each NoSQL database provides different mechanisms to store and retrieve data, which directly affects performance. Each non-relational database has different optimizations, resulting in different data loading time and execution times for reads or updates. Evaluation performed allows us to compare different types of NoSQL databases while performing read and update operations, based on their execution time. We tested 10 NoSQL databases: Cassandra, HBase, Oracle NoSQL, Redis, Scalaris, Tarantool, Voldemort, Elasticsearch, MongoDB and OrientDB, evaluating the data load times and execution speeds for different types of requests. For this purpose we used a benchmark with a typical range of workloads, Yahoo! Cloud Serving Benchmark [6], which simulates various uses of a database. Analysis and comparison of the results allowed us to verify how the different features and optimizations result in terms of assessment of the system and its main features.

This paper describes our evaluation of NoSQL databases while comparing their execution time for different operations. We compare execution time of diverse requests for three types of NoSQL databases: Document Store, Column Family and Key-value Store, in order to better understand how performance of a database is affected by its type. On the other hand, this study was performed in a non-distributed environment. It is important to notice that performance evaluation is important and necessary to better understand the differences between NoSQL databases types.

The remainder of this paper is organized as follows. The next section presents related work. Section 3 describes NoSQL databases. Section 4 presents the setup used for tests followed by the section 5 with the experimental evaluation. Lastly, in section 6 we present our conclusions and suggest future work.

2. RELATED WORK

The concept of NoSQL was first used in 1998 by Carlo Strozzi to refer to an open source database that does not use SQL interface [7]. Strozzi prefers to refer to NoSQL as "nosequel" or "Norel" (no relational) since it is the main difference between this technology and relational model. Its origin can also be related to the creation of Google's BigTable model [8]. This database system, BigTable, is used for storage of projects developed by Google, for example, Google Earth. Amazon subsequently developed its own system, Dynamo [9]. These projects allowed taking a step towards the evolution of NoSQL. However, the term re-emerged only in 2009, at a meeting in San Francisco organized by Johan Oskarsson [10]. The name for the meeting, NoSQL meetup, was given by Eric Evans and from there on NoSQL became a buzzword.

Over the last years NoSQL databases have been tested and studied as well as their performance has been evaluated. As overall analysis and theoretical approach that describes characteristics and mechanisms of NoSQL databases, there are a variety of papers, such as, [11, 12, 13]. However, due to increased interests in non-relational technology, NoSQL databases have been analysed not only from application perspective, but as enterprise ready and advantageous databases. Therefore, the research of their performance, characteristics and used mechanisms, has been increased. Some of those studies, such as [6, 14, 15], evaluate advantages of use of NoSQL technology by analysing the throughput and the advantages that are brought by scalability of NoSQL databases. Differently of previous authors we do not evaluate throughput but compare execution time presented by tested databases to determine their performance.

In the next section we describe NoSQL databases by presenting some of their characteristics and existing types.

3. NOSQL DATABASES

In order to overcome some limitations, usually NoSQL databases are based on BASE (Basically Available, Soft State, and Eventually Consistent) principle that is characterized by high availability of data, while sacrificing consistency [16, 17]. On the other hand, relational databases are represented by ACID (Atomic, Consistent, Isolated, and Durable) principle where all the transactions committed are correct and do not corrupt database, and data is consistent [16]. However, there was an increase of popularity of NoSQL databases, some of those based on the ACID principle and characterized by all the data being immediately consistent. Both principles come from the CAP theorem - Consistency, Availability, and Partition Tolerance [18]. According to this theorem, when it comes to working with distributed systems, there can be only two of the three guarantees (C, A or P) so we need to choose the most important. When the consistency of data is crucial, relational databases should be used. Comparing both models, it may be considered that BASE is more flexible than ACID. When the data is spread across multiple servers, the consistency becomes hard to achieve. All NoSQL databases are characterized by the inexistence of the relations between different records. However, those databases can be divided into four categories accordingly to different optimizations [19]:

- Key-value store. In these databases all the stored data is represented by a pair of key and value per record, meaning that each key is unique and it allows accessing record's

information, represented as value. This structure is also known as “hash table” where data retrieval is usually performed by using key to access value.

- Document Store. These databases are designed to manage data stored in documents that use different format standards, such as, XML [20] or JSON [21]. This type of storage is more complex in comparison to storage used by Key-value Stores and enables data querying.
- Column Family. The database structure of this NoSQL type is similar to the standard Relational Database Management System (RDMS) since all the data is stored as sets of columns and rows. Columns, that store related data that is often retrieved together, may be grouped.
- Graph Database. These databases are mostly used when the stored data may be represented as a graph with interlinked elements such as, social networking, road maps or transport routes.

Hence, Key-value Store databases are more appropriate for the management of stocks, products and real time data analysis, providing high data retrieving speed while the greatest amount of data is mapped into memory. Document Store databases are a good choice when working with large amounts of documents that can be stored into structured files such as text documents, emails or XML and CMS and CRM systems. Column Family databases should be used when the number of write operations exceeds reads, for example in logging. Finally, graph databases are more appropriate for working with connected data, for example, to analyse social connections among a set of individuals, road maps and transport systems.

In the next section we will describe experimental setup that was used during evaluation and specify versions of databases that were tested.

4. SETUP

For the experimental analysis we used the YCSB - Yahoo! Cloud Serving Benchmark [6], which is widely used for evaluation and performance comparison of NoSQL databases. The benchmark consists of two components: a data generator and a set of performance tests to evaluate read and update operations. Each of the test scenarios is called workload and is defined by a set of features such as a percentage of read and update operations, total number of transactions, number of records used, etc. The benchmark package provides a set of default workloads that may be executed, as follows:

- Workload A: Update Heavy. Consists of a ratio of 50/50 of read/update;
- Workload B: Read Mostly. It consists of a ratio of 95/5 of read/update;
- Workload C: Read Only. This workload is 100% read;
- Workload D: Read Latest. This workload consists of 95/5read/insert. In this workload new records are inserted and these are treated as the most popular;

- Workload E: Short Ranges. Consists of a ratio of 95/5 scan/insert. In this workload are consulted small intervals of records, between 1 and 100, instead of individual records. Includes insertion of the new records;
- Workload F: Read-modify-write. This workload consists of a ratio of 50/50 read/read-modify-write. In this workload a record is read, updated and all the changes are saved.

However, in order to better understand the optimizations of databases and the execution speed of update operations, we have created two additional workloads that replaced the execution of workloads D and E. We decided not to execute workloads D and E due to our main interest in evaluating read and update operations and data loading speed was analysed apart. The new workloads have the following characteristics:

- Workload G: Update mostly. Consists of a ratio of 5/95 of read/update;
- Workload H: Update Only. This workload is 100% update.

In order to evaluate the loading time, we generated 600,000 records, each with 10 fields of 100 bytes generated randomly over the key registry identification, roughly 1kb total per record. Each record is uniquely identified by a key composed by string "user" followed by several digits, for example "user379904308120", which is the registration key. Each field of the record is identified as field0, field1, ..., field i respectively. Loading of the data was done using the workload A that has predefined a Zipfian distribution [6]. The execution of workloads was made using 1000 operations, which means that there were 1000 requests to the database under test, while varying the number of records stored and operations.

All the tests were executed on a virtual machine Ubuntu Server 32bit with 2GB RAM available, hosted on a computer with Windows 7 and a total of 4GB RAM. It is important to notice that evaluation of Oracle NoSQL was performed using KVLite version [22]. This is a version of the database which consists of a single process and easy configuration, but which is not optimized for performance. To perform a full evaluation, it is necessary to have a large JVM cache, which would increase the performance of the database.

In the study that was performed, Graph databases have not been evaluated. Because as stated by [23], Graph databases should not be evaluated according to the scenarios used in the analysis of the other types of NoSQL databases (Key-value Store; Document Store; Column Family), with requests formed by reads and update operations. Usage of links between records requires a different approach, so there are specific benchmarks developed to evaluate the performance of Graph databases such as, XGDBench [24].

During experimental evaluation, we tested the following NoSQL databases:

- Cassandra version 1.2.1 (<http://cassandra.apache.org/>)
- HBase version 0.94.10 (<http://hbase.apache.org/>)
- Oracle NoSQL version 2.1.8 (<http://www.oracle.com/>)

- Redis version 2.6.14 (<http://redis.io/>)
- Scalaris version 0.5 (<https://code.google.com/p/scalaris/>)
- Tarantool version 1.5.1 (<http://tarantool.org/>)
- Voldemort version 0.96 (<http://www.project-voldemort.com/>)
- Elasticsearch version 0.90.3 (<http://www.elasticsearch.org/>)
- MongoDB version 2.4.6 (<http://www.mongodb.org/>)
- OrientDB version 1.5 (<http://www.orientdb.org/>).

5. EXPERIMENTAL EVALUATION

In the following subsection we present and analyse the results of loading 600.000 records generated with the YCSB. We also present the execution times for workloads based only on read operations, updates and both at the same time.

5.1. Data Loading

Figure 1 shows the results of data loading that was evaluated by comparing execution time of tested databases while loading 600.000 records.

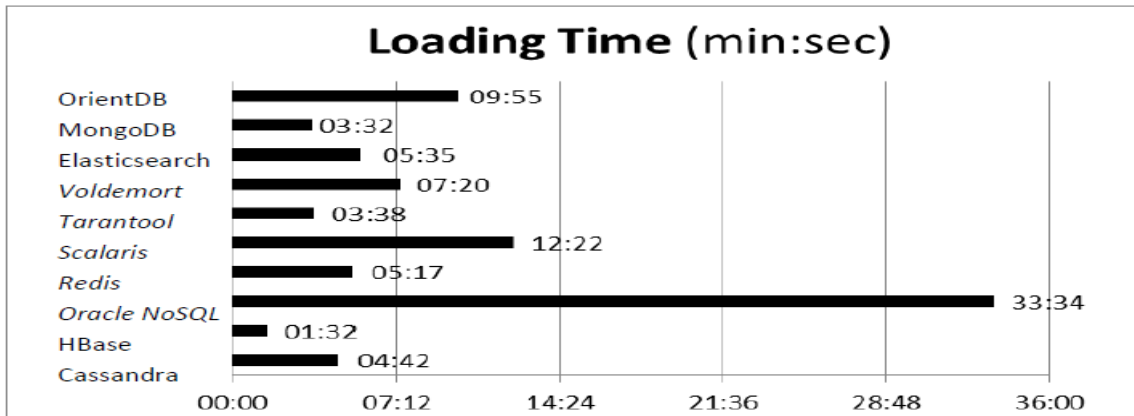


Figure 1. Loading time for 600.000 records

Figure 1 shows loading times of the ten tested databases while inserting 600.000 records. It also organizes the results according to the three types of databases that were tested. At the bottom are presented Column Family databases (Cassandra and HBase); in the middle Key-value Store databases (Oracle NoSQL, Redis, Scalaris, Tarantool and Voldemort); and at the top Document Store databases (Elasticsearch, MongoDB and OrientDB). While loading 600.000 records, the best insertion time among all databases was presented by HBase, with a loading time of only 1 minute and 32 seconds. This means that this database was 3.34 times faster than Cassandra.

OrientDB was 1.78 times slower than the Elasticsearch and 2.87 times slower than MongoDB. OrientDB is the Document Store database with the worst loading time. The Oracle NoSQL database showed inferior performance compared to all the other databases, with a loading time of 33 minutes and 34 seconds and being 9.86 slower than the fastest database among Key-value Store databases, Tarantool. With this number of loading records (600.000), Redis, Tarantool and OrientDB failed to handle the load request with only 2GB of memory available. To be able to evaluate the performance of loading 600.000 records, the amount of RAM had to be increased up to 2.9GB. However, Redis was not able to meet the request of insertion of 600.000 records at once. Thus, the loading time of 600.000 records is equal to the sum of two loadings of 300.000 records since this was the higher amount of records we were able to load at once. Oracle NoSQL could handle the load with only 2GB of RAM, although affecting its results and exhibiting the worst performance. By analysing the results presented by HBase, it is possible to conclude that this database does not require a large amount of memory, making effective management of operations in the virtual machine environment.

Results obtained show that different types of databases have different data loading results. Column Family databases had the best average performance, and HBase showed better results than Cassandra. Among Document Store databases, MongoDB presented better performance while OrientDB had the worst result. Oracle NoSQL had highest loading time between Key-value Store databases, and also overall worst performance among all tested databases. High performance of Key-value Stores and Column Family databases is due to use of logs and posterior flush to disc while Document Store databases store data on the hard disc more frequently.

In the following sections we will present the results of execution time for the different workloads.

5.2. Workload A

Figure 2 shows the results obtained in seconds while executing workload A that consists of 50% reads and 50% updates, over 600.000 records.

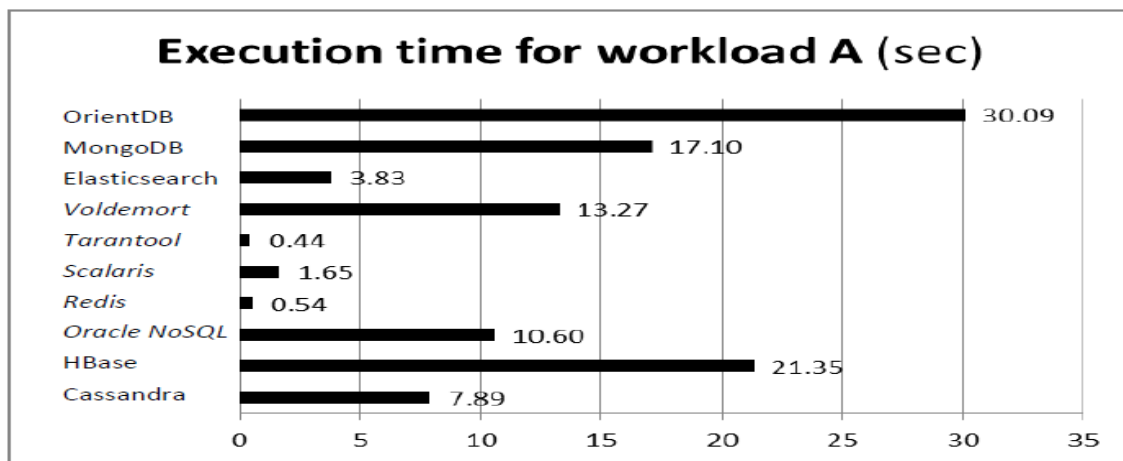


Figure 2. Execution time for workload A (50/50 of read/update)

When analysing the results of execution of workload A, good performance results are presented by Key-value Store databases, with Tarantool being the fastest due to the mapping of data into memory. This is followed by Redis, with only a little difference. Among Key-value Store databases, Voldemort presented the worst result, 30.15 times worse performance than Tarantool. Among the databases of Column Family type, Cassandra exhibited a performance of 7.89 seconds, 2.70 times faster than HBase. The worst performance was presented by OrientDB (30.09 seconds), with an execution time 7.85 times higher than shown by Elasticsearch, and 1.75 times higher compared to MongoDB. The large execution time of OrientDB is due to the fact that records have to be read from disk, consequently with its lower speed compared to the volatile memory.

Overall, Key-value Store databases showed good results during execution and the fastest databases among all databases evaluated were Redis and Tarantool. On the other hand, most of the Document Store databases showed lower performance and OrientDB had higher execution time while Column Family databases, Cassandra and HBase, displayed average results.

5.3. Workload B

Figure 3 shows the results obtained while executing workload B that is composed by 95% reads and 5% updates with a total of 1000 operations into 600.000 records.

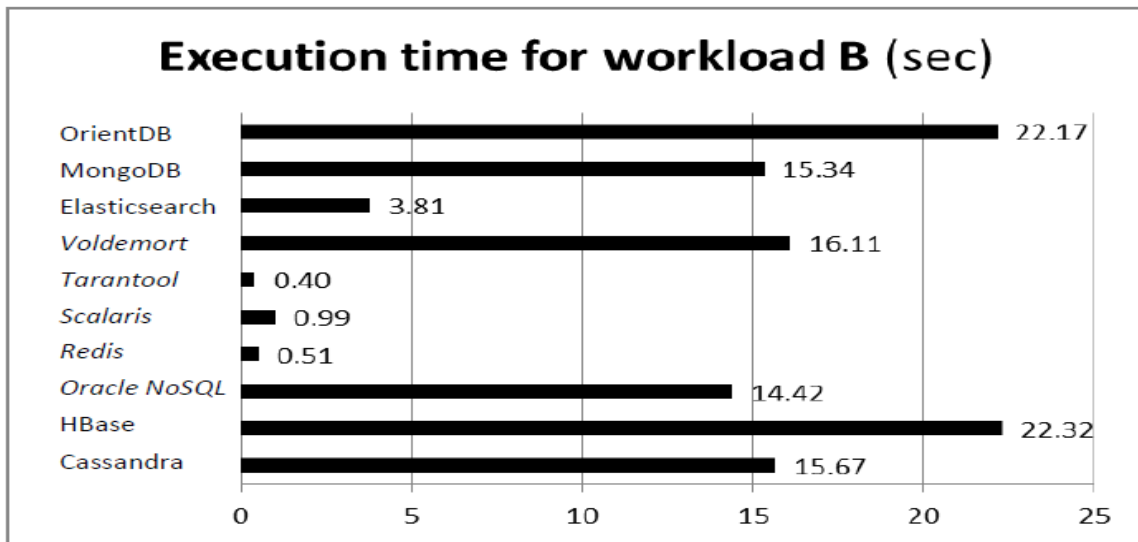


Figure 3. Execution time for workload B (95/5 of read/update)

Databases with the best execution time of workload B with 600.000 records were Tarantool and Redis with execution times of 40ms and 51ms, respectively. Among Column Family databases, HBase exhibited the worst execution time, while Cassandra had an execution time 1.42 times lower, compared to HBase. Although both of those databases use the same sequential writing mechanisms, Cassandra was able to show better results. Among the Document Store databases, Elasticsearch showed the best performance, while OrientDB had the worst outcome. Elasticsearch is built on top of Apache Lucene engine, which allows that database to use main memory as well as compress existing documents which improves retrieval. Since there are no

records in cache, OrientDB has higher execution time since it is necessary to read data from disk. The best results for Key-value Store databases are due to the optimization of those databases to work with limited resources and a small number of operations (remember that tests have a total of 1000 operations). Conversely, lower performance of HBase is due to the lack of the optimization to execute read operations. Since this database uses sequential writing mechanisms, parts of the same record can be stored in different files, which make reads less efficient.

During execution of workload B, Column Family databases and Document Store databases had similar results. Among Key-value Store databases, worst results were exhibited by Voldemort.

5.4. Workload C

Figure 4 shows the results obtained while executing workload C that consists of execution of 1000 read operations into 600.000 records.

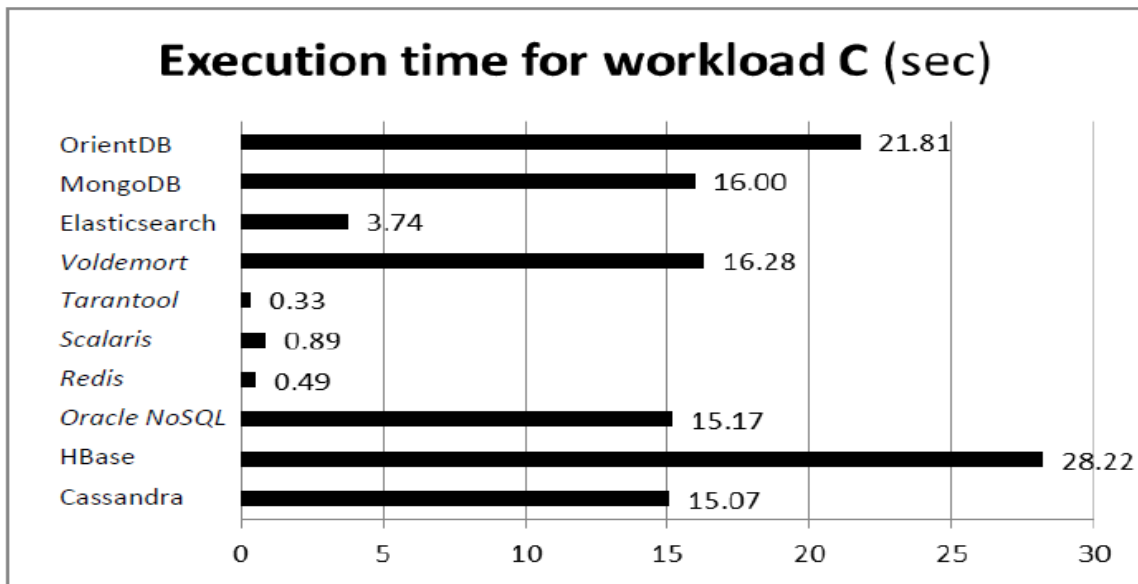


Figure 4. Execution time for workload C (100% read)

The results of execution of workload C indicate that HBase and OrientDB are the databases that showed the highest execution time for read operations. HBase presented the worst result, with its performance 1.86 times lower compared to performance presented by Cassandra. Given a large number of records, HBase showed more difficulty on reads execution. This database is optimized for the execution of updates, since it uses sequential write mechanisms and with that, parts of the same record that is being read are stored in different files, which results in an increased execution time. The second worst outcome was by OrientDB, due to not using the memory for mapping records since in our evaluation we used local version of this database instead of memory version. This database was 5.83 times slower than the Document Store database that showed the best results, Elasticsearch. Moreover, Key-value Store databases had good execution times, while keeping records in memory, and Tarantool is the database with the minimal execution time for read operations. These databases are projected to fast record retrieval using key while mapping data in memory.

The execution of workload C allowed us to better understand performance of tested databases while executing only reads. During these tests, Cassandra had the worst results and was the slowest database. Since Cassandra use sequential write mechanisms, reads become slower and performance is decreased. Document Store databases had higher execution time compared to Key-value Stores since those databases use less volatile memory and execute workloads while using disk I/O operations.

5.5. Workload F

Figure 5 shows the results obtained while executing workload F that is read-modify-write workload. Were performed 1000 operations over databases with 600.000 records stored.

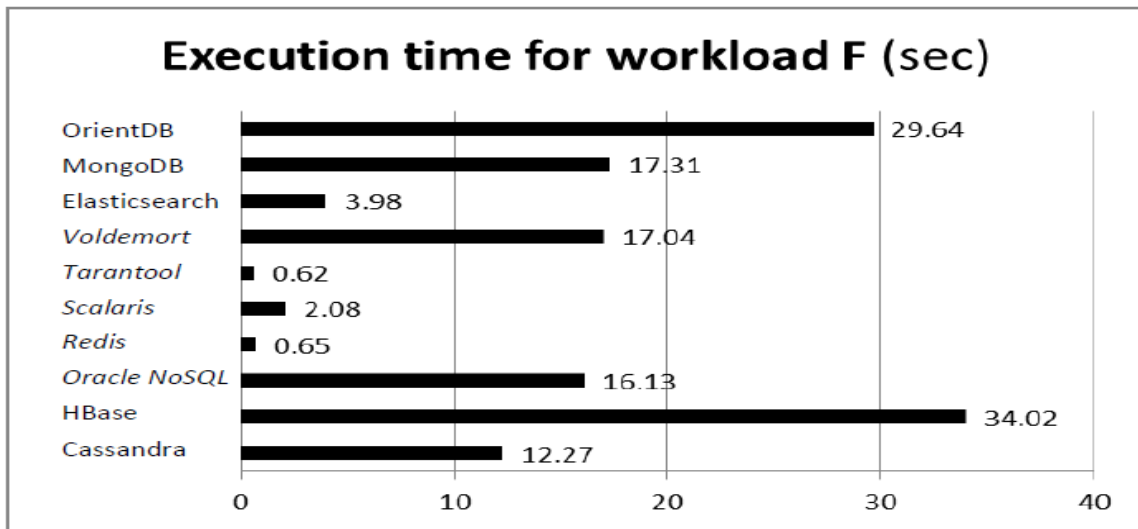


Figure 5. Execution time for workload F (read-modify-write)

Workload F is a read-modify-write workload and in this workload, a record is read, changed and the changes are written permanently. With the execution of 1000 operations in a database with 600.000 records, Tarantool and Redis kept the best results on execution of the workload. Among Key-value Store databases the execution time of Voldemort was 27.48 times higher compared to the result presented by Tarantool. The higher execution time and the worst performance were presented by HBase and this database has presented a performance 2.77 times lower compared to Cassandra. The best result was presented by Tarantool, due to the fact that it uses volatile memory and since this database makes an efficient management of requests. On the other hand, HBase showed a lower performance due to the difficulty on executing reads compared with the updates.

Similarly to previous workloads, the best results are displayed by Key-value Store databases while Document Store databases had worse performance. However, the slowest database was Cassandra and it is Column Family database. HBase not only had higher execution time among Column Family databases but in comparison with all tested databases. Best performance had two of the Key-value Store databases, Redis and Tarantool. Both those databases are in-memory which improves data access speed and significantly reduces execution time.

5.6. Workload G

Figure 6 shows the results obtained while executing workload G with 600.000 records, performing 1000 operations with 5% reads and 95% updates.

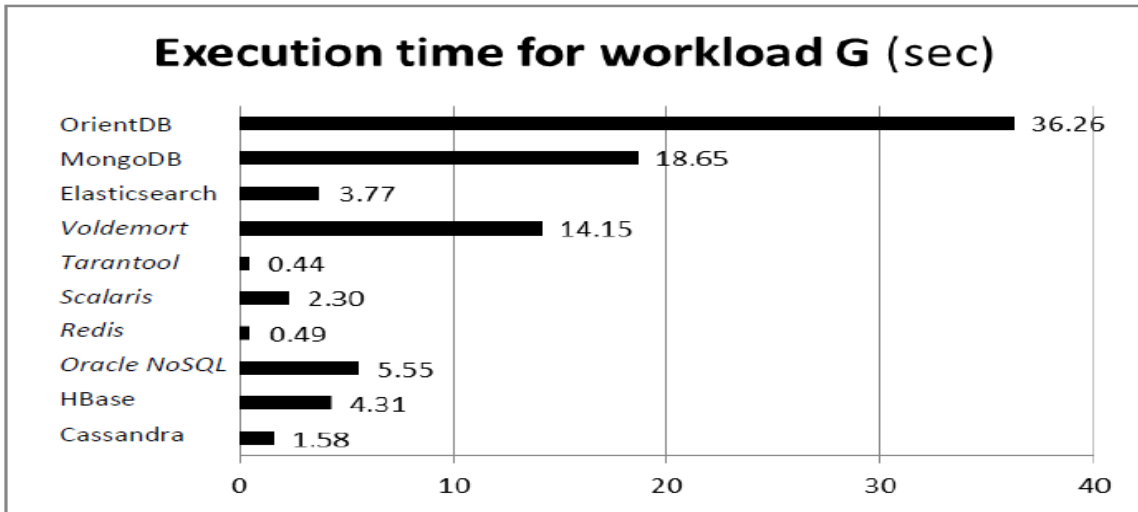


Figure 6. Execution time for workload G (5/95 of read/update)

While executing workload G, Tarantool and Redis showed the highest performance among Key-value Store databases and had the best performance since those databases map records into memory. Among Column Family databases, Cassandra was 2.72 times faster than HBase and had a good performance due to the use of a log where all transactions are written into append mode, registering all the operations ever made. This means that only the log needs to be written to disk while the records are stored in memory. Thus, by reducing disk operations, performing updates becomes faster. Both HBase and Cassandra use sequential write mechanism and those databases are highly optimized to execute updates. The worst result had OrientDB due to high use of disk during execution of operations.

Results of execution of workload G demonstrated high difference in comparison with previous workloads that were tested. Both Column Family and Key-value Store databases showed good results with low execution times. On the other hand, Document Store databases had the worst results with OrientDB showing the highest execution time. Among Key-value Store databases, Voldemort had the worst results. This database uses WAL – Write Ahead Log that increases execution time since before updating record database register operations in log

5.7. Workload H

Figure 7 shows the results of the execution of workload H with 1000 updates into a database with 600.000 records.

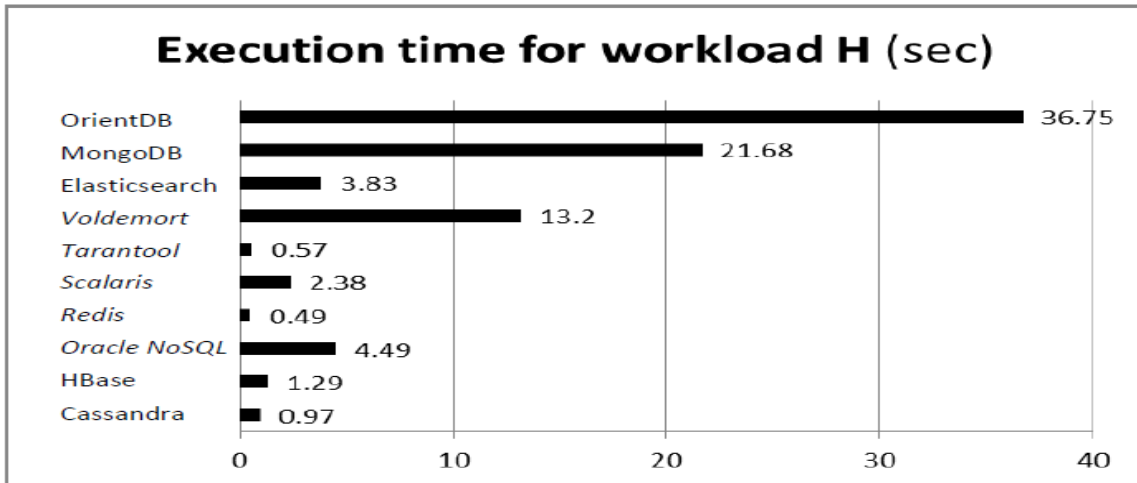


Figure 7. Execution time for workload H (100% update)

With the execution of workload H with 1000 updates we observed better optimization of some databases on execution of updates. Cassandra and HBase Family Column databases are optimized for performing updates since records are stored in memory, reducing the number of operations performed on the disk and thereby increasing the performance. OrientDB had the highest execution time with a total of 36.75 seconds having a performance 9.59 times lower compared to the performance shown by Elasticsearch, which is the Document Store database with the best performance. This difference in execution time is due to the distinction of the type of storage used by databases. The OrientDB keeps records on disk rather than use in memory mapping. Also, the poor results of MongoDB are due to use of locking mechanisms to perform update operations, increasing execution time. Key-value Store databases are in-memory databases and since volatile memory is used to map records the database performance is increased.

Execution of workload H showed that Column Family and Key-value Store databases are highly optimized for execution of updates. Furthermore, Cassandra and HBase showed good results and were faster than some of the Key-value Store databases. However, Document Store databases high execution time and relatively low performance. Those databases show lower performance especially in comparison with most of Key-value Store Databases that highly use volatile memory in order to reduce execution time.

5.8. Overall Evaluation

Over previous subsections we presented and described results obtained during execution of workloads and data. Until now, previously described results are divided by the operation type and do not fully allow us to understand each database performance. Figure 8 shows the total execution time, values in minutes and seconds, for each one of the tested databases. These results represent total execution time, obtained by adding the execution times of all executed workloads (A + B + C + F + G + H), sorted in descending order, from highest execution time to lowest.

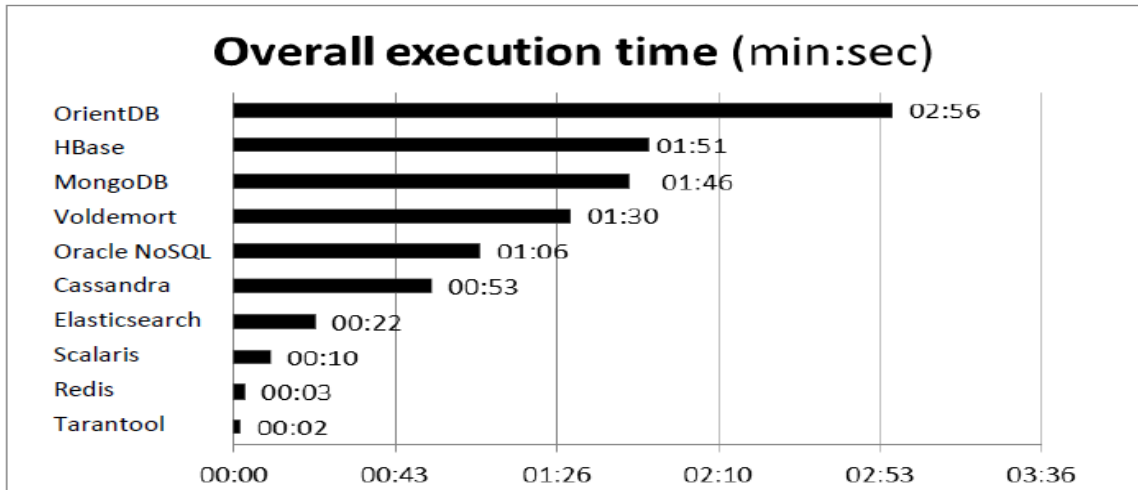


Figure 8. Overall execution time

While analysing overall results, it is possible to notice that in-memory databases had the best performance. The execution time increased with the increased use of disk operations by databases. Lowest execution time was exhibited by Tarantool, which is a Key-Value Store database, and the worst outcome was that of OrientDB. This database is a combination of two types of NoSQL databases, Document Store and Graph.

5.9. Performance Evaluation

Over the past years NoSQL databases has gained more and more popularity due to the big volumes of processed and stored data. These databases bring a number of advantages compared to relational databases, especially for large volumes of data that is not structured. In order to provide better database solution, according to the application type, NoSQL databases are divided into different types, each providing a specific set of features. The experimental evaluation has enabled us to compare three types of non-relational databases: Column Family, Key-Value Store and Document Store, by comparing their execution times for different types of workloads.

Throughout the experimental evaluation, the performance of different NoSQL databases was evaluated using Yahoo! Cloud Serving Benchmark with a set of workloads consisted of a combinations of read and update operations defined in a range of scenarios. Through the analysis of the results we concluded that different types of NoSQL database show diverse optimizations and used mechanisms in order to improve performance. However, some of the databases depend on the amount of volatile memory, which reduces execution time but is a more expensive and volatile storage media. The databases with the best results use volatile storage, so all the operations are executed in shorter execution time cycles, due to the fast speeds of volatile memory compared to the extraction of the files stored on the hard drive. The databases with the best results, Tarantool and Redis, exhibited extremely fast response times regardless of workloads. Databases with lowest performance were Oracle NoSQL on data loading and OrientDB on execution of a set of workloads. While analysing obtained results we concluded that both these databases require more system capacities compared with the environment used in the evaluation. Therefore, their capabilities were limited by the memory management, the operating system and the use of virtual machine environment. HBase and Cassandra are databases that use

a log for storing all performed changes, while the records are stored in memory for subsequent disk flush. The use of these mechanisms and following sequential writing to disk reduces the amount of disk operations that are characterized by low speed compared to the speed of the volatile memory. Thus, these databases are especially optimized for performing updates, while reads are more time consuming when compared with in-memory databases. MongoDB is the database that showed largest increase in execution time directly proportional to the number of performed updates. This database uses locking mechanisms that increase execution time but, on the other hand, the reads are not exclusive, so the mapping of registers in memory increases performance. OrientDB performance also degraded with the increasing number of update operations. As a global analysis, in terms of optimization, the databases can be divided into two categories, databases optimized for reads and databases optimized for updates. Thus, MongoDB, Redis, Scalaris, Tarantool and OrientDB are databases optimized to perform read operations, while Cassandra, Oracle NoSQL, HBase and Voldemort have a better performance on execution of updates. Even though Redis showed same execution time during execution of workload C (100% read) and workload H (100% update), we noticed that this database had better performance executing workload B (95% read and 5% update) than workload A (50% read and 50% update). Therefore we considered Redis more optimized for execution of reads.

According to the results, the best database is Tarantool, showing good execution times for all types of workloads. We believe that in-memory databases have high potential and are highly optimized to reduce execution time.

6. CONCLUSIONS AND FUTURE WORK

NoSQL stores promise good performance and scalability for simple operations over possibly huge datasets. In this paper we have analysed the performance of some of the most promising examples, testing them against workloads with read, update and mixed read and update characteristics. After showing performance results and analysing them, we interpreted the results in the light of the characteristics of the NoSQL engines being tested. In particular, we discussed memory and disk patterns to reach evaluation analysis conclusions.

As future work, we will analyse the execution time with the increase in the number of operations performed and the effect of running over multiple servers. This evaluation will allow us to better understand how NoSQL behaves while running in distributed and parallel environments. We also plan to evaluate performance of Graph databases using a standard benchmark. Finally, we will evaluate the use of NoSQL and NewSQL solutions for decision support workloads.

ACKNOWLEDGMENTS

This work was partially financed by iCIS – Intelligent Computing in the Internet Ser-vices (CENTRO-07- ST24 – FEDER – 002003), Portugal.

REFERENCES

- [1] Fayeche, I. and Ounalli, H.: Towards a Flexible Database Interrogation. International Journal of Database Management Systems (IJDMS) Vol.4, No.3, June 2012 .
- [2] <http://nosql-database.org/>.

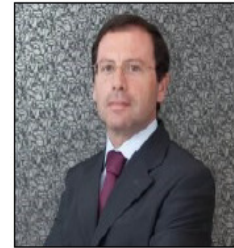
- [3] Vimala, S., Khanna Nehemiah, H., Bhuvaneshwaran, R. S., and Saranya, G.: Design Methodology for Relational Databases: Issues Related to Ternary Relationships in Entity-relationship Model and Higher Normal Forms. *International Journal of Database Management Systems (IJDMS)* Vol.5, No.3, June 2013.
- [4] Stonebraker, M.: SQL databases vs. NoSQL databases. *Communications of the ACM*, Vol. 53 No. 4, Pages 10-11.
- [5] Gajendran, S.: A Survey on NoSQL Databases, 2012, <http://ping.sg/story/A-Survey-on-NoSQL-Databases---Department-of-Computer-Science>.
- [6] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R.: Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, New York, NY, USA, 143-154.
- [7] Strozzi, C.: NoSQL – A relational database management system, 2013, <http://www.strozzi.it>.
- [8] Chang, F., Jeffrey, D., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.: Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 26(2), Article 4.
- [9] Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., and Vogels, W.: Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems principles (SOSP '07)*. ACM, New York, NY, USA, 205-220.
- [10] Notes from the NoSQL Meetup, 2013, http://developer.yahoo.com/blogs/ydn/posts/2009/06/nosql_meetup.
- [11] Hecht, R. and JABLINSKI, S.: NoSQL Evaluation A Use Case Oriented Survey. *Proceedings International Conference on Cloud and Service Computing*, pp. 12-14.
- [12] Han, J.: Survey on NOSQL Databases. *Proceedings 6th International Conference on Pervasive Computing and Applications*, pp. 363-366.
- [13] Leavitt, N.: Will NoSQL Databases Live up to Their Promise?. *Computer Magazine*, Vol. 43 No. 2, pp. 12-14.
- [14] Floratou, A., Teletia, N., Dewitt, D., Patel, J. and Zhang, D.: Can the elephants handle the NoSQL onslaught?. *Proc. VLDB Endow.* 5,1712-1723.
- [15] Tudorica, B.G. and Bucur, C.: A comparison between several NoSQL databases with comments and notes. *Roedunet International Conference (RoEduNet)*, pp.1-5.
- [16] Pritchett, D.: BASE: An Acid Alternative. *ACM Queue* 6(3), 48-55.
- [17] Cook, J. D.: ACID versus BASE for database transactions, 2009, <http://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base>.
- [18] Browne, J.: Brewer's CAP Theorem, 2009, <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>.
- [19] Indrawan-Santiago, M.: Database Research: Are We at a Crossroad? Reflection on NoSQL. *Network-Based Information Systems (NBIS)*, 15th International Conference on Network-Based Information Systems, pp.45-51.
- [20] Zhang, H. and Tompa, F.W.: Querying XML documents by dynamic shredding. In *Proceedings of the 2004 ACM symposium on Document engineering (DocEng '04)*. ACM, New York, NY, USA, 21-30.
- [21] Crockford, D.: *JavaScript: The Good Parts*. Sebastopol, CA: O'Reilly Media.
- [22] Lamb, C.: Oracle NoSQL Database in 5 minutes, 2013, https://blogs.oracle.com/charlesLamb/entry/oracle_nosql_database_in_5.
- [23] Armstrong, T., Ponnkanti, V., Dhruva, B., and Callaghan, M.: LinkBench: a database benchmark based on the Facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 1185-1196.
- [24] Dayarathna, M. and Suzumura, T.: XGDBench: A benchmarking platform for graph stores in exascale clouds. *Cloud Computing Technology and Science (CloudCom)*, IEEE 4th International Conference on, Taipei, 363 – 370.

AUTHORS

Veronika Abramova is currently a researcher at CISUC – Centre for Informatics and Systems of the University of Coimbra. Previously she has received a bachelor degree at Instituto Superior Engenharia de Coimbra (ISEC). She is currently working in an industrial project, with an electricity sector company, focused on transferring part of the company’s data to the non-relational storage. She has evaluated and studied different NoSQL databases, focusing on their performance comparison and characteristics. Her main research fields are business intelligence, big data, database knowledge management, NoSQL and SQL databases performance evaluation.



Jorge Bernardino received the PhD degree in computer science from the University of Coimbra in 2002. He is a Coordinator Professor at ISEC (Instituto Superior de Engenharia de Coimbra) of the Polytechnic of Coimbra, Portugal. His main research fields are big data, data warehousing, business intelligence, open source tools, and software engineering, subjects in which he has authored or co-authored dozens of papers in refereed conferences and journals. Jorge Bernardino has served on program committees of many conferences and acted as referee for many international conferences and journals. He was President of ISEC from 2005–2010. Actually, he is serving as General Chair of IDEAS conference and visiting professor at Carnegie Mellon University (CMU).



Pedro Furtado is Professor at University of Coimbra, Portugal, where he teaches courses in both Computer and Biomedical Engineering. His main research interests are data scalability and bigdata, data mining, service-oriented systems and real time systems. Lately, his research has focused both on scalable and real time warehousing, and also on middleware for wireless sensors in industrial and health-care applications. He has more than 100 papers published in international conferences and journals, books published and several research collaborations with both industry and academia. Besides a PhD in Computer Engineering from University Coimbra (UC) in 2000, Pedro Furtado also holds an MBA from Universidade Catolica Portuguesa (UCP).



Universidade Catolica Portuguesa (UCP).