NoSQL Databases: MongoDB vs Cassandra

Veronika Abramova
Polytechnic Institute of Coimbra
ISEC - Coimbra Institute of Engineering
Rua Pedro Nunes, 3030-199 Coimbra, Portugal
Tel. ++351 239 790 200
a21190319@alunos.isec.pt

Jorge Bernardino
Polytechnic Institute of Coimbra
ISEC - Coimbra Institute of Engineering
Rua Pedro Nunes, 3030-199 Coimbra, Portugal
Tel. ++351 239 790 200
jorge@isec.pt

ABSTRACT
In the past, relational databases were used in a large scope of applications due to their rich set of features, query capabilities and transaction management. However, they are not able to store and process big data effectively and are not very efficient to make transactions and join operations. Recently, emerge a new paradigm, NoSQL databases, to overcome some of these problems, which are more suitable for the usage in web environments. In this paper, we describe NoSQL databases, their characteristics and operational principles. The main focus of this paper is to compare and evaluate two of the most popular NoSQL databases: MongoDB and Cassandra.

Categories and Subject Descriptors
H.2 [**Database Management**]. H.2.5 [**Heterogeneous Databases**]. H.2.6 [**Database Machines**].

General Terms
Management, Measurement, Performance, Experimentation, Verification.

Keywords
Database Management Systems (DBMS), NoSQL Databases.

1. INTRODUCTION
Some years ago, databases appeared as a repository with organized and structured data, where all that data is combined into a set of registers arranged into a regular structure to enable easy extraction of information. To access data is common to use a system usually known as DataBase Management System (DBMS). DBMS can be defined as a collection of mechanisms that enables storage, edit and extraction of data; over past years the concept of DBMS has become a synonym of database. Size and complexity of databases are defined by the number of registers used. A simple database can be represented as a file with data while more complex databases are able to store millions of registers with a huge amount of gigabytes all over the globe. More and more, databases became an important enterprise tool. For the past years with the evolution of Information and Communications Technology, the storage type, functionalities and interaction with databases has improved. Moreover, databases became a resource used every day by millions of people in countless number of applications. All that value and usage created a must to have all data structured and organized in the best way so extraction can be made fast and easy. Whenever quantity of data increases, databases become larger. With the exponential growth of database size, access to data has to be made as more efficient as possible. That leads to the well-known problem of efficiency in information extraction.

Conference C3S2E, July 10–12, 2013, Porto, Portugal.

Edgar Frank Codd introduced the relational model in 1970 publishing a paper in Communications of ACM magazine while working as IBM programmer [2]. As research result, Codd proposed a solution to overcome data storage and usage difficulties according to principles based on relations between data. So, 12 rules were introduced to manage data as relational model, known as "E. F. Codd's 12 rules" [3]. Meanwhile System R [15], experimental database system, was developed to demonstrate usability and advantages of relational model. With it was created a new language, Structured Query Language, known as SQL [6]. Since then, SQL became a standard for data interaction and manipulation. Relational Databases store data as a set of tables, each one with different information. All data is related so it is possible to access information from different tables simultaneously. Relational model is based on "relationship" concept. The origin of relational model was the concept Codd used to define a table with data, he called it "relation". So, basically, a relation is a table organized in columns and rows. Each table is formed by set of tuples with same attributes. Those attributes contain information about some object. More complex database contains a lot of tables with millions of entries. Those tables are connected so data from one table can be related to other by key. There are different types of keys, but essentially there are of two types: primary key and foreign key. Primary key is used to identify each entire table, tuple, as unique. Foreign key is used to cross-reference tables. Foreign key in one table represents a Primary key in the other.

While data volume increases exponentially, some problems became evident. One of those is database performance related to data access and basic structure of relational model. SQL enables easy data extraction but when information volume is huge, query execution time can become slow [10, 12, 18]. Any application with large amount of data will unavoidably lose performance. To overcome those efficacy problems, emerged different types of databases. One of those is known as NoSQL corresponding "NotOnlySQL" [16]. NoSQL was introduced by Carlo Strozzi in 1980 to refer an open source database that wasn't using SQL interface. Carlo Strozzi prefer to refer to NoSQL as "noseequel"

or "NoRel", which is a principal difference between that technology and already existent [13]. The origin of NoSQL can be related to BigTable, model developed by Google [7]. That database system, BigTable, was used to storage Google's projects, such as, Google Earth. Posteriorly Amazon developed his own system, Dynamo [5]. Both of those projects highly contributed for NoSQL development and evolution. However NoSQL term was not popular or known until the meeting held in San Francisco in 2009 [20, 21]. Ever since then, NoSQL became a buzzword.

This paper is focused on testing NoSQL databases and compare performance of two widely used databases, MongoDB and Cassandra. We will describe the main characteristics and advantages of NoSQL databases compared to commonly used relational databases. Some advantages and innovation brought by *noseequel* model and different existing types of NoSQL databases will be discussed. The benchmarking of these two NoSQL databases, MongoDB and Cassandra is also described.

The experimental evaluation of both databases will test the difference in managing and data volume scalability, and verify how databases will respond to read/update mix while running just on one node without a lot of memory and processor resources, just like personal computers. More specifically will be used Virtual Machine environment. It is common to benchmark databases on high processing and with large capabilities clusters, but in our study the main goal is focus on less capacity servers.

The remainder of this paper is organized as follows. Section 2 reviews related work on the topic and Section 3 makes a brief summary of NoSQL databases. Section 4 describes the comparison between MongoDB and Cassandra. Section 5 describes the YCSB – Yahoo! Cloud Serving Benchmark. In section 6 the experimental results obtained in the study are shown. Finally, Section 7 presents our conclusions and suggests future work.

2. RELATED WORK

Performance and functional principles of NoSQL databases has been approached ever since those gained popularity. While analyzing different papers and studies of NoSQL databases two different types of approaches can be defined. The first is focused on compare commonly used SQL databases and NoSQL databases, evaluate and study performance in order to distinguish those two types of databases. The other one consists of comparison only between NoSQL databases. Those studies commonly pick most known NoRel databases and compare their performance. However, both of those comparisons in most cases are focused on analyzing the number of operations per second and latency for each database. While latency may be considered an important factor while working in cluster environment, there is no value for it in a single node study.

Brian F. Cooper et al. analyzed NoSQL databases and MySQL database performance using YCSB benchmark by relating latency with the number of operations per second [4]. In our paper the main focus is to perform studies prioritizing different execution parameters. More specifically our goal is based on relating execution time to the number of records used on each execution. More importantly, all benchmarking is commonly done in high processing and with lots of memory clusters, it is also important to understand how these databases behave in simpler environments and while using just one server.

The main difference of our paper is its goal to study execution time evolution according to increase in database size. Although all different studies performed are important and allow better understanding of capabilities of NoSQL database and how those differ, we consider data volume a very important factor that must be evaluated. At the same time, execution time provides better perception of performance while the number of operations per second may be hard to analyze. At the same time, while examine related work, it is important to notice that there are no much papers discussing performance and benchmarking NoSQL databases. With all aspects defined above, the main aim of our study is to increase the number of analysis and studies available, while focusing on different parameters compared to existing papers.

3. NOSQL DATABASES

The main reason to NoSQL development was Web 2.0 which increased the use and data quantity stored in databases [8, 11]. Social networks and large companies, such as Google, interact with large scale data amount [12]. In order not to lose performance, arises the necessity to horizontally scale data. Horizontal scaling can be defined as an attempt to improve performance by increasing the number of storage units [19]. Large amount of computers can be connected creating a cluster and its performance exceeds a single node unit with a lot of additional CPUs and memory. With increased adherence to social networks, information volume highly increased. In order to fulfill users demands and capture even more attention, multimedia sharing became more used and popular. Users became able to upload and share multimedia content. So, the difficulty to keep performance and satisfy users became higher [19]. Enterprises became even more aware of efficiency and importance of information promptness. The most widely used social network, Facebook, developed by Mark Zuckerberg grew rapidly. With that, meet all requirements of its users became a hard task. It is difficult to define the number of millions of people who use this network at the same time to perform different activities. Internally interaction of all those users with multimedia data is represented by millions of requests to database at the same time. The system must be designed to be able to handle large amount of requests and process data in a fast and efficient way. In order to keep up with all demands as well as keep high performance, companies invest in horizontal scaling [18]. Beyond efficiency, costs are also reduced. It is more inexpensive to have a large amount of computers with fewer resources than build a supercomputer. Relational databases allow to horizontally scaling data but NoSQL provide that in an easier way. This is due to ACID principles and transaction support that is described in the next section. Since data integrity and consistency are highly important for relational databases, communication channels between nodes and clusters would have to instantly synchronize all transactions.

NoSQL databases are designed to handle all type of failures. Variety of hardware fail may occur and system must be prepared so it is more functional to consider those concerns as eventual occurrences than some exceptional events.

In the next sections it will be described the principles of operation, characteristics and different types of NoSQL databases.

3.1 ACID vs BASE

Relational databases are based on a set of principles to optimize performance. Principles used by Relational or NoSQL databases are derived from CAP theorem [11]. According to this theorem, following guarantees can be defined:

- **C**onsistency – all nodes have same data at the same time;
- **A**vailability – all requests have response;
- **P**artition tolerance – if part of system fails, all system won't collapse.

ACID is a principle based on CAP theorem and used as set of rules for relational database transactions. ACID's guarantees are [17]:

- **A**tomic – a transaction is completed when all operations are completed, otherwise rollback[1] is performed;
- **C**onsistent – a transaction cannot collapse database, otherwise if operation is illegal, rollback is performed;
- **I**solated – all transactions are independent and cannot affect each other;
- **D**urable – when commit[2] is performed, transactions cannot be undone.

It is noticeable that in order to have robust and correct database those guarantees are important. But when the amount of data is large, ACID may be hard to attain. That why, NoSQL focuses on BASE principle [17, 20]:

- **B**asically **A**valiable – all data is distributed, even when there is a failure the system continues to work;
- **S**oft state – there is no consistency guarantee;
- **E**ventually consistent – system guarantees that even when data is not consistent, eventually it will be.

It is important to notice, that BASE still follows CAP theorem and if the system is distributed, two of three guarantees must be chosen [1]. What to choose depends of personal needs and database purpose. BASE is more flexible that ACID and the big difference is about consistency. If consistency is crucial, relational databases may be better solution but when there are hundreds of nodes in a cluster, consistency becomes very hard to accomplish.

3.2 Data access

When it comes to data access, data interaction and extraction in NoSQL databases is different. Usual SQL language cannot be used anymore. NoSQL databases tend to favor Linux so data can be manipulated with UNIX commands. All information can be easily manipulated using simple commands as ls, cp, cat, etc. and extracted with I/O and redirect mechanisms. Even though, since SQL became a standard and widely used, there are NoSQL databases where SQL-like query language can be used. For example, UnQL – Unstructured Query language developed by Couchbase [22] or CQL – Cassandra Query language [23].

[1] Operation that returns database to consistent state

[2] Operation that confirms all changes done over database as permanent

3.3 Types of NoSQL databases

With high adherence to NoSQL databases, different databases have been developed. Currently there are over 150 different NoSQL databases. All those are based on same principles but own some different characteristics. Typically can be defined four categories [9]:

- **Key-value Store.** All data is stored as set of key and value. All keys are unique and data access is done by relating those keys to values. Hash contains all keys in order to provide information when needed. But value may not be actual information, it may be other key.

  Examples of Key-value Store databases are: BynamoDB, Azure Table Storage, Riak, Redis.

- **Document Store.** Those databases can be defined as set of Key-value stores that posteriorly are transformed into documents. Every document is identified by unique key and can be grouped together. The type of documents is defined by known standards, such as, XML or JSON. Data access can be done using key or specific value.

  Some examples of Document Store databases are: MongoDB, Couchbase Server, CouchDB, RavenDB.

- **Column-family.** That is the type most similar to relational database model. Data is structured in columns that may be countless. One of projects with that approach is HBase based on Google's Bigtable [24]. Data structure and organization consists of:
  - *Column* – represents unit of data identified by key and value;
  - *Super-column* – grouped by information columns;
  - *Column family* – set of structured data similar to relation database table, constituted by variety of super-columns.

  Structure of database is defined by super-columns and column families. New columns can be added whenever necessary. Data access is done by indicating column family, key and column in order to obtain value, using following structure:

  <columnFamily>.<key>.<column> = <value>

  Examples of Column-family databases: HBase, Cassandra, Accumulo, Hypertable.

- **Graph database.** Those databases are used when data can be represented as graph, for example, social networks.

  Examples of Graph databases: Neo4J, Infinite Graph, InfoGrid, HyperGraphDB.

In the next section we describe the main characteristics of the two popular NoSQL databases under test.

4. MONGODB VS CASSANDRA

In this section we describe MongoDB and Cassandra, which are the databases chosen for analysis and tests. The main characteristics to be analyzed are: data loading, only reads, reads and updates mix, read-modify-write and only updates. Those databases were chosen in order to compare different types of

databases, MongoDB as Document Store and Cassandra as Column family.

4.1 MongoDB

MongoDB is an open source NoSQL database developed in C++. It is a multiplatform database developed in 2007 by 10gen with first public release in 2009, currently in version 2.4.3 and available to download at (http://www.mongodb.org/downloads).

MongoDB is a document store database where documents are grouped into collections according to their structure, but some documents with different structure can also be stored. However, in order to keep efficiency up, similarity is recommended. The format to store documents in MongoDB is BSON – Binary JSON and the maximum size for each is limited to 16MB. The identification is made by defined type, not just id. For example, it can be the combination of id and timestamp in order to keep documents unique. It is important to notice that 32bit MongoDB has a major limitation. Only 2GB of data can be stored per node. The reason of that is memory usage made by MongoDB. In order to increase performance data files are mapped in memory. By default data is sent to disc every 60 seconds but that time can be personalized. When new files are created, everything is flushed to disc, releasing memory. It is not known if the size of memory used by MongoDB can be defined, eventually unused files will be removed from memory by operating system. So, since 64 bit OS are capable of address more memory, 32 bit OS are limited. In order to increase performance while working with documents, MongoDB uses indexing similar to relational databases. Each document is identified by _id field and over that field is created unique index. Although indexing is important to execute efficiently read operations, it may have negative impact on inserts. Apart from automatic index created on _id field, additional indexes can be created by database administrator. For example, can be defined index over several fields within specific collection. That feature of MongoDB is called “compound index”. However, all indexes use the same B-tree structure. Each query use only one index chosen by query optimizer mechanism, giving preference to more efficient index. Eventually query optimizer reevaluates used indexing by executing alternative plans and comparing execution cost.

Some of the most important characteristics of this database are durability and concurrency. Durability of data is granted with creation of replicas. MongoDB uses Master-Slave replication mechanism. It allows defining a Master and one or more Slaves. Master can write or read files while Slave serves as backup, so only reading operations are allowed. When Master goes down, Slave with more recent data is promoted to Master. All replicas are asynchronous, what means that all updates done are not spread immediately. Replica members can be configured by system administrator in a variety of ways, such as:

- Secondary-Only Members. Those replicas store data but cannot be promoted to Master under any circumstances.
- Hidden Members. Hidden replicas may not become primary and are invisible to client applications. Usually those members provide dedicated backups and read-only testing. However, those replicas still vote for new Master when failover occurs and primary unit must be chosen.
- Delayed Members. Replicas that copy primary unit operations with specified delay. Which means that data on replica will be older compared to the Master and will not match last updates done.
- Arbiters. These members exist only to participate in elections and interact with all other members.
- Non-Voting Members. These replicas may not take part in elections and usually are used only with large clusters with more than 7 members.

Starting from version 2.2 MongoDB uses locks to ensure consistency of data and prevent multiple clients to read and update data at the same time. Before, information could be simple replaced while being transferred to memory.

Similarly to RDBMS may be defined four core database operations executed over MongoDB. The set of those operations is called CRUD and stands for Create, Read, Update and Delete. In Figure 1 is shown an example of the MongoDB interface.

Figure 1 – MongoDB interface

Like other NoSQL databases, MongoDB is controlled by UNIX shell but there are some projects that developed an interface, such as, Edda, MongoVision and UMongo [25].

4.2 Cassandra

Cassandra is a NoSQL database developed by Apache Software Foundation written in Java. Cassandra is available as Apache License distribution at (http://cassandra.apache.org/).

Being part of Column-Family, Cassandra is very similar to the usual relational model, made of columns and rows. The main difference were the stored data, that can be structured, semi structured or unstructured.

While using Cassandra, there is a community of support and professional support from some companies. Cassandra is designed to store large amount of data and deal with huge volumes in an efficient way. Cassandra can handle billions of columns and millions of operations per day [26]. Data can be distributed all over the world, deployed on large number of nodes across multiple data centers. When it comes to storage in cluster and nodes, all data is stored over clusters. When some node is added or when it is removed, all data is automatically distributed over other nodes and the failed node can be replaced with no downtime. With that it is no longer necessary to calculate and assign data to each node. Every node in the cluster have same role

which means that there are no master. That architecture is known as peer-to-peer and overcomes master-slave limitations such as, high availability and massive scalability. Data is replicated over multiple nodes in the cluster. It is possible to store terabytes or petabytes of data. Failed nodes are detected by gossip protocols and those nodes can be replaced with no downtime. The total number of replicas is referred as replication factor. For example, replication factor 1 means that there is only one copy of each row on one node but replication factor 2 represents that there are two copies of same records, each one on different node. There are two available replication strategies:

- Simple Strategy: it is recommended when using a single data center. Data center can be defined as group of related nodes in cluster with replication purpose. First replica is defined by system administrator and additional replica nodes are chosen clockwise in the ring.
- Network Topology Strategy: it is a recommended strategy when the cluster is deployed across multiple data centers. Using this strategy it is possible to specify the number of replicas to use per data center. Commonly in order to keep tolerance-fault and consistency it should be used two or three replicas on each data center.

One of the important features of Cassandra is durability. There are two available replication types: synchronous and asynchronous, and the user is able to choose which one to use. Commit log is used to capture all writes and redundancies in order to ensure data durability.

Another important feature for Cassandra is indexing. Each node maintains all indexes of tables it manages. It is important to notice that each node knows the range of keys managed by other nodes. Requested rows are located by analyzing only relevant nodes. Indexes are implemented as a hidden table, separated from actual data. In addition, multiple indexes can be created, over different fields. However, it is important to understand when indexes must be used. With larger data volumes and a large number of unique values, more overhead will exist to manage indexes. For example, having database with millions of clients' records and indexing by e-mail field that usually is unique will be highly inefficient.

All stored data can be easily manipulated using CQL – Cassandra Query Language based on widely used SQL. Since syntax is familiar, learning curve is reduced and it is easier to interact with data. In Figure 2 is shown a Cassandra client console.

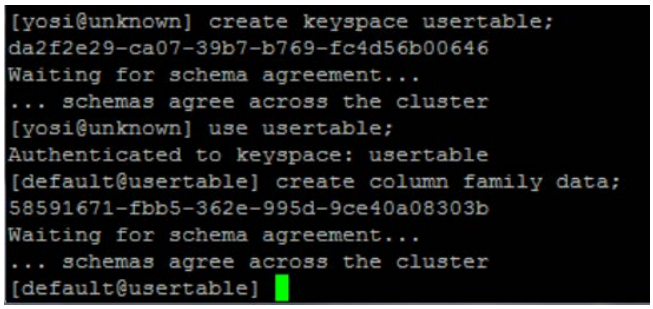

Figure 2 – Cassandra console

There are different ways to use Cassandra, some of most prominent areas of use are: financial, social media, advertising, entertainment, health care, government, etc. There are many companies that use Cassandra, for example, IBM, HP, Cisco and eBay [24].

4.3 Features comparison

In order to better understand differences between MongoDB and Cassandra we study some features of those NoSQL databases such as: development language, storage type, replication, data storage, usage and some other characteristics. All those characteristics are shown in Table 1.

Table 1. MongoDB and Cassandra features

	MongoDB	**Cassandra**
Development language	C++	Java
Storage Type	BSON files	Column
Protocol	TCP/IP	TCP/IP
Transactions	No	Local
Concurrency	Instant update	MVCC
Locks	Yes	Yes
Triggers	No	Yes
Replication	Master-Slave	Multi-Master
CAP theorem	Consistency, Partition tolerance	Partition tolerance, High Availability
Operating Systems	Linux / Mac OS / Windows	Linux / Mac OS / Windows
Data storage	Disc	Disc
Characteristics	Retains some SQL properties such as query and index	A cross between BigTable and Dynamo. High availability
Areas of use	CMS system, comment storage	Banking, finance, logging

By analyzing core properties it is possible to conclude that there are similarities when it comes to used file types, querying, transactions, locks, data storage and operating systems. But it is important to notice the main difference, according to CAPs theorem, MongoDB is CP type system – Consistency and Partition tolerance, while Cassandra is PA – Consistency and Availability. In terms of replication, MongoDB uses Master-Slave while Cassandra uses peer-to-peer replication that is typically named as Multi-master.

In terms of usage and best application, MongoDB has better use for Content Management Systems (CMS), while having dynamic queries and frequently written data. Cassandra is optimized to store and interact with large amounts of data that can be used in different areas such as, finance or advertising. Following, we describe the benchmark to test MongoDB and Cassandra databases.

5. YCSB BENCHMARK

The YCSB – Yahoo! Cloud Serving Benchmark is one of the most used benchmarks to test NoSQL databases [10]. YCSB has a client that consists of two parts: workload generator and the set of scenarios. Those scenarios, known as workloads, are combinations of read, write and update operations performed on randomly chosen records. The predefined workloads are:

- **Workload A: Update heavy workload.** This workload has a mix of 50/50 reads and updates.
- **Workload B: Read mostly workload.** This workload has a 95/5 reads/update mix.
- **Workload C: Read only.** This workload is 100% read.
- **Workload D: Read latest workload.** In this workload, new records are inserted, and the most recently inserted records are the most popular.
- **Workload E: Short ranges.** In this workload, short ranges of records are queried, instead of individual records.
- **Workload F: Read-modify-write.** In this workload, the client will read a record, modify it, and write back the changes.

Because our focus is on update and read operations, workloads D and E will not be used. Instead, to better understand update and read performance, two additional workloads were defined:

- **Workload G: Update mostly workload.** This workload has a 5/95 reads/updates mix.
- **Workload H: Update only.** This workload is 100% update.

The loaded data is from a variety of files, each one with a certain number of fields. Each record is identified by a key, string like "user123423". And each field is named as *field0*, *field1* and so on. Values of each field are random characters. For testing we use records with 10 fields each of 100 bytes, meaning a 1kb per record.

Since client and server are hosted on the same node, latency will not take part of this study. YCSB provides thread configuration and set of operation number per thread. During initial tests we observed that using threads, the number of operations per second actually reduced. That is due to the fact that tests are running on virtual machine with even lower resources than a host.

6. EXPERIMENTAL EVALUATION

In this section we will describe the experiments while using different workloads and data volumes. Tests were running using Ubuntu Server 12.04 32bit Virtual Machine on VMware Player. As experimental setup it is important to notice that VM has available 2GB RAM and Host was single-node Core 2 Quad 2.40 GHz with 4GB RAM and Windows 7 Operating System. The tested versions of NoSQL databases are MongoDB version 2.4.3 and Cassandra version 1.2.4.

As focus of study, we take the execution time to evaluate the best database performance. All workloads were executed three times with reset of computer between tests. All the values are shown in (minutes:seconds) and represent the average value of the three executions.

In the following figures we show data loading phase tests and time execution for the different types of workloads: A, B, C, F, G, and H.

Data loading phase

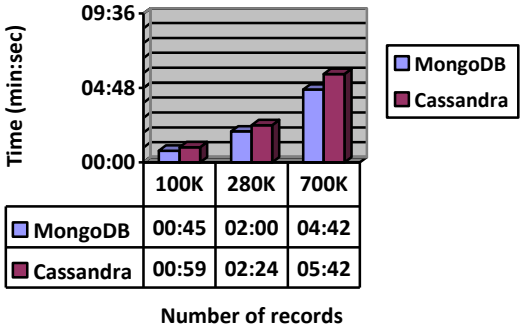

Figure 3 - Data loading test

To compare loading speed and throughput different volumes of were loaded with 100.000, 280.000 and 700.000 records as shown in Figure 3. While observing results, it is possible to see that there was no significant difference between MongoDB and Cassandra. MongoDB had slightly lower insert time, regardless of number of records, compared to Cassandra, which has an average overhead of 24%. When the size of loaded data increases, the execution time increased in a similar proportion for both databases with highest time of 04:42 for MongoDB and 05:42 for Cassandra when inserting 700.000 records.

Workload A (50/50 reads and updates)

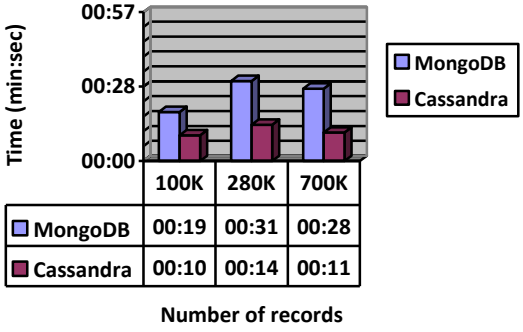

Figure 4 - Workload A experiments

Compared to MongoDB, Cassandra had better execution time regardless database size. The performance of Cassandra can be 2.54 times faster than Mongo DB using a mix of 50/50 reads and updates with 700.000 records. Another important fact that can be observed is the decrease in time execution when number of records used goes from 280.000 up to 700.000, for both databases (see Figure 4). This happens due to optimization of databases to work with larger volumes of data.

Workload B (95/5 reads and updates)

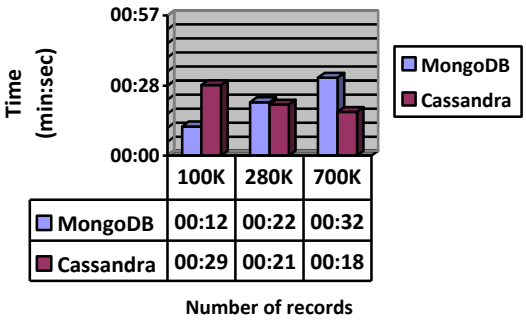

Figure 5 - Workload B experiments

When we test the databases with a 95/5 reads/update mix the results for Cassandra and MongoDB had different behavior as shown in Figure 5. While execution time for MongoDB kept increasing, Cassandra was able to reduce time while data volume became larger. However, the highest time for Cassandra was 00:29 and corresponds to querying over 100.000 records when for MongoDB highest time was of 00:32 for 700.000 records. The performance of Cassandra with this workload is 56% better when comparing to MongoDB, using 700.000 records. Although for small size data (100.000 records) the MongoDB has better results.

Workload C (100% reads)

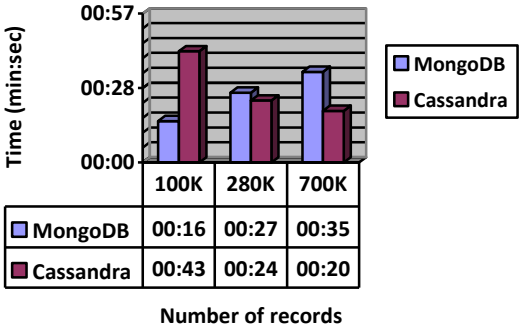

Figure 6 - Workload C experiments

In this workload we have 100% of reads. As the previous experiments, when it comes to large amount of read operations, Cassandra becomes more efficient with bigger quantity of data, as illustrated in Figure 6. MongoDB showed similar behavior of the previous workload, where execution time is directly proportional to data size. However, MongoDB is 2.68 faster when using 100.000 records but 1.75 slower for 700.000 records, when comparing to Cassandra execution time. Fastest execution time of MongoDB is 00:16 and for Cassandra is 00:20, however those results represent opposite volumes of data, being better execution time for Cassandra with high number of records and for MongoDB with just 100.000 records.

Workload F (read-modify-write)

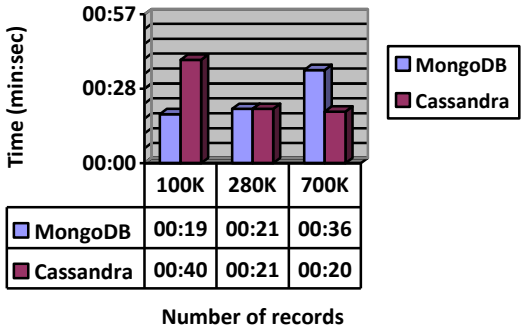

Figure 7 - Workload F experiments

In this workload, the client will read a record, modify it, and write back the changes. In this workload Cassandra and MongoDB showed opposite behavior results as illustrated id Figure 7. The Cassandra's higher execution time was with small data volume and with increase it kept reducing while MongoDB has worst time with bigger data size. MongoDB is 2.1 faster for querying over 100.000 records but 1.8 slower for 700.000 records, and have the same value for 280.000 records, when comparing to Cassandra execution time. Smallest execution time variations were 00:01 for Cassandra when increasing number of records from 280.000 up to 700.000 and 00:02 for MongoDB when lowing number of records used from 280.000 down to 100.000 records.

Workload G (5/95 reads and updates)

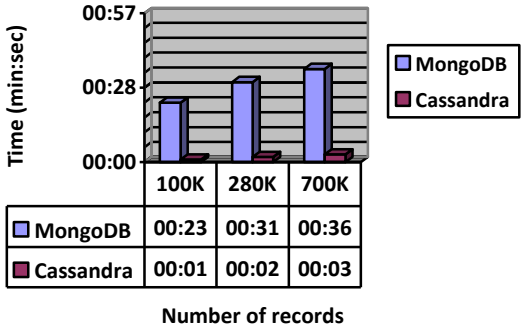

Figure 8 - Workload G experiments

This workload has a 5/95 reads/updates mix. The results shown in Figure 8 are absolutely demonstrative of the superiority of Cassandra over MongoDB for all database sizes. On every execution time Cassandra showed better results. With grown of data volume both Cassandra and MongoDB started having higher execution time, but MongoDB was not even close to Cassandra. The performance of Cassandra with this workload varies from 23 to 12 times faster than MongoDB. That established difference in performance allows us to conclude that in this environment, Cassandra is more optimized to update operations compared to MongoDB, showing surprisingly high performance results.

Workload H (100% updates)

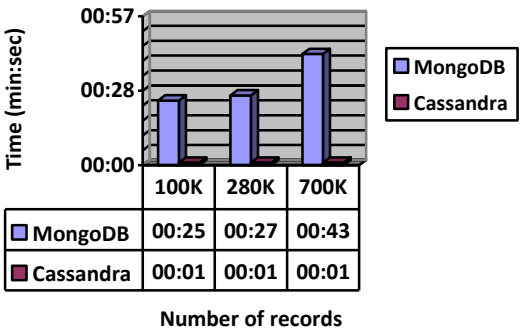

Figure 9 - Workload H experiments

When it came to a 100% update workload Cassandra had stable performance even with increased number of records, as shown in Figure 9. Similarly to results of workload G, Cassandra showed great results compared to MongoDB, which varies from 25 to 43 times better. For MongoDB the difference in execution time between 100.000 records and 280.000 records was not big but almost doubled when using 700.000 records

7. CONCLUSIONS AND FUTURE WORK

The development of the Web need databases able to store and process big data effectively, demand for high-performance when reading and writing, so the traditional relational database is facing many new challenges. NoSQL databases have gained popularity in the recent years and have been successful in many production systems. In this paper we analyze and evaluate two of the most popular NoSQL databases: MongoDB and Cassandra. In the experiments we test the execution time according to database size and the type of workload. We test six different types of workloads: mix of 50/50 reads and updates; mix of 95/5 reads/updates; read only; read-modify-write cycle; mix of 5/95 reads/updates; and update only. With the increase of data size, MongoDB started to reduce performance, sometimes showing poor results. Differently, Cassandra just got faster while working with an increase of data. Also, after running different workloads to analyze read/update performance, it is possible to conclude that when it comes to update operations, Cassandra is faster than MongoDB, providing lower execution time independently of database size used in our evaluation. As overall analysis turns out that MongoDB fell short with increase of records used, while Cassandra still has a lot to offer. In conclusion Cassandra show the best results for almost all scenarios.

As future work, we pretend to analyze the number of operations per second vs database size. That would help to understand, how those databases would behave with higher number of records to read/update with data volume grown.

8. AKNOWLEDGMENTS

Our thanks to ISEC – Coimbra Institute of Engineering from Polytechnic Institute of Coimbra for following us to use the facilities of the Laboratory of Research and Technology Innovation of Computer Science and Systems Engineering Department.

9. REFERENCES

[1] Brewer, E., "CAP twelve years later: How the "rules" have changed," Computer , vol.45, no.2, pp.23,29, Feb. 2012. doi: 10.1109/MC.2012.37.

[2] Codd. E. F. 1970. A relational model of data for large shared data banks. Communications of ACM 13, 6 (June 1970), 377-387. doi=10.1145/362384.362685.

[3] Codd. E. F. 1985. “Is your DBMS Really Relational?” and “Does your DBMS Run by the Rules?” *Computer World*, October 14 and October 21.

[4] Cooper B. F., Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. 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. DOI=10.1145/1807128.1807152 http://doi.acm.org/10.1145/1807128.1807152

[5] DeCandia Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. 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.

[6] Donald D. Chamberlin, Raymond F. Boyce: SEQUEL: A Structured English Query Language. SIGMOD Workshop, Vol. 1 1974: 249-264.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (OSDI '06), Vol. 7. USENIX Association, Berkeley, CA, USA, 15-15.

[8] Hecht, R.; Jablonski, S., "NoSQL evaluation: A use case oriented survey," Cloud and Service Computing (CSC), 2011 International Conference on , vol., no., pp.336,341, 12-14 Dec. 2011. doi: 10.1109/CSC.2011.6138544.

[9] Indrawan-Santiago, M., "Database Research: Are We at a Crossroad? Reflection on NoSQL," Network-Based Information Systems (NBiS), 2012 15th International Conference on , vol., no., pp.45,51, 26-28 Sept. 2012. doi: 10.1109/NBiS.2012.95.

[10] Jayathilake, D.; Sooriaarachchi, C.; Gunawardena, T.; Kulasuriya, B.; Dayaratne, T., "A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree," Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on , vol., no., pp.106,111, 27-29 Sept. 2012. doi: 10.1109/ICIAFS.2012.6419890.

[11] Jing Han; Haihong, E.; Guan Le; Jian Du, "Survey on NoSQL database," Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on , vol., no., pp.363,366, 26-28 Oct. 2011. doi:10.1109/ICPCA.2011.6106531.

[12] Leavitt, N., "Will NoSQL Databases Live Up to Their Promise?," Computer , vol.43, no.2, pp.12,14, Feb. 2010. doi: 10.1109/MC.2010.58.

[13] Lith, Adam; Jakob Mattson (2010). "Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data". Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology.

[14] Lombardo, S.; Di Nitto, E.; Ardagna, D., "Issues in Handling Complex Data Structures with NoSQL Databases," Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on , vol., no., pp.443,448, 26-29 Sept. 2012. doi: 10.1109/SYNASC.2012.59.

[15] M.M. Astrahan, A history and evaluation of system R, Performance Evaluation, Volume 1, Issue 1, January 1981, Page 95, ISSN 0166-5316, 10.1016/0166-5316(81)90053-5.

[16] nosql-database.org, accessed on 30th April 2013.

[17] Roe C. 2012 “ACID vs. BASE: The Shifting pH of Database Transaction Processing” - http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/.

[18] Shidong Huang; Lizhi Cai; Zhenyu Liu; Yun Hu, "Non-structure Data Storage Technology: A Discussion," Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on , vol., no., pp.482,487, May 30 2012-June 1 2012. doi: 10.1109/ICIS.2012.76.

[19] Silberstein, A.; Jianjun Chen; Lomax, D.; McMillan, B.; Mortazavi, M.; Narayan, P. P S; Ramakrishnan, R.; Sears, R., "PNUTS in Flight: Web-Scale Data Serving at Yahoo," Internet Computing, IEEE , vol.16, no.1, pp.13,23, Jan.-Feb. 2012. doi: 10.1109/MIC.2011.142.

[20] Tudorica, B.G.; Bucur, C., "A comparison between several NoSQL databases with comments and notes," Roedunet International Conference (RoEduNet), 2011 10th , vol., no., pp.1,5, 23-25 June 2011. doi:10.1109/RoEduNet.2011.5993686.

[21] Yahoo! Developer Network 2009. Notes from NoSQL Meetup. - http://developer.yahoo.com/blogs/ydn/notes-nosql-meetup-7663.html.

[22] http://www.couchbase.com/press-releases/unql-query-language, accessed on 30th April 2013

[23] http://www.datastax.com/docs/1.0/references/cql/index, accessed on 30th April 2013.

[24] http://cassandra.apache.org/, accessed on 30th April 2013.

[25] http://docs.mongodb.org/ecosystem/tools/administration-interfaces/, accessed on 30th April 2013.

[26] http://www.datastax.com/what-we-offer/products-services/datastax-enterprise/apache-cassandra, accessed on 30th April 2013.