

**Research Report**

**Benchmarking the performance of NoSQL DBMSs**

**Case Study: Cassandra Vs Redis Vs MongoDB**

Author: Bertocco Thomas-Killian - #13758322

Lecturer: Dr. Jeyaseelan Thirunavukarasu

CP5520 Advanced Databases and Applications | SP53 | 2019

26 January 2020

# Abstract

Nowadays, due to the increasing use of web-based programs, smartphones and social networks, game consoles, ... the production of information by users increases drastically. It seems that the transition from rational databases to non-relational databases is a necessity. Regarding the different use cases and their requirements, as well as the main features of NoSQL databases, the choice of the most appropriate NoSQL database can be a great concern for developers. The correct choice of NoSQL databases will avoid wasting time, money and energy. In this paper, we present an experimental evaluation of the three most popular NoSQL databases: MongoDB, Redis and Cassandra (DBENGINE,2020).

We provide a set of criteria and indicators for stakeholders to make decisions about the solutions adopted for their companies.

Table of Contents

[Abstract 2](#_Toc30939796)

[Introduction 5](#_Toc30939797)

[The NoSQL 6](#_Toc30939798)

[Introduction 6](#_Toc30939799)

[The CAP Theorem 6](#_Toc30939800)

[BASE 7](#_Toc30939801)

[NoSQL 7](#_Toc30939802)

[Features 7](#_Toc30939803)

[Types 8](#_Toc30939804)

[The use of NoSQL 8](#_Toc30939805)

[Column Oriented Databases 8](#_Toc30939806)

[Advantages 9](#_Toc30939807)

[Document-Oriented Databases 9](#_Toc30939808)

[Advantages 9](#_Toc30939809)

[Key-Value-Oriented Databases 9](#_Toc30939810)

[Advantages 10](#_Toc30939811)

[Graph oriented databases 10](#_Toc30939812)

[Advantages 10](#_Toc30939813)

[List of NoSQL databases 11](#_Toc30939814)

[Hybrid NoSQL databases 11](#_Toc30939815)

[Which NoSQL DBMS should I use? 11](#_Toc30939816)

[Transactional applications 11](#_Toc30939817)

[Calculation applications 11](#_Toc30939818)

[Applications web 12](#_Toc30939819)

[Cassandra, Redis, MongoDB, and YCSB 12](#_Toc30939820)

[Introduction 12](#_Toc30939821)

[Redis 12](#_Toc30939822)

[Presentation 12](#_Toc30939823)

[The data model 12](#_Toc30939824)

[Characteristics 13](#_Toc30939825)

[Cassandra 14](#_Toc30939826)

[Presentation 14](#_Toc30939827)

[The data model 14](#_Toc30939828)

[Characteristics 15](#_Toc30939829)

[MongoDB 16](#_Toc30939830)

[Presentation 16](#_Toc30939831)

[Le sharding 17](#_Toc30939832)

[Characteristics 17](#_Toc30939833)

[YCSB 18](#_Toc30939834)

[Workloads 18](#_Toc30939835)

[Conclusion 19](#_Toc30939836)

[Implementation and experimentation 19](#_Toc30939837)

[Introduction 19](#_Toc30939838)

[The experimentation 19](#_Toc30939839)

[The commands 20](#_Toc30939840)

[Start databases 20](#_Toc30939841)

[Additional set up for Cassandra 20](#_Toc30939842)

[Load a set of data 21](#_Toc30939843)

[Run operations 21](#_Toc30939844)

[RESULTS 22](#_Toc30939845)

[Workload A (50% Read, 50% Update) 22](#_Toc30939846)

[Workload B (95% Read, 5% Update) 24](#_Toc30939847)

[Workload C (100% Read) 25](#_Toc30939848)

[Wokload D (5% Insert, 95% Read) 27](#_Toc30939849)

[Workload E (95% Scan, 5% Insert) 28](#_Toc30939850)

[Workload F (50% Read, 50% Read-Modify-Write) 30](#_Toc30939851)

[Overall evaluation of MongoDB, Cassandra, and Redis 31](#_Toc30939852)

[General Conclusion 32](#_Toc30939853)

[References 33](#_Toc30939854)

# Introduction

Since the beginning of the 21st century, the Internet has become accessible to billions of users, and since then, a new term has become known in computing, non-relational databases, or what is commonly known as NoSQL. More and more large companies handling large volumes of unstructured data have adopted NoSQL. As a result, many developers have started to move towards this new product and offer adequate and futuristic solutions. Relational databases were born in an era of personal computers and business applications, even before the arrival of the Internet, the Cloud, Big Data, smartphones, ... these relational databases such as Oracle were designed to work on a single powerful server. To increase the capacity of a database, the only solution is to increase the capacity of the processor, RAM or hard disk. The problem is that this increase has a threshold, a limit!

Currently, many users of classic DBMS called "SQL" want to switch to the new NoSQL technologies. On the other hand, the appearance of different NoSQL models that provide new functionalities on the one hand and the lack of standardization of the solutions available on the market, on the other hand, imposes a very relevant question: which NoSQL solution to adopt?

The objective of this work is to define the term NoSQL, why it existed, explain the architecture of its famous databases available on the market and what its expectations are. We will address the definitions of the main concepts of NoSQL and explore the different types of these best-known databases; their advantages and shortcomings. Finally, a comparative experimental study of three NoSQL databases, namely; Mongo DB Redis and Cassandra, and their performance with technical parameters, to arrive at a useful performance analysis using YCSB, a tool well known for its testing power and used in several NoSQL database evaluation works. We chose these three databases because they are the three most popular databases of their kind (document-oriented, key-value oriented, column-oriented).

We will not test any graph-oriented database, as they should not be evaluated according to the scenarios used in the analysis of other types of NoSQL databases (column-oriented, document-oriented, and value-key oriented). The use of links between records requires a different approach, so there are specific benchmarks developed to evaluate the performance of graph databases such as XGDBench (Armstrong, 2013).

# The NoSQL

## Introduction

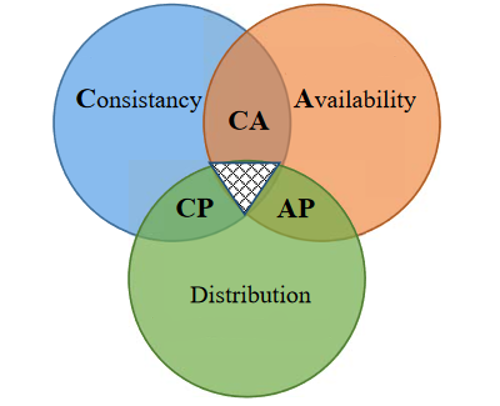
In this chapter we will talk about the CAP theorem which was the basis of the research that led to the appearance of the NoSQL; then the notion of the BASE, the derivative of the theorem.

## The CAP Theorem

The principle of this theorem is the following: any distributed database must theoretically respect three constraints (Brewer, 2000):

* Consistency: A data item has only one visible state regardless of the number of replicas.
* Availability: A data must always be available, whatever the state of the system.
* Partition Tolerance: A query must provide a correct result regardless of the number of servers.

Whereas according to Brewer's theorem, practically, at a given time, two constraints at the same time can be guaranteed by a database.



The three options that may exist are:

* Marginalize Distribution Tolerance (DC): The system does not consider the distribution of data over a network. This is the typical case of RDBMSs.
* Marginalizing consistency (AP): In the case of distribution, the data can be requested, but because of node breakage, consistency is not guaranteed, because updates are asynchronous on the network. This option is interested in providing a fast response time.
* Marginalize Availability (CP): Data can only be used if its consistency is guaranteed. An updated data on one node must be blocked on the other nodes until the new version is propagated throughout the network. In a distributed environment, a database takes a considerable amount of time to have a consistent state, which makes relative availability.

The option to marginalize the distribution is not realistic, because nowadays it is not practical or even unimaginable to work in an undistributed environment.

## BASE

According to the conclusion of the previous paragraph, there remains the choice between consistency represented by ACID and availability represented by BASE (Basically Available Soft-state Eventually consistent), the term coined by Brewer and his successors to refer to the case of possible consistency.

* Available: A response is guaranteed for each request, whether successful or unsuccessful, regardless of the database load.
* Soft-state: The state of the system can change during updates, the base is not necessarily consistent at all times.
* Eventually, consist: The database may be inconsistent temporarily, but it must be consistent eventually.

According to Brewer, the choice is not binary, one can have a proportional mix of consistency and availability levels for better results. In 2005, Google was the largest site on the web, and relational DBMSs showed a glaring insufficiency to cope with the increase in speed and data volumes. The Big Data challenges that businesses face today, Google already met them almost 20 years ago. (Harrison, 2015).

## NoSQL

NoSQL refers to a family of database management systems (DBMS) that departs from the classical paradigm of relational databases. The most popular explanation of the acronym is "Not Only SQL". The data are not relational, and the SQL language is not used for data definition and manipulation. This is the impact of the fact that everyone has begun to focus on creating web-wide databases, i.e. meeting the growing needs of hundreds of millions of users and now increasingly billions of connected devices, including but not limited to mobiles, smartphones, internet TV, and many more. Nowadays, NoSQL is the term designated to classify DBs that are not based on the classic relational DBMS model of ACID nature, and which is - NoSQL - invented specifically to manage the speed and extension of web data such as Google, Facebook, Yahoo, Twitter and others.

NoSQL DBMSs are trying to find solutions to the problems of scalability and data availability despite the atomicity and consistency involved by relational DBMSs with their ACID transactions. Unlike relational DBMS, data units are not well defined in terms of type, size and other constraints. There is no notion of columns and rows where the columns have relationships, and there are no ACID transactions. Let us imagine a user who wants to buy a book from the Amazon site and the transaction is atomic, which means that other users have to wait until the end of the transaction to be able to launch their orders for inventory reasons; which is impossible given the number of orders placed instantly. Amazon uses cached data and even unlocked records, which directly affects consistency. It should be noted that in the case of banking transactions, it is impossible to work with cached data or with a temporary state due to the criticality (critical nature) of the situation., NoSQL is not always a solution.

### Features

NoSQL is not just to deal with the problem of scalability, but it has also proposed other solutions :

* A data representation without a schema: A new technician recruited within a company must first know the entire model of an existing relational database, i.e. the entities, their relationships and the code in order to be able to maintain it. On the other hand, with a NoSQL DB, the representation is more flexible, we do not have to think about the rigid structure of the DB, it can evolve during its development, including the addition of new fields and data merging (JSON) and even possible integration with other heterogeneous platforms.
* No more complex queries: No long SQL queries and complex joins, which significantly reduces development time and optimizes resource utilization.
* Speed: Gain user confidence through the speed of response (we talk about milliseconds instead of hundreds of milliseconds)
* Updating the data: With relational DBMS, updating data through linked or cross tables is a complex scenario, especially if the relevant table is not part of the current transaction, it leaves the transaction open for a long time and slows down performance. On the other hand, NoSQL with the principle of "Eventually Consistent" allows concurrent updates across several Datacenters with conflict management and in a very acceptable time frame
* Planned scalability: We do not have to worry about the size of the application, the volume of data or the data schema at the design stage. For example, the support of dozens of orders at the same time on the same product by geographically distributed customers cannot be met by a traditional DBMS.

### Types

NoSQL DBs are classified according to the data storage model: Document-oriented, key-value oriented, column-oriented, graph-oriented.

### The use of NoSQL

The domains most invaded by NoSQL comics are:

* SaaS applications (CRM-ERP) flexible schema: the document-oriented
* E-learning: Column-oriented
* Social applications (possibility of integration) document-oriented with column-oriented
* Relations (Twitter): The graph-oriented

## Column Oriented Databases

Unlike RDBMS, they serialize the values of one column together, then the values of the next column, etc. A relational DB presents the data in a two-dimensional table composed of rows and columns, but manipulates them row by row, whereas column-oriented NoSQL stores the data as columns.

It is advisable to start modeling in column-oriented NoSQL because it allows a quick understanding of the model and gives a taste of this new domain. If the work requires aggregated values, you have no choice but to use it.

### Advantages

* It allows easy insertion of new columns at any time without worrying about defaults. This provides greater flexibility to the model and facilitates scaling.
* The performance of this model is clearly shown in the calculation of maximum, minimum, average and sum values.
* If new values are to be applied to all rows or a subset of columns, this model allows partial access to the data with no effect on the data not affected, thus speeding up execution.
* Optimization of storage space thanks to uniform column types, which are mostly strings of the same size. Such a feature (e.g. China as a country for 1 billion users) optimizes data compression. Document-oriented DBs The records (the RDBMS rows) are represented by documents; they are semi-structured compared to the rigid relational representation and allow data insertion, querying and manipulation.

## Document-Oriented Databases

Two records can have different structures or sets of columns. Records may not conform to a specific schema or table definition, which means that there is no document validation against a schema as there is for RDBMS. In short, a document-oriented DB provides dynamic flexibility or a modifiable or complete schema for documents without a schema. This advantage has made this model more responsive and more used among other NoSQL DB models. This model uses JSON, which is one of the languages that adopt document orientation.

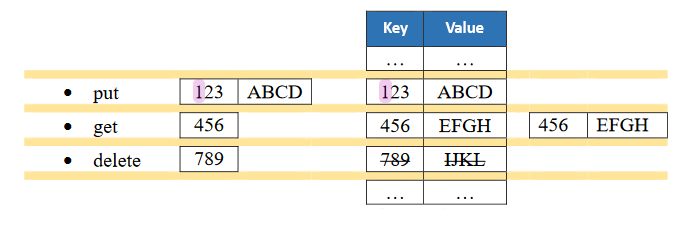
If we need to work with aggregations across multiple entities, this model gives us effective control over how we query the data. For example, working with JSON through merged data or with XQuery using XML and getting custom views. The best known DBs in this model are MongoDB, CouchDB, Jackrabbit, Lotus Notes, Terrastore, Redis and BaseX.

### Advantages

* Content is flexible without schema
* Searching through multiple entities is negligible compared to a single search in a traditional RDBMS.

## Key-Value-Oriented Databases

This model is similar to the document-oriented model, but the creation of the key is mandatory when creating the pair, it allows the data to be stored in key-value pairs. No schema or typing. The definition of the key is obligatory while the value is opaque, for this, a key must be known to find the associated value. Its advantage is the efficiency of working in distributed memory or in a cache to reduce I/O operations. Access to a value is direct and efficient because a key-value pair is unique; the complexity of its search algorithm is O (Lokesh, 2015). Keys can be indexed for better performance. The difference from a document-oriented DB is the absence of queries against values. The data is queried exclusively against keys. Peers can be imagined as a table with two entries:



Most of the key-value oriented databases are inspired by Amazon's Dynamo, which guarantees exceptional scalability and availability. Voldemort and Riak are the implementations of the Dynamo foundation.

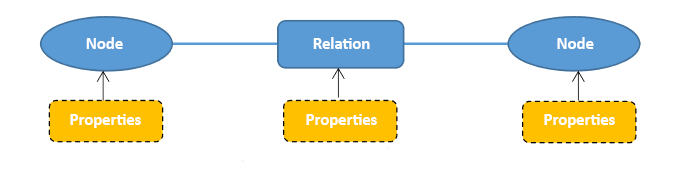
### Advantages

* The search is optimal due to the use of keys and cache. For example, Redis running on an ordinary range microcomputer can scan up to 1 million keys in less than 40 milliseconds.
* The data type of the values is not specified, any type of data can be stored.

This model is not designed for applications requiring indexing of values.

## Graph oriented databases

This model is based on graph theory, i.e. nodes, relationships and properties. It is relatively new on the NoSQL market. Its particularity is the ease in defining the relationships directly at the DB level unlike other models where the relationships are visible at the application level. It is very useful for all applications that have complex relationships between their objects such as social networks.



It benefits from all the advantages of graph theory:

* Finding the shortest way
* Find the neighbors of a node that have specific properties
* What is the resemblance between two nodes taking into account their neighbors, etc.? The W3C uses this model in its semantic web representation language, RDF.

### Advantages

* This model is ideal when we have several objects linked to each other in a complex way, and its objects have properties (brother of, sister of, father of, ...). It allows by a simple request to have the neighbors of a node or to have a whole path by more or less complex requests.
* It does not stop at the point of giving us the relationships between nodes, but also detailed reports on the nature of these relationships.
* As any modeling based on a graphical representation, the major advantage is the easy comprehension by humans compared to a textual modeling, it is a representation of the real world, names, cities, workstations (workstations) of a computer network, ...; a simple mouse click does the insertion and deletion of the relationships between nodes.

## List of NoSQL databases

Below is a list of the famous NoSQL databases that are the most mature and powerful on the market according to their data models:

* Document oriented: MongoDB, CouchDB, RavenDB et Terrastore
* Key-value: Redis, Membase, Voldemort et MemcacheDB
* Columns oriented: Cassandra, BigTable, SimpleDB et Cloudera
* Graph oriented: Neo4J, FlockDB et InfiniteGraph

Most of these databases are open source and Community Driven. Interested companies can start with a relational DBMS and then migrate to a NoSQL; or better yet, start directly with a NoSQL, and it would be best to mix the relational DBMS with the NoSQL. "The example is Netflix, the company migrated from Oracle to Cassandra, it reaches one million writes per second while maintaining the average response time at 0.015 milliseconds, with a total cost of installation and configuration on the Amazon EC2 Cloud of about $60 per hour for a cluster of 48 nodes and a storage capacity of 12.8 Terabytes, the bandwidth is 22 Mbps reading and 18.6 Mbps writing. » (Vaish, 2013)

## Hybrid NoSQL databases

There are other NoSQL DBs that support different types of storage, such as :

* OrientDB: document-oriented, key-value oriented and graph-oriented.
* ArangoDB: a universal database that can be document-oriented, key-value oriented or graphical oriented at the same time.
* Aerospike: a hybrid DB between RDBMS and NoSQL, it is document-oriented, key-value oriented, graphical oriented and also an RDBMS.

## Which NoSQL DBMS should I use?

The choice of a NoSQL DB depends strongly on the nature of the application to be developed.

### Transactional applications

For this type of applications, the column-oriented can be useful in defining a structure that evolves. Document oriented can also be useful in the implementation of joins or the creation of views. Nevertheless, there is no definition of relationships, nor of the notion of a transaction, except the Neo4j, nor of ACID properties in transactions. The support of joins and views via the document-oriented approach is a burden on the complexity of the requests. It is clear that for such applications, the choice of RDBMS is more logical.

### Calculation applications

Column-oriented can help to define a rigorous structure, so document-oriented can be used; both can provide speed and scalability by processing partial data. Document oriented can be used in joins and view creation. On the other hand, apart from the graph-oriented, the creation of relationships can be a headache; they can be created and maintained at the application level but the data will not be consistent.

The factors of speed and scalability give great advantages to NoSQL where data can be partitioned horizontally and vertically.

### Applications web

Document Oriented is an ideal choice for a flexible and scalable schema, it can provide the desired scalability because it does not implement ACID transactions.

# Cassandra, Redis, MongoDB, and YCSB

## Introduction

In this chapter, we will focus on the two technologies of NoSQL: Cassandra and MongoDB. The reason we have chosen these databases is that Cassandra is the most popular type of column-oriented database available on the market. Cassandra is widely used, especially by Facebook, it is an open-source program and can be run on multiple platforms, which makes it a column-oriented database of choice. Similarly, MongoDB is a document-oriented database widely used. It implements the functionalities of the document. It is also open source.

Finally, we will present the YCSB, the tool that we will test their performances with, and we will talk about the workloads that must be injected in these databases to measure their performance.

## Redis



### Presentation

Redis, which comes from Remote Dictionary Server, is a very high-performance scalable key/value NoSQL database, written in C, distributed under BSD license and developed by Salvatore Sanfilippo.

One of the main features of Redis is to keep all the data in RAM. This allows for obtaining excellent performances by avoiding disk accesses, which are particularly costly in this respect.

When the size of the data is too large to hold in memory, Redis can also use virtual memory.

Redis is currently used by companies such as The Guardian, GitHub, and Blizzard Entertainment (Heinrich, 2012).

### The data model

Redis is a key/value type database designed to offer high performance by running entirely in RAM. Of course, the first difficulty arises; it is the risk of loss of data in case of power failure. In order to solve this problem, Redis offers two functions (Heinrich, 2012) :

* RDB: will write a file to the disk containing the entire database in memory. The transport of this database is thus facilitated. However, this is not suitable for recording a transaction. It will be necessary to make a record of the database to save the desired transaction. One can thus understand that it is not recommended to make anything else than periodic complete backups (Furrer, 2015).
* AOF: It is a log in which transactions are saved in a repeated and determined time (Furrer, 2015).

### Characteristics

**data replication**

Replication is the process of creating master-slave cache nodes. Slave nodes always listen to the master node, which implies that when the master node is updated, the slaves are automatically updated as well. Redis can also update the slaves asynchronously.

**Pub/Sub**

Redis supports the use of publication and subscription (Pub/Sub) commands, so users can create powerful chat and messaging services for all their applications and services. This includes the ability to use list data structures to execute atomic operations and blocking capabilities.

**Redis allows inserting huge amounts of data into its cache very easily**

Sometimes, it is required to load millions of pieces of data into the cache within a short period. This can be done quickly using mass insertion, a feature supported by Redis.

**Redis support transaction**

Redis supports transactions, which implies that commands can be performed as a queue instead of being run one at a time. In most cases, commands after MULTI will be added to a queue and once EXEC is issued, all commands in the queue will be executed at once.

**Lua scripting**

The majority of the tasks you perform in Redis have many steps. Rather than performing these steps in the language of your request, you can do them inside Redis with Lua.

* This may result in better performance.
* In addition, all the steps of a script are executed in an atomic way. No other Redis commands can be performed while a script is running.

**Keys with a limited time-to-live**

Set a timeout on the key. Once the timeout expires, the key will be automatically removed. A key with an associated timeout is usually described as volatile in Redis terminology.

**LRU eviction of keys**

When Redis is used as a cache, it is usually helpful to allow it to expel old data when you add new data automatically. This behavior is well known in the developer community since it is the default behavior of the popular Memcached system.

**Automatic failover**

Redis Sentinel ensures high availability for Redis. Concretely, this means that using Sentinel, you can create a Redis deployment that is resistant to certain types of malfunctions without human intervention.

If a master does not work as expected, Sentinel can initiate a failover process where one replica is promoted to master, additional replicas are reconfigured to use the new master, and applications using the Redis server are informed of the new address to use when connecting.

**Clustering**

The Redis cluster automatically distributes data sets to multiple nodes. This improves the performance and scalability of database deployments while maintaining business continuity if subsets of nodes cannot communicate with the rest of the cluster.

## Cassandra



### Presentation

"Apache Cassandra is a powerful and highly extensible NoSQL database. It is designed to process BigData loads in real-time across multiple data centers without any failures. » (Bradberry, 2014)

The name Cassandra was inspired by a queen of the legendary Troy who thought she was a repository of metaphysical events. It was originally developed by Facebook to speed up site searches. Before being adopted and distributed as open source by Google in 2008.

Cassandra is a column-oriented database, its highly available and highly distributed nature leaves no single point of failure, meaning that each node in its cluster can respond to any query. It supports replication across multiple data centers.

Web giants such as Netflix, eBay, Twitter, Reddit, and Ooyala use Cassandra. "To date, the largest public Cassandra cluster has more than 300 Terabytes of data distributed across 400 machines.

### The data model

In Cassandra, the Keyspace is the data container of the application (a bit like a Database or a schema for a relational database). In these keyspaces are one or more families of columns (which correspond to tables in a relational database).

These column families contain columns and a set of related columns that are identified by a row key. Besides, each row in a column family does not necessarily have the same columns as another row.

For all column families, each row is uniquely identified by its key (similar to the concept of a primary key for relational databases). A column family is always partitioned by its row keys, which are implicitly indexed.

In fact, in Cassandra, a column is the smallest increment of data. It is modeled by a tuple that contains the name, the value, and a timestamp. This timestamp is used by Cassandra to determine the most recent update in the column (and, in this case, it is the most recent that takes precedence in a query).

Queries in Cassandra are static, so we need to know their patterns before creating the data model; this data model does not have the same goals as the relational model, namely efficient storage and relationships between entities, but its goal in Cassandra is to optimize the performance of rows and the storage of large volumes of data.

### Characteristics

* Without schema: Like all other NoSQLs, with Cassandra, you can directly start inserting data without the puzzle of the structure name. Afterward, each column is independent and can have rows that differ from the others; this is total flexibility. If you have a clear idea about the structure and the types of data to be implemented, Cassandra allows you to structure your data like the classic relational method.
* CQL: "Cassandra Query Language", which is close to the traditional SQL but without the GROUP BY, JOIN functions and with limited use of ORDER BY; they are replaced by others more improved especially for Cassandra.
* Cluster: Two or more Cassandra instances (nodes) working together; they communicate using a protocol called Gossip.
* Homogenous Environment: All Cassandra nodes are similar and each contains all the configuration needed to complete the cluster; unlike other NoSQL databases such as HBase which has heterogeneous nodes.
* Quorum: This is the number of nodes that respond to a query; it is based on RF; a read/write quorum means that the query queries RF/2+1 nodes; for RF = 3, a query queries 2 nodes. This ensures data consistency.
* Replication factor (RF): It is the number of copies of each line of the database that we decide to save in each cluster; RF of 3 means that 3 copies of the database reside in the cluster. It is also used to determine the number of quorums, which ensures availability and fault tolerance. In Cassandra, there is no master copy and slave copy, all copies have the same value with respect to the cluster. A rule to be respected when creating the copies is that the number of copies must not exceed the number of nodes available in the cluster, otherwise, one risks not being able to write to the database, only reads remain possible for reasons of availability.
* Adjustable coherence: Cassandra allows the coherence level to be adjusted according to the read/write needs. The lowest is the "ANY" level while "ALL" is the highest coherence. These levels are:
  + ANY: the weakest, write or update must succeed in any available node. A write must be written to at least one node. If all replica nodes of the given line key are failed, the write can still succeed once a transfer has been written.
  + ONE: write\update must succeed in at least one node responsible for this line (primary or replica).
  + QUORUM: write\update affects the majority of replicas, i.e. must succeed in a minimum number of replica nodes determined by RF/2+1. Other settings are allowed.
* Partitioning: This is the mechanism that determines the strategy of where the replications are located; it is a simple hash function. It calculates the token number to assign for each line it uses to find that line later. Different algorithms are used: alphabetical sorting, random sorting ...etc.
* Replication Strategies :
  + Simple strategy: used with a single machine, therefore it is the default location for storage space. The Partitioner is responsible for the distribution of replications on the nodes. The first replication is arranged randomly, then the others are added in a clockwise direction. All nodes are on the local machine or datacenter, and once started the request does not have to look elsewhere.
  + Network topology strategy: used when deploying Cassandra on multiple machines or datacenters in a cluster. It determines the number of replications to be created on each datacenter.
* The number of replications: When deciding how many replications to create in each node, there are two criteria to consider:
  + Latency: If a reading has to be made on another Datacenter than the one requested.
  + Acceptable failures: Scenarios to be expected if a node fails

We may have a case where the number of replications differs from one Datacenter to another; when the number of readings on a Datacenter is very high compared to ours, it is advisable to increase the number of replications on the first one to reduce the latency.

* Snitches: A snitch is a protocol that takes care of determining which node should be requested during a reading by creating a topology that brings the nodes together.
* Deployment: Cassandra can run on any system that supports the Java JVM, but to take full advantage of its performance, it is strongly recommended to use a Cloud service (Amazon Web Service, Google Compute Engine, Windows Azure...).

## MongoDB



### Presentation

The name MongoDB is derived from the English adjective "humongous" Data Base, which means a huge or gigantic database. It is one of the NoSQL databases, where there are not the concepts of table, schema, SQL query, join, foreign key and ACID properties. MongoDB is a document-oriented database, its slogan is "one size does not fit all" (Hows, 2013), it uses documents instead of the classical rows, and it is thanks to this change that we will see that it is faster, considerably more scalable horizontally and easier to use. To achieve this result, MongoDB has sacrificed some properties, therefore it is not ideal for all situations.

The notion of consistency is relative, that is why MongoDB is not a solution for bank management applications.

One of the main concepts of MongoDB is that it is based on more than one copy on different servers, it is one crash, there will always be a restored copy. It is suitable for all operating systems, Windows, Linux, Solaris and Mac OS.

The user does not have to worry about how to organize his data, just put them together in a document and store them in MongoDB.

MongoDB uses the BSON format to store data, BSON stands for the abbreviation of "Binary JSON"; JSON "JavaScript Object Notation" allows the representation of complex and structured data in a simple and human-readable way, it is both a language of data exchange and data storage. Because it stores all related data in a single document and all similar documents in one place, BSON makes the processing of queries faster.

### Le sharding

Sharding, fragmentation, is a technique that allows the horizontal extension of a system on multiple servers (distributed storage). It allows us to overcome hardware limitations by propagating data across multiple physical partitions (shards).

The documents of MongoDB can be distributed on several servers, each server verifies if it contains the desired information and returns the result. Thanks to the sharding concept, MongoDB processes a request on two servers with the same ease and efficiency on a hundred servers. It supports the operations of splitting and recombining data while maintaining transparency on the developers' side. This concept gives wings to the scalability possibilities of MongoDB.

### Characteristics

* MongoDB is developed with C++, which is why it is executable everywhere on all platforms, whether 32 or 64 bits and with minimal requirements.
* It is required that each document is uniquely identified if the user does not create the identifier, MongoDB generates it automatically. All documents are indexed on the \_id field; this field cannot be deleted.
* The content of a document is a set of key-value pairs; contrary to RDBMS, when the value is unknown, the key is omitted.
* Collections: MongoDB allows the creation of collections of similar items or mixed items as needed; for indexing reasons, it is preferable to use collections of identical items.
* MongoDB is a collection of collections of heterogeneous items, which gives it the flexibility and ease to create databases.
* Dynamic queries: odd as a criterion, because other NoSQL DBs do not support dynamic queries such as CouchDB.
* Geospatial indexing: recently introduced, it is the possibility to index items according to their distance of coordinates from a given geographic point.
* Query optimizer: MongoDB has an internal tool that can help to optimize customer queries by optimizing the response time.
* The data update is done directly in memory, which means that MongoDB does not need the extra memory or writing to disk; this gives it more execution speed but less data reliability.
* Replica sets: Set of identical copies of databases where only one is master and the others are slaves, the writing is done only on the master copy and then an update is done on the slave copies. If a failure occurs on the master copy, one of the slave copies is elected master; this guarantees the durability of the base.

## YCSB

YCSB (Yahoo! Cloud Serving Benchmark)is an open-source testbed for evaluating the performance of NoSQL databases and Data Warehouse Clouds. Initially Yahoo! developed it in 2010 to compare their own NoSQL PENUTS DBMS with other NoSQL DBMS (Shashank, 2011).

It deals with two aspects: performance and scalability. Database performance is a set of measurable criteria that can differentiate one database from another. One of these criteria is the latency of operations or the response time of user queries; the second criterion is the execution time of a workload (set of Operations), and the third criterion is the number of operations per second (operation rate). Performance is the aspect that is of most concern to users when browsing the web. In this section, the YCSB, through its workload generator, generates a dataset and loads it into the databases to be tested, then it executes the performance measurement operations. The results obtained are essentially the number and rate of successfully executed CRUD operations (read, write, update or insert) and the average latency of each operation. These results can be represented in a graph for better understanding, analysis and comparison.

Scalability testing is done by running the same workload on a DBMS and then comparing the same results by adding servers to the system. If we deploy new instances (servers) and the system down, we talk about scalability (extensibility); if we deploy instances and the system up, we talk about elasticity.

We will use it to test different workload scenarios on the same dataset in the two NoSQL DBs, Cassandra, and MongoDB.

### Workloads

The workloads are variations of the operations of reading, writing, inserting and updating datasets. The data created is random; while the size of a record, the number of records and the number of operations are adjustable.

The operations possible by YCSB are:

* Insert: Insert a new record
* Update: update a record
* Read: Play a recording
* Scan: Scan a subset of records, the start record and the number of records to be scanned are randomly selected.

Besides, the choice of data to be read or updated is related to the target record selection method; these methods are:

* Uniform: All records have the same weight when selected.
* Distribution of Zipfian: Sorting records by popularity (appearance)
* The last: Sort the records by their date of insertion, the last inserted are the first in the distribution.

The benchmark provides interface layers to almost all known DBs, Cassandra, MongoDB, HBase, Accumulo, and Voldemort. Nevertheless, the user can extend it to other databases.

## Conclusion

In this chapter, we have described the architectures, technical features and components of three of the famous systems NoSQL, Cassandra, Redis, and MongoDB. In order to give an illustrative idea of the performance of these NoSQL DBs, we presented the Yahoo! YCSB, the testbed we will use in the next chapter.

# Implementation and experimentation

## Introduction

This chapter is devoted to the experimental part of the study. We will define the YCSB test bench, its implementation, and use; we will mention the test environment, the different configurations and implementations of the tools used, and we will analyze the results from the practice of the different experiments.

## The experimentation

The YCSB tests were done on a machine with the following characteristics

•Processor : Intel® Core™ i7-7700HQ CPU @ 2.80 GHz (8CPU), 2.80GHz

•RAM: 16 GB

•System: Microsoft Windows 10 Pro 64 bit

•SSD: 256 GB

To perform our tests, we will use the basic workloads provided with YCSB :

* Workload A: 50% Read, 50% Update
* Workload B: 95% Read, 5% Update
* Workload C: 100% Read
* Workload D:5% Insert, 95% Read
* Workload E: 95% Scan, 5% Insert
* Workload F: 50% Read, 50% Read-Modify-Write

To evaluate the loading time, we generated 500,000 records, each with 10 randomly generated 100-byte fields on the registry ID key, that is about 1kb per record. Each record is identified by a key composed by the string "user" followed by several digits, for example, "user152348765014", which is the record key. Each field in the record is identified as field0, field1, ..., field i respectively. Workload execution consisted of running 500,000 operations, which means that there were 500,000 database queries each time during the test.

The comparative study was carried out on the following NoSQL databases:

* MongoDB: version 4.2
* Redis: version 5.0
* Cassandra: version 3.11.5

These databases are hosted in Linux x64 docker containers.

The YCSB version is 0.17.0.

## The commands

### Start databases

**Synthax**

Docker run -p [HOST\_MACHINE\_PORT]:[CONTAINER\_PORT] –name [CONTAINER\_NAME] -d [IMAGE\_NAME]

**MongoDB**

docker run -p 27017:27017 --name mongo -d mongo

**Redis**

docker run -p 6379:6379 --name redis -d redis

**Cassandra**

docker run -p 9042:9042 --name cassandra -d cassandra

### Additional set up for Cassandra

It looks like YCSB is not able to create a keyspace and a table automatically. We fix this problem by creating them manually.

**Open the bash shell of the Cassandra container**

docker exec -it cassandra bash

**Open the cqlsh shell**

cqlsh

**Create the keyspace ycsb**

create keyspace ycsb WITH REPLICATION = {'class' : 'SimpleStrategy', 'replication\_factor' : 3};

**Select the keyspace ycsb**

use ycsb;

**Create the table user table**

Create table usertable(y\_id varchar primary key, field0 varchar, field1 varchar, field2 varchar, field3 varchar, field4 varchar, field5 varchar, field6 varchar, field7 varchar, field8 varchar, field9 varchar);

### Load a set of data

**Synthax**

powershell ".\ycsb load [DATABASE\_TYPE] -s -P [WORKLOAD\_PATH] -p recordcount=[AMOUT\_OF\_RECORDS] -threads [NUMBER\_OF\_THREADS] -p [SERVER\_INFO]"

**MongoDB**

powershell ".\ycsb load mongodb -s -P ../workloads/workload[WORKLOAD\_LETTER] -p recordcount=500000 -threads 16 -p mongodb.url='mongodb://127.0.0.1/admin'"

**Redis**

powershell ".\ycsb load redis -s -P ../workloads/workload[WORKLOAD\_LETTER] -p recordcount=500000 -threads 16 -p redis.host=127.0.0.1"

**Cassandra**

powershell ".\ycsb load cassandra-cql -s -P ../workloads/workload[WORKLOAD\_LETTER] -p recordcount=500000 -threads 16 -p hosts='127.0.0.1'"

### Run operations

**Synthax**

powershell ".\ycsb run [DATABASE\_TYPE] -s -P [WORKLOAD \_PATH] -p [SERVER\_INFO]" -p operationcount=[AMOUT\_OF\_OPERATIONS] -threads [NUMBER\_OF\_THREADS] | tee [OUTPUT\_FILE]"

**MongoDB**

powershell ".\ycsb run mongodb -s -P ../workloads/workload[WORKLOAD\_LETTER] -p mongodb.url='mongodb://127.0.0.1/admin' -p recordcount=500000 -p operationcount=500000 -threads 16 | tee mongoDB\_[WORKLOAD\_LETTER].txt"

**Redis**

powershell ".\ycsb run redis -s -P ../workloads/workload[WORKLOAD\_LETTER] -p redis.host=127.0.0.1 -p recordcount=500000 -p operationcount=500000 -threads 16 | tee redis\_[WORKLOAD\_LETTER].txt"

**Cassandra**

powershell ".\ycsb run cassandra-cql -s -P ../workloads/workload[WORKLOAD\_LETTER] -p hosts='127.0.0.1' -p recordcount=500000 -p operationcount=500000 -threads 16 | tee cassandra\_[WORKLOAD\_LETTER].txt"

## RESULTS

### Workload A (50% Read, 50% Update)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

### Workload B (95% Read, 5% Update)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

### Workload C (100% Read)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

### Wokload D (5% Insert, 95% Read)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

### Workload E (95% Scan, 5% Insert)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

### Workload F (50% Read, 50% Read-Modify-Write)

**RUNTIME**

**THROUGHPUT**

**AVERAGE LATENCY**

Overall evaluation of MongoDB, Cassandra, and Redis

These tests were performed with the 16 threads option only. It may be interesting to modify this value to observe other results in a future search.

The experimental results of the different tests carried out allowed us to evaluate and compare the three types of NoSQL databases: document-oriented, column-oriented and value-key oriented, based on the execution time of the different workloads.

Among the NoSQL solutions studied, we can affirm that the key/value database studied, Redis obtains performances far superior to the other NoSQL databases. Its superiority can be explained by its characteristics mentioned in the previous chapter. However, it remains inferior when running workload E (Scan operations).

# General Conclusion

Our research consisted of making a comparative study between the different families of NoSQL solutions: document-oriented, column-oriented and value key, namely MongoDB, Cassandra and Redis.

The objective of this study is to evaluate the performance of these databases by inserting 500,000 records, then launching a set of tests in the form of a workload composed of 500,000 operations each of different natures:

reading, scanning or updating.

The tool used to arbitrate the six systems is Yahoo! Cloud Serving Benchmark, which is highly recommended for such studies in the field of NoSQL databases.

After reading and analyzing the experimental results, it can be said that the key/value databases are superior in terms of performance. However, this does not mean that they should be the default choice on all types of projects, as the specificities of database types must also be taken into account. For example, despite the advantage of Redis in terms of performance, it may be preferable to use MongoDB for a WEB project because it is very scalable and does not respect ACID transactions.

In conclusion, we can retain that the choice of using a DBMS depends on a set of parameters related to the environment in which the data are exploited. Indeed, the type of data and the type of processing carried out on this data are important clues in defining the solution to adopt. The estimated frequency of reading, writing and updating as well as the size of the data are the essential determining factors in the choice of an alternative among others. Currently, the trend towards a specific NoSQL solution is far from being indisputable due to a large number of existing systems.

This work can form the basis for further in-depth work in the future, to try to test NSQL databases in a real distributed environment, preferably the Cloud, and test cases of adding servers with the system on and off, and examine the case of failure of a server with a system in operation.

# References

Armstrong, T. P. (2013). LinkBench: a database benchmark based on the Facebook social graph. *ACM SIGMOD International Conference on Management of Data 2013* (pp. 1185-1196). New-York: ACM.

Berman, J. J. (2013). *Principles of Big Data.* Elsevier.

Brewer, E. (2000). Towards robust distributed systems. *Symposium on Principles of Distributed Computing*, p. 45.

Clémencon, S. (2017). *Voyage au coeur du Big Data.* CEA.

Connolly, S. (2012). *7 key drivers for the big data market.*

Cood, T. (377-387). A Relational Model of Data for Large Shared Data Banks. In T. Cood, *Communications of ACM* (p. 1970).

David Hows, E. P. (2013). *The definitive guide to MongoDB.* Apress.

DB-Engines. (2020). *DB-Engines Ranking*. Retrieved from DB-Engines: https://db-engines.com/en/ranking

DeCandia, G. e. (2007). *Dynamo: Amazon’s Highly Available Key-value Store.* Amazon.

Deepak, G. (2016). A Critical Comparison of NOSQL Databases in the Context of Acid and Base. *Culminating Projects in Information Assurance*.

Deepika, V. S. (2016). Comparative Study of SQL and NoSQL Databases to evaluate their suitability for Big Data Application. *International Journal of Computer Science and Information Technology Research*, 2348-1196.

Furrer, H. (2015). *SQL, NoSQL, NewSQL, strategie de choix.* Haute Ecole de Gestion de Geneve.

Garcia-Molina, H. U. (2009). *Database Systems.* Pearson Education.

Harrison, G. (2015). *Next Generation Databases.* Apress.

Heinrich, L. (2012). *Architecture NoSQL et réponse au Théorème CAP.* Haute Ecole de Gestion de Geneve.

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? *IEEE Computer Society*, 12-14.

Lokesh Kumar, S. R. (2015). Comparative analysis of NoSQL (MongoDB) with MySQL Database. *International Joutnal of Modern Trends in engineering and Research*, 2393-8161.

Omar Hajoui, R. D. (2018). An advanced comparative study of the most promising NoSQL and NewSQL databases with a multi-criteria analysis method. *International Journal of Computer Science and Information Security*, 1947-5500.

Russell Bradberry, E. L. (2013). *Practical Cassandra: A developer's Approach.* Apress.

Tiwari, S. (2011). *Professional NoSQL.* Wrox.

Vaish, G. (2013). *Getting Started with NoSQL.* Packt.