Not Only SQL as a Alternative to Relational Database Systems

Nuno Cancelo
ISCTE-IUL, Lisbon, Portugal
Avª das Forças Armadas, 1649-026 Lisboa, Portugal
nasco@iscte-iul.pt nuno.cancelo@gmail.com

ABSTRACT
Every device has the ability of acquiring data from the surrounding environment and sooner or later send it through the Internet so it could be stored in the Cloud, in their clusters of machines with large amount of storage to maintain these information. NoSQL appear as response to a need of the internet usage, with huge sets of unstructured data need to be stored but avoiding the need of storing into relational databases. Architecture designed to provide flexibility of usage, great performance, scalability (automatic) and replication (automatic) capabilities, since the relational systems could not provide natively this features. The goal of the paper is to provide the knowledge necessary to identify the differences between the systems and provide guidance through the available NoSQL stores to choose the best choice to the application needs.

Author Keywords
NoSQL; SQL; Database; Performance; Scalability; Security; Web Environment ; Cloud

ACM Classification Keywords
H.2.4. Systems: Concurrency; H.2.4. Systems: Object-oriented databases; H.2.4. Systems: Query processing; H.2.4. Systems: Relational databases; H.2.4. Systems: Textual databases; H.2.8 Database Applications: Data mining; H.2.8 Database Applications: Spatial databases and GIS; H.2.8 Database Applications: Data mining

INTRODUCTION
Since IBM computer scientists research and brought to the world the concept of relational database, the computer world never was the same. The ability of structuring the data, allowing reliable storage and organized was only the first steps in a path that lead us to current Internet of Things[4].

Every device has the ability of acquiring data from the surrounding environment and sooner or later send it through the Internet so it could be stored in the Cloud, in their clusters of machines with large amount of storage to maintain these information.

So what is the problem? The data is not saved and available for querying? The short answer is Yes, the data is available and could be made queries into the destination data repository, but the real question should be made is When should i get the data that I’m querying now? or even What is the reasonable waiting time for getting the results? There is performance issue regarding big sets of data and the relational databases, and when dealing with cloud environment applications and thousands (or even million) end-users interaction, the expectation demand the service to be fast, very fast.

The relational database system architecture is designed to provide consistency and integrity of the data, to achieve this goal many features were implemented to guarantee this principle, but with great strength there come great costs and two of them was performance and scalability.

According to Stonebraker[20] "There are two possible reasons to move to either of these alternate DBMS technologies: performance and flexibility.". NoSQL is in fact an alternative to relational database systems.

NoSQL appear as response to a need of the internet usage, with huge sets of unstructured data need to be stored but avoiding the need of storing into relational databases. Architecture designed to provide flexibility of usage, great performance, scalability (automatic) and replication (automatic) capabilities.

In section ”STATE-OF-THE-ART” is an overview of the work made by other authors about NoSQL, in section ”RELATIONAL DATABASE” is exposed the key features of a convectional database, in section ”NOSQL” is exposed the database and their flavours and characteristics, their advantages and disadvantages, a comparative between the databases and a guidelines to choose the appropriate database according to the usage needs.

STATE-OF-THE-ART
There have been some work in the community regarding NoSQL, since 2009 when the NoSQL movement has been growing becoming the developers top choice to support web applications, because of the special needs such scalability and performance.
There are four categories of work regarding NoSQL, from the literature that was gathered and studied.

**Base Study and Comparative**

Some authors such as Hecht and Jablonski[3] and Mapanga and Kadebu[8] compare features of the NoSQL point of view, Sharma and Dave[16] and Nayak, Poriya, Poojary [10] focus in a simple comparative between relational databases and NoSQL databases.

**Scalability and Distributed**

The core of the NoSQL architecture turns on this ability, but DeLoatch and Blindt[1] talk about the NoSQL databases turn a scalable Cloud and the potential Enterprise solutions, while Cattel[?] talks about scalable relational databases and scalable NoSQL databases and Pokorny[14] about web environment scalability regarding NoSQL.

**Security**

Only one of the articles study security, and was the work of Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes and Jenny Abramov[13] brought some light into the security in Cassandra and MongoDB NoSQL databases.

**Great NoSQL Applications**

There are many fields were the NoSQL database show their worth and the Big Data field of study is become one of great interests. Ranjbar[15], Moniruzzaman and Hossain [9] covers the relation between Big Data and NoSQL.

Another case study of usage of NoSQL database is made by Loscio, Oliveira and Pontes [7] when talk about collaborative web applications and FOTACHE and COGEAN[2] focus their usage in mobile applications, comparing MongoDB and PostgreSQL.

**RELATIONAL DATABASE**

The relational database evolved over the years since it appear in 1970 and was largely adopted by companies across the world. With one proper programmatic language named SQL (Structured Query Language), interact with one relational database became easy, dealing with tables as shown in figure 1.

The language as many features that later became an international standard with SQL-86[17] and evolved to SQL:2011[18], been the current standard. The language provide interaction with structure data with their CRUD(Create, Read, Update and Delete) statements and allows combination with multiple statement for more complex actions.

Behind the scenes, the relational database management system (rdbms) is working to guarantee the ACID (Atomicity, Consistency, Isolation, Durability) properties of the transactions that are working with the existing database data. If we consider the others features of the language, such as constraints (primary and foreign keys, data type, etc) or stored procedures/functions, we make aware that every time a transaction is executed every record in the database must respect the consistency of the data in it. As it may seem, as the database grows (more records) more slow became the execution of the transaction and the companies with great amount of hourly data need to have fast systems.

As Ameya, Anil and Dikshay[10] stated, the relational model has some scalability problems due performance and Stonebraker[19] refers also a flexibility problem, because of the structured data of the database. Someone could upgrade the machines where the database are hosted, by adding more fast memory and fast storage but we are adding more hardware, we are not solving the performance and flexibility issues just delaying.

Leavitt[6] mention some limitations of the relational database model such as:

- **Scaling:** At some point there is a need to scale the database and distribute across multiple servers.
- **Complexity:** All data must be converted to table structures.
- **SQL:** Add more complexity to the developer code and does not work well with agile development.
- **Large Feature Set:** Oriented to data integrity and not to data itself.

Stonebraker[19] also refers other functionalities that are extremely useful in a traditional RDBMS but deteriorates the performance of the system such as logging, locking, latching and buffer management and refer the by eliminating these functionality the RDBMS could perform 25% faster, which is a great improvement over billions of records. In figure 2 shows that be writing a record the distributed systems become locked while the transaction is action, while in figure 3 the write is full parallel not blocking the communication between nodes.

With the Web 2.0 evolution, there is information everywhere to be captured and stored, but these data is not structured,
how could be images, emails, web pages, videos, musics, etc stored in a database system and still be an performance system? How could the relational database system be adapted to full-fill the needs of the companies that deal with billions of record transactions? Well, you don’t.

Relational databases were not developed to answer this paradigm, born to bring structure in a age where the information was kept in boxes of paper and needed a way to querying it once it was the data introduced in the system. Many things happened through the last three decades, the personal computer exponential grow, the internet usage grow, the mobile devices grow, the Internet of Things[5].

Aware with this menace, big companies started to develop database systems that fit their needs. Leavitt[6] mention that one of the key moments of the NoSQL movement was in 2007 when Amazon introduced Dynammo distributed NoSQL system for internal use.

There was a time to change, a time to evolve.

**NOSQL**

At the beginning the term NoSQL refer to a system that had no structure and no standard querying language, but through the years the community of developers add another and more mean-full translation: *Not Only SQL*[11].

The NoSQL movement began in 2009 and since then it is growing, mostly because the high scalability provided, their architecture was designed to provide great performance, availability and scalability, features that were gain since it was removed features that were unnecessary[10]. The NoSQL systems provide BASE (Basically Available, Soft state, Eventual consistency) system, but not ACID as a Relational Database Management System. By leaving the constraints of the ACID system the system became more performance. This is one of the big differences of the system.

There are over one hundred NoSQL[11] databases current available for almost all flavours. The databases fall in four categories:

- Key-Value Stores
- Wide-Column Stores
- Document Databases
- Graph Stores

Through this four main categories are the database stores that could fill the special need that an relational database could not fill.

**Key-value stores**

This Store is the simplest to implement and use. It is a schema-less store with a key-value structure, where the KEY identifies the data stored and the VALUE is the data or object it self, it is a well known structure between the developers because it is used among in-numerous programming languages. A example could be seen in figure4.

**Wide-column stores**

It is a hybrid column/row store, similar to relational database but not alike. In this stores each KEY is associated with one or more columns. A Column stores their data in such way that can be efficiently aggregated, reducing the I/O activity.

Also provide high scalability, since the data is stored by column order and it is suitable for data mining and analytic systems. A example could be seen in figure5.
Document databases
The data is stored in the form of documents. This kind of store offer great performance and horizontal scalability. The internal storage is similar to relational databases but more flexible because of their lack of schema, and the store document is in a standard format (xml, pdf, json, etc). Because of his flexibility the documents may have only the filled and important fields, letting the empty and null out, saving some storage space.

Document databases are similar to key-value stores but a bit more complex, since it allows that value could be another key-value store. A example could be seen in figure 6.

Graph stores
The Graph Store saves the data in a form of graph, where the nodes are object and the edges relations between the objects. The main goal of this kind of store is to know the relations between nodes and how the nodes are inter-connected. A example could be seen in figure 7.

Scalability
According the MongoDB[12] site, the relational database scale vertically meaning that they have one server hosting the database and limited to the server capabilities to grow and adds some failure point to the infrastructure availability. To bypass this constraint, the system has to grow horizontally, by adding more hosts, becoming a distributed system solution.

Relation database could shard the database through the server, but it has some complex arrangements to make the multiples servers act as one alone. Because this distribution is not offer natively, much of the work is made by the development teams to provide a stable and coherent database system. The computer application as the responsibility of distributing the information across the databases, handling the errors and exceptions, data balancing and replication. All this, because it is not supported by nature.

NoSQL databases usually support auto-sharding, meaning they automatically spread data across the servers, without requiring any application development. Data and query load are balanced across servers, having fault tolerance if one server goes down. Most NoSQL databases also support data automatic replication, providing in this manner high availability and disaster recovery.

Security
Okman, Gal-Oz, Gonen, Gudes, Abramov[13] provided some enlightenment relating security in *NoSQL* databases. They stated that "The common features of NoSQL databases can be summarized as: high scalability and reliability, very simple data model, very simple (primitive) query language, lack of mechanism for handling and managing data consistency and integrity constraints maintenance (e.g., foreign keys), and almost no support for security at the database level".

Although every *NoSQL* database is a special case with an answer to special need, the same paper[13] studies two type of *NoSQL* database *Cassandra* and *MongoDB*, being the first a Wide-column store and the last a Document Store.

The focus will be *MongoDB*, since it is the well known document store and choice of many web developers. The developers choose *MongoDB* because it schema-less is very similar to JSON (Javascript Object Notation) format and it uses a RESTful API.

This document store has some security flaws, some of the because security was not a concern. This was an option took by the designers but the developers should be aware. This list shows some of them:

- *MongoDB* data files are unencrypted.
- *MongoDB* don’t provide methods to automatically encrypt the data files
- Potential Javascript injection attacks
- Does not support authentication in shared mode
- Does not support authorization, because it does not support authentication
- Does not support secure protocols such https

Some of this flaws make someone to think twice while choosing it to a particular usage in production mode, because we are talking about security of data, we are talking about privacy.

**Advantages**

One of the key advantages of NoSQL databases in comparison to relational database management systems is scalability, as it was mention previously, relational databases are designed to run over a single host server and although it is possible to run in a distribute manner, it has much complexity added to the designed. However, NoSQL databases are designed to be scalable and providing auto-sharding and replication capabilities. Another key advantage is the best performance due the fact that war removed all ACID capabilities and features that turn the system slower and were unnecessary.

The design of *NoSQL* database focus the usage of large volumes of data, being them structured, semi-structured or unstructured. These kind database provide great support to technologies that use *Agile* methodologies and frequent code interaction and changes, becoming easy the usage in object oriented programming languages due their syntax.

This kind of databases do not required the constant work of one database administrator, since it has no structure and the work is made by the development teams and it has automatically replication and scalability.

**Disadvantages**

Not all are roses. The *NoSQL* has some disadvantages worth reporting. One of them, reported by database administrators, is the lack of compliance of ACID, when try to keep data consistency and reliability. Other disadvantage, this time by developers, is that there is no standard query language and there is no standard programmatic interface, every database has it own query language and programmatic interface.

Software developers has some more complaints, because there is no standard language, all the programming, validation and query work is done by the developer, giving him more overhead and complex work.

Due the fact of *NoSQL* deals with no structured data, maintenance because difficult because with the lack of development documentation, is hard to manage the database.

This is a result of the level of immaturities of the *NoSQL* and with the fact there is not yet international standard. And companies do not adopt this technology because they are unfamiliar with them and may not have the needed human resources to work with them, and the lack of costumer support and management tools in most database in the market.

**NoSQL vs. SQL Summary**

The authors of the site MongoDB[12] provided a summary of the comparative between *NoSSQL* and *SQL*, which is transcript:

**SQL Databases**

- **Types:** One type (SQL database) with minor variations
- **Development History:** Developed in 1970s to deal with first wave of data storage applications
- **Examples:** SQLServer, MySQL

**NoSSQL Databases**

- **Data Storage Model:** Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees"
in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary.

- Schemas: Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.

- Scaling: Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required.

- Development Model: Mix of open-source and closed source

- Supports Transactions: Yes, updates can be configured to complete entirely or not at all

- Data Manipulation: Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE...

- Consistency: Can be configured for strong consistency

NoSQL Databases

- Types: Many different types including key-value stores, document databases, wide-column stores, and graph databases

- Development History: Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage

- Examples: MongoDB, Cassandra, HBase, Neo4j

- Data Storage Model: Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically.

- Schemas: Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.

- Scaling: Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary

- Development Model: Open-source

- Supports Transactions: In certain circumstances and at certain levels (e.g., document level vs. database level)

- Data Manipulation: Through object-oriented APIs

- Consistency: Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra)

There are some valid points between each other and depending on the end usage someone may choose one or the other.

Choosing the right database

There are too many reasons to a relational database be the first choice: database support, standard query language, core application structured storage, security, integrity, consistency, etc, etc, etc.

Do you need a better performance? Do you need a (cheap) scalable system? Need to deal with huge unstructured data?

If the answer is yes to either one, well you can’t rely on relational database. Well not only.

The question now is what is the purpose of database? what will they store do? Now is the time of choosing the right NoSQL according to your needs, as shown in figure 8.

Need Key-Value Store? If the usage is simple, such as mapping keys and values maybe this is for you. Amazon’s Dynamo was one of the first NoSQL to appear and it is used by Amazon, in their internal processes.

Need Wide-Column Store? Well, you have a complex usage and like to have some kind of aggregation of the contents, this is the choice. Google used their implementation of this store called Google’s Bigtable.

Need Document Store? If made this far and your usage do not take in consideration relations between records and would like a very versatile database, a document store is the right bet. The most used of this kind of database is MongoDB, with great support by the community, and educational initiatives by the company such as their MongoDB University, where developers learn for free how to work the MongoDB database.

Need Graph Store? If you care about relations between data, yes you need a graph store. One of the most familiar is Neo4J.

Their is a pretty long list of the NoSQL database open to public usage and could be found at NoSQL site[11].

This mean that you only able to use NoSQL or Relational Database? No, the full potential is achieved when you combined both. Imagine you are logging your application activity, you could use one NoSQL to capture the huge data and the have a ETL process that loads that data to the relational database to be able to provide some business intelligence support.

CONCLUSION
There are differences between a standard relational database and a NoSQL architecture, they serve different purposes of usage. One is focus on data integrity, while the other is focus on performance.

While the data integrity is important in operational systems and it is also true they have performance and scalability problems with the grow of the data. It’s important to know the key aspects of the system that provides lack of performance through the time and why is difficult to provide some scalability to the system.

NoSQL databases come with one fresh architecture focus on performance, scalability, data replication, with programming interface and four use cases according to the needs. But as shown through the document, it has some security issues and some disadvantages that is the consequence of his young age and lack of standards. It is taking his first steps, the same way SQL took roughly seven years to achieved his international standard.

Both systems could co-exist and providing a exponential potential of the organization value when combined. The versatile cheap solution of distributive database system united with a relational database to provide support of business intelligence.

REFERENCES
2. FOT ACHE, M., and COGEAN, D. Nosql and sql databases for mobile applications. case study: Mongodb versus postgresql.


15. Ranjbar, M. Nosql: a distributed solution of big data.


