The Open University of Israel

Department of Mathematics and Computer Science

Comparative Survey of NoSQL/ NewSQL DB Systems

Final Paper

Submitted as partial fulfillment of the requirements towards an

M.Sc. degree in Computer Science

The Open University of Israel

Computer Science Division

By

Yuri Gurevich

Prepared under the supervision of Prof. Ehud Gudes

December 2015

Acknowledgements

Firstly, I would like to sincerely thank my supervisor, Prof. Ehud Gudes, for his skillful assistance, attention to details and helpful remarks throughout the whole process of research and writing. I much appreciate Prof. Gudes's time investment, guidance and continuous support.

I am also grateful to Dr. Nurit Gal-Oz for providing "second reader" feedback on this final paper. Dr. Gal-Oz's valuable comments and suggestions greatly improved the paper.

I must also acknowledge the authors of BG benchmark, Dr. Sumita Barahmand and Prof. Shahram Ghandeharizadeh (The University of Southern California), for their kind assistance with BG interfaces during the implementation of the project described in Section 7 of the paper.

Finally, I wish to thank my wife Xenia and my son Daniel for their unconditional support, encouragement and patience during my work on this paper. Without such a team behind me, I doubt that I would be in this place today.

Contents

List of Figures

List of Tables

Abstract

While traditional relational databases are still used in a large scope of applications, we have seen recently an explosion in the number of a new data bases technologies developed in particular for Big Data serving. Currently the main alternatives to RDMBS are NoSQL and NewSQL databases. The primary focus of this paper is to survey, compare and evaluate a number of NoSQL and NewSQL databases.

The NoSQL databases were created as a mean to offer high availability at the price of losing the ACID (Atomic, Consistent, Isolated, Durable) guarantees of the traditional databases in exchange with keeping a weaker BASE (Basic Availability, Soft state, Eventual consistency) feature. We're going to survey four main categories of NoSQL databases:

- Key-value stores
- Document stores
- Column family stores
- Graph databases

Additionally we will evaluate NewSQL which is another class of modern database management systems that seek to provide the same scalable performance of NoSQL systems for OLTP (online transaction processing) workloads while still maintaining the ACID trait of a traditional database system.

This paper provides a comparative survey of NoSQL and NewSQL data stores focusing on the technical characteristics as well as the popularity and applications suitability.

Additionally we also refer in this paper to the practical evaluation of the performance of NoSQL and NewSQL data stores that we performed as a part of the Final Project. In

the project we utilized BG benchmarking system that rates data store by processing interactive social networking actions. To do so we implemented separate clients for Cassandra and NuoDB and then conducted various experiments using BG in order to obtain practical results for these two data store types.

1. Introduction

Over the past decade, we have witnessed a rapid growth in a number of different types of data bases. It's related to the development of the Internet, mobile devices and cloud computing. All these environments impose new requirements for an effective storage and processing of data.

In order to deal with these challenges, some companies maintain data centers and farms which contains clusters with thousands of commodity hardware machines.

Old fashioned relational databases do not provide a good solution in this situation, due to their normalized data model and full ACID support.

Additionally it turns out that the common approach "one size fits all" is no longer valid for current application scenarios. On the contrary, in the new reality it is more appropriate to design systems based on the nature of the applications and its data/ query demands.

Consequently, multiple alternative database solutions were specifically designed to satisfy diverse needs. Many of these new databases belong to NoSQL and NewSQL classes of DBMS. In this paper we survey, compare and evaluate four main categories of NoSQL and also NewSQL databases. In particular we analyze and compare the following DB characteristics:

- Data model
- Querying possibilities
- Concurrency control
- Replication
- Scalability

- Partitioning
- Consistency
- Security features/ drawbacks
- Use cases/ applications suitability
- Popularity
- Performance

The rest of this paper is structured as follows. Section 2 provides the background information to the survey. It discusses Big Data challenges, the CAP Theorem and the critical reception of NoSQL/ NewSQL movement. Section 3 introduces taxonomy used throughout this paper. It discusses databases popularity and different data models. Section 4 compares NoSQL/ NewSQL databases types by various features, such as querying possibilities, concurrency control, replication, scalability, partitioning, consistency and security. Section 5 discusses applications and scenario suitability. Section 6 surveys several articles discussing performance. Section 7 provides the description of the Final Project. It discusses the implementation and thoroughly analyzes practical results that were obtained. Finally, Section 8 presents our conclusions.

2. Background

This chapter provides a brief background for NoSQL / NewSQL survey. Here we discuss Big Data challenges, ACID (Atomicity, Consistency, Isolation, and Durability) properties versus BASE (Basic Availability, Soft state, and Eventual consistency) and the CAP Theorem (also known as Brewer's theorem). This chapter also covers the critical reception of NoSQL movement and in particular focuses on Prof. Stonebraker's opinion.

2.1. The CAP Theorem

Current NoSQL trend is motivated by the tremendous growth of data mostly in web and mobile domain. If we consider Social Networking web pages such as Facebook, LinkedIN and Twitter, which are dealing with thousands of terabytes of data then, we must notice that besides handling huge data volume, those systems still have to maintain a reasonable latency, meaning that read and write are supposed to be responded very promptly. These new requirements for a data volume and processing speed sometimes are referred as the Big Data challenge.

The term Big Data was originally used by Doug Laney in his research report in 2001 [3]. Nowadays the widely accepted interpretation defines Big Data as 3 V's:

1. Variety: data today is present in varied formats be it text, video, images, sound and much more.
2. Velocity: the rate at which data is generated is far beyond imagination
    a. E.g. around 100 terabytes of data is uploaded daily on facebook.com.
    b. YouTube users upload 48 hours of video every minute.
3. Volume: terabytes of data being processed daily requires efficient techniques to store and process data

Figure 1 describes the increase in data volume, variety and complexity while new Big Data components are introduced.

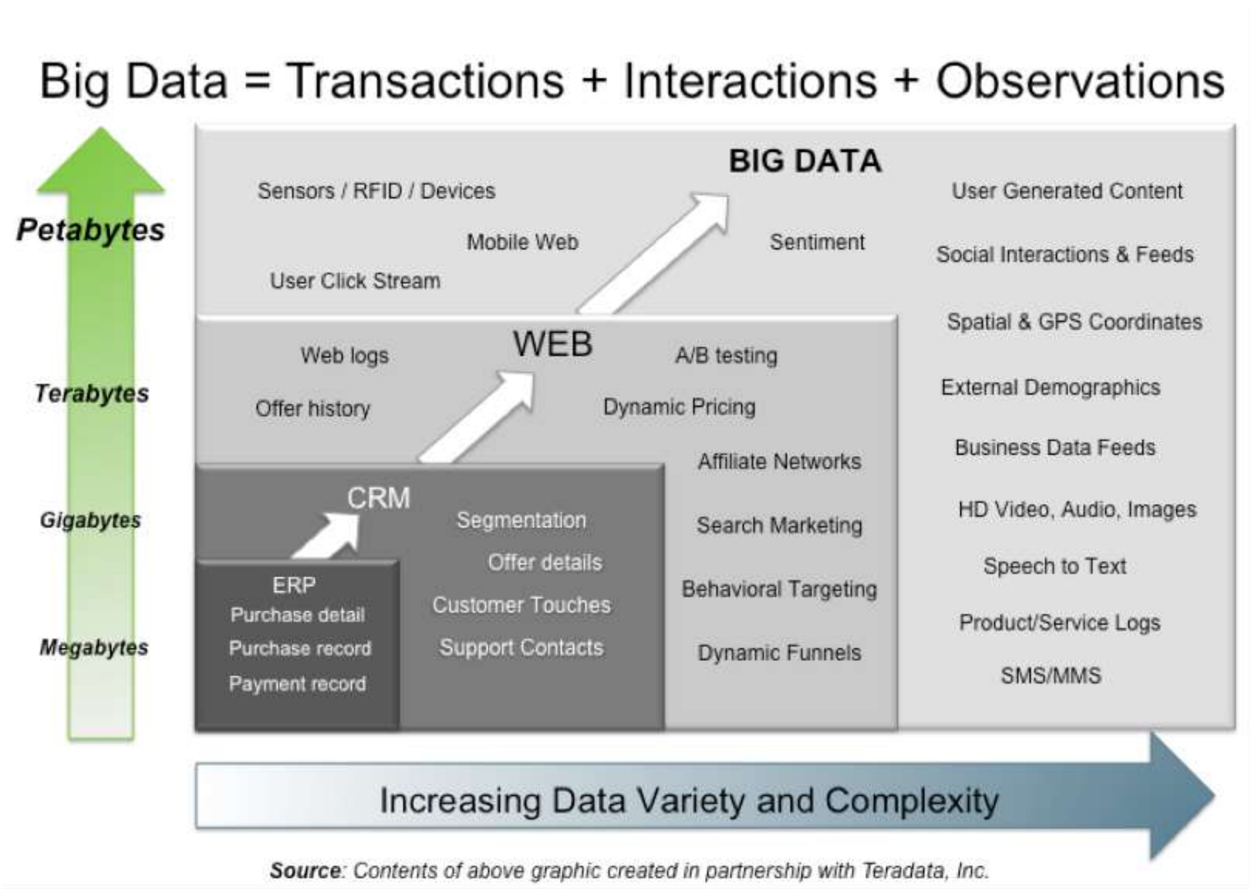


Figure 1: Big Data Transactions with Interactions and Observations [57]

The traditional relational database is facing many new challenges with large scale and high concurrency while handling big data and moreover some of the researchers in the field [11] consider traditional “old” DB wisdom as obsolete.

While traditional relational DB still hold the biggest part of the market, some of the modern Internet corporations and other companies turned their gaze toward another data store paradigm, the so called NoSQL. The reason why NoSQL has been so popular the last few years is mainly because, when a relational database grows out of one server, it is no longer that easy to use. In other words, they don't scale out very well in a distributed system.

NoSQL stands for “Not Only Structured Query Language” and usually define a family of data stores that forfeits the ACID properties of consistency and isolation in favor of “availability, graceful degradation, and performance” or BASE.

The acronym ACID stands for

- Atomicity: all of the operations in the transaction will complete, or none will.
- Consistency: transactions never observe or result in inconsistent data.
- Isolation: the transaction will behave as if it is the only operation being performed.
- Durability: upon completion of the transaction, the operation will not be reversed, but will be persistent

The term BASE was defined by Eric Brewer [6] who is also known for formulating the CAP theorem. BASE stands for Basically Available, Soft state, Eventual consistency and Brewer himself does admit that the acronym is contrived [9]:

- Basically Available - indicates that the system does guarantee availability, in terms of the CAP theorem.
- Soft state - indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.
- Eventual consistency - indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

The CAP Theorem coined by Brewer in 2000, states that it is impossible for a distributed service to be consistent, available, and partition-tolerant at the same instant in time [6].

Proponents of NOSQL often cite Eric Brewer's CAP theorem which is summarized as following: a system can have no more than two out of the following three properties

- Consistency:
- Availability
- Partition Tolerance

In other words, according to Brewer, the BASE model gives up on ACID property of consistency and isolation in favor of "availability, graceful degradation and performance".

Figure 2 describes the characteristics of NoSQL systems in terms of the CAP Theorem.

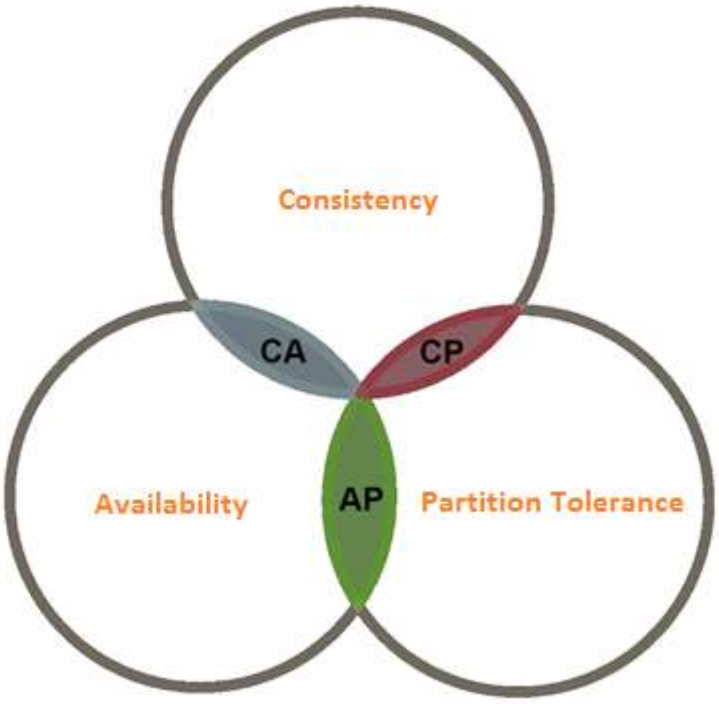


Figure 2: Different properties that a distributed system can guarantee at the same time, according to Brewer's theorem

In 2012 Eric Brewer has released another article providing some details and more up-to-date vision of the subject. That paper mentioned that "In ACID, the C means that a transaction preserves all the database rules, such as unique keys. In contrast, the C in CAP refers only to single-copy consistency, a strict subset of ACID consistency. ACID consistency also cannot be maintained across partitions; partition recovery will need to restore ACID consistency. More generally, maintaining invariants during partitions might be impossible, thus the need for careful thought about which operations to disallow and how to restore invariants during recovery." [10]

ACID	BASE
Strong consistency for transactions highest priority Availability less important Pessimistic Rigorous analysis Complex mechanisms	Availability and scaling highest priorities Weak consistency Optimistic Best effort Simple and fast

Table 1: ACID vs BASE per Brewer [6]

In his articles Eric Brewer contrasts ACID vs BASE (see Table 1), however in general those two concepts are usually not considered as an alternatives, but rather as a spectrum.

2.2. Critical reception

2.2.1. Skepticism

There are various aspects on which NoSQL movement have been criticized, one of them is the business side. In the article about NoSQL meetup in San Francisco in 2009 [42] the authors mention business related issues “preventing” or disturbing wide spread of NoSQL technologies. The fact that most of the NoSQL solutions are open-source software is very highly appreciated by the developers’ community. However, “sometime this is a “scary” situation for business people, who are not willing to end up with failures with nobody to blame for it” [65]. Since then the situation was slowly changing. For example for open-source Cassandra [25] one can use and Enterprise Edition version that is support by DataStax [20]. Most of the NewSQL solutions are free only for “sand box” purposes and require a license for commercial use.

Some of other NoSQL criticist are just being sceptics saying that “any new technology provokes a considerable enthusiasm and there is nothing new in the NoSQL/ NewSQL movement. The participants of NoSQL meetup provided actually a pragmatic advice on such remarks: they said that there is no need to switch to NoSQL if your current RDBMS solution does a job, there is simple no reason to replace that [42].

2.2.2. Stonebraker view

Michael Stonebraker is a computer scientist specializing in database research. Through a series of academic prototypes and commercial startups, Stonebraker's research and products are central to many relational database systems on the market today. He is also the founder of a number of database companies, including Ingres, Postgres and VoltDB [49].

Prof. Michael Stonebraker has been a huge critic of the NoSQL movement from the very beginning. In his publication from 2009 [39] Prof. Stonebraker defines two main reasons for moving towards non-relational data stores: flexibility and performance. The flexibility argument according to Stonebraker basically means that there might be a data that does not fit into a rigid relational model and which is bound too tightly by the structured of a traditional RDBMS. In this case something more flexible might be needed. As to the performance argument: Stonebraker mostly focuses on the workloads for which NoSQL databases are most often considered update- and lookup- intensive OLTP workloads . He sees 2 main options to improve OLTP transactions performance:

1) Horizontal scaling archived by automatic sharding, meaning that performance gets improved by adding new nodes. This has been done by traditional DBMS many years ago, so "nobody should ever run DBMS that does not do this".
2) Better performance of a single node while handling OLTP.

Stonebraker concentrates in that work [39] on the second option and he and his team analyze factors that impact performance of a single node. They have found out that this overhead has "nothing to do with SQL" and the degradation is primarily caused by the following five factors:

- Communication
- Logging
- Locking
- Latching
- Buffer Management

Prof. Stonebraker claims when these factors are eliminated the performance of single node has been improved drastically by the order of magnitude [39] [40].

It worth to mention that currently he's advocating NewSQL technologies and he was involved into development of some of them including VoltDB [38].

I have also reviewed other related articles by Michael Stonebraker in the research-seminar, the seminar report can be found in the Appendices to this work [Appendix A].

3. Taxonomy

In this chapter we introduce the taxonomy which is used throughout this paper. First, we discuss databases popularity. After that we describe various data models surveyed in this paper, such as: Key-value stores, Document stores, Column family stores, Graph databases and NewSQL databases.

3.1. Popularity

In this final paper we survey general taxonomy of NoSQL/NewSQL databases, define major categories based on Data Model and will focus on the most popular database systems representing each one of the categories.

To determine which data stores are "most prominent" we evaluate popularity of the databases using DB-Engine Ranking [34]. This system ranks database systems according to their popularity by using the parameters like:

- General interest according to Google Trends.
- Number of mentions on Web sites, measured as number of results in search engines queries. . At the moment Google and Bing are used for this measurement.
- Frequency of technical discussions on the Web. To count this measure they use the number of related questions and the number of interested users on the well-known IT-related Q&A sites Stack Overflow [31] and DBA Stack Exchange [32].
- Number of job offers. Here they use the most popular professional network measured by means of LinkedIn [33] social network.

- Number of professional profiles in which the solutions are mentioned (measured by the leading job search engines).
- Relevance in social networks according to the number of Twitter tweets, in which the system is mentioned.

DB-engines ranking portal also provides a ranking per various DB types categories. Next we compare several NoSQL models and systems based on their popularity.

Figure 3 shows the popularity ranking of Document Stores in May 2015.

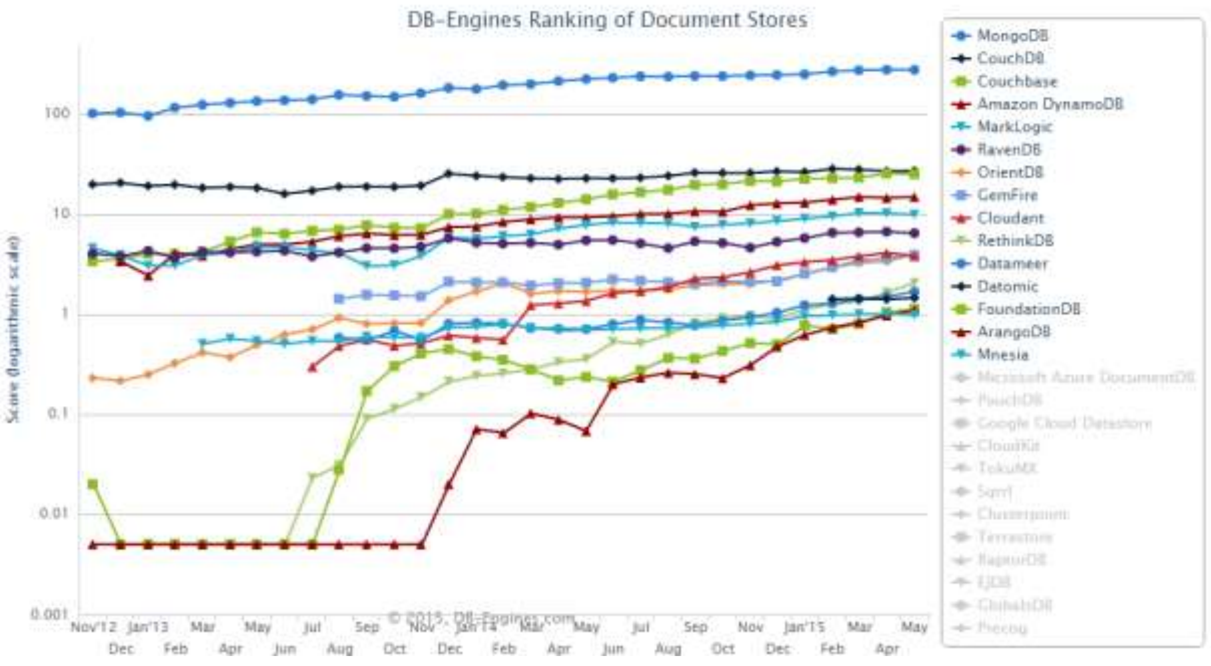


Figure 3: NoSQL databases ranking – May 2015 [34]

Figure 4 shows that currently Neo4J is by far the most popular Graph database.

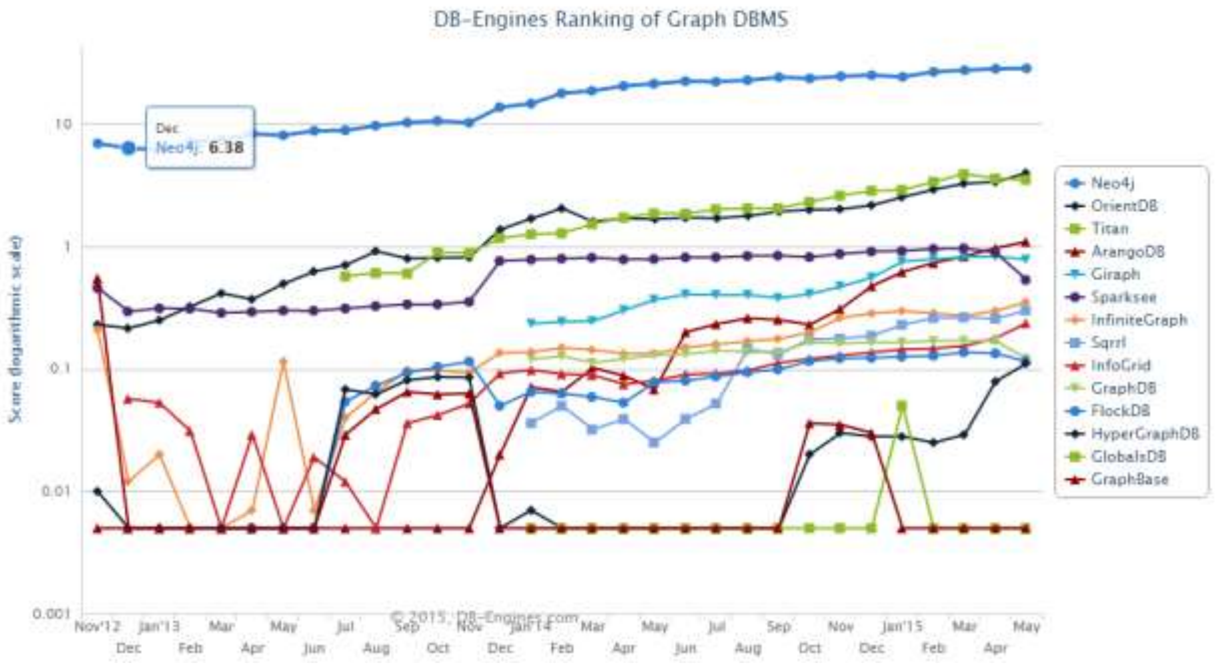


Figure 4: Graph DB ranking as of May-2015

The graph in the Figure 5 shows the historical trend of the popularity of various database categories.

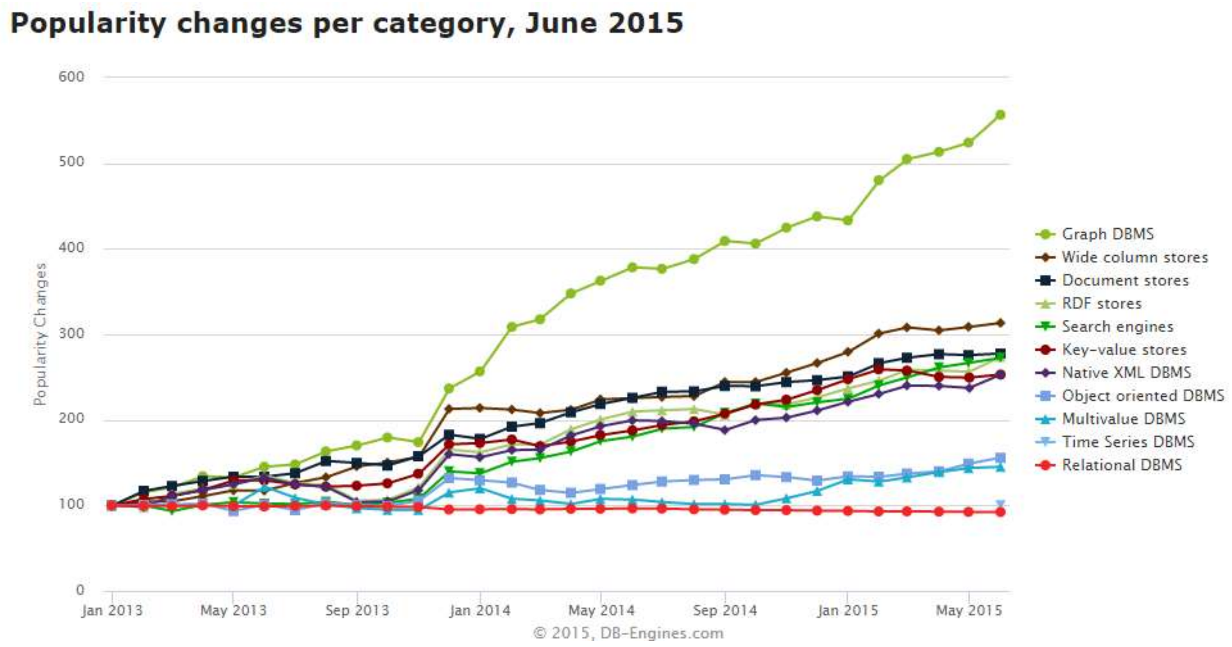


Figure 5: Popularity changes per category, June-2015 [34]

As it can been seen from the chart (Jun-2015) above, the Relational Databases popularity trend is decreasing, while the most popular trending categories are:

- Graph DB
- Column-family
- Document stores

In the next chapter we provide a detailed description for these three categories as well as for Key-value stores.

3.2. Data Model

NoSQL data stores vary widely by data model and have some distinct features on its own. In this paper we review the taxonomy of NoSQL databases based on the data model, which usually propose following categories:

- Key-value stores
- Document stores
- Column family stores
- Graph databases

In addition to NoSQL systems we also review NewSQL data stores category, which is essentially a hybrid between NoSQL and relational databases.

3.2.1. Key-value stores

Key-value data stores sometimes considered to be the “simplest” form of data base. Those DBs are usually schema-less. The data is stored in a form of a pair key→value, so KV data model resembles structure similar to Associative or Hash Array data structure (also known as a map or dictionary). Keys are used as indexes to fetch the data (value) and this makes those data stores much more efficient for data retrieval than classical RDBMS.

Figure 6 provides an example illustrating simple key-value data.

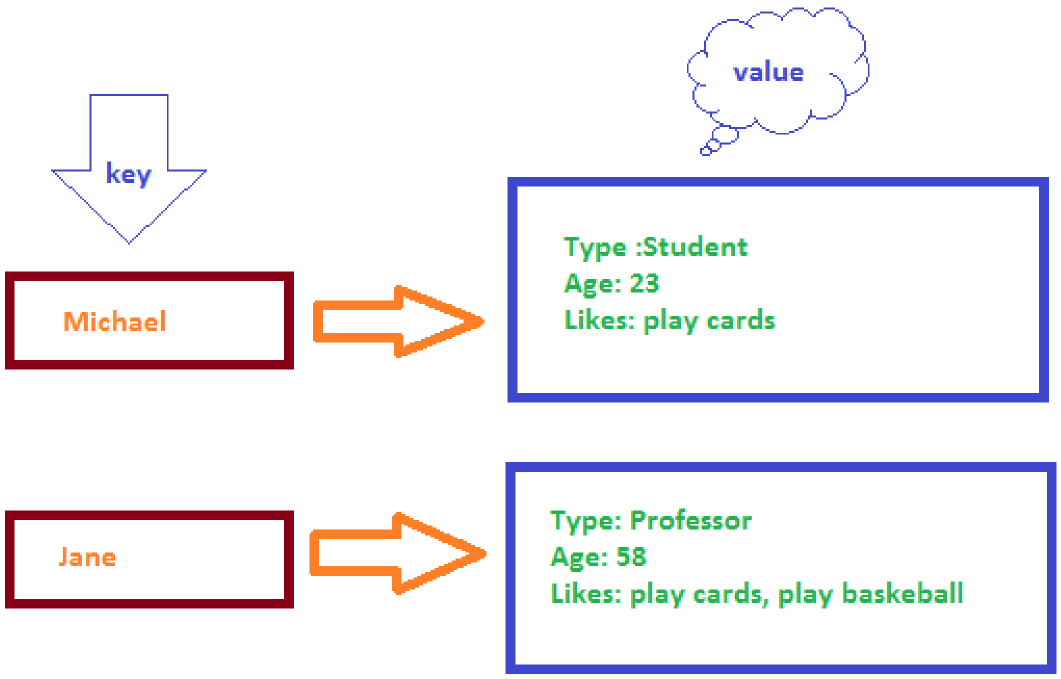


Figure 6: Key/Value Store NoSQL Database - data example

KV data model is very simplistic and sometimes it's used as a baseline while more complicated data models are implemented on top of it. The key-value model can be extended to an ordered model that maintains keys in lexicographic order. This extension is powerful, in that it can efficiently process key ranges [14].

In terms of the CAP theorem those databases usually prefer availability over consistency. Example key value databases include Voldemort [22] and Amazon Dynamo DB [1]. Amazon Dynamo DB model provides a fast, highly reliable and cost effective NOSQL database service designed for internet scale applications. It offers low, predictable latencies at any scale [1].

Further in this final paper we'll focus on Project Voldemort as an example of KV store. Project Voldemort is an advanced key-value store, written in Java [21, 22]. It is open source, with substantial contributions from LinkedIn [33].

3.2.2. Document-based stores

Document-based databases are probably most popular among other NoSQL types and their popularity keeps growing. These databases store their data in a form of documents. In general this type of data bases encapsulates "key-value" concept, while key is an ID of the document and the value is the document itself, which can be retrieved by ID. Various formats can be used as a metadata for document oriented DBs: XML, JSON and some others.

In contrast with the traditional RDBMS, where every row follows the schema, in document-oriented DBs each document may have a different structure. And usually document oriented stores provide additional indexing based on document contents. That's one of the main enhancements of document stores over the more basic key-value store model.

Both provide querying mechanism based on the "primary key", but in document store model one usually able to query data also by the value (document) contents. Similarly to KV-stores, this type of DB systems is less efficient when application requires multiple-key transactions.

Figure 7 provides a visual demonstration of the differences between the Relational data model and the Document data model.

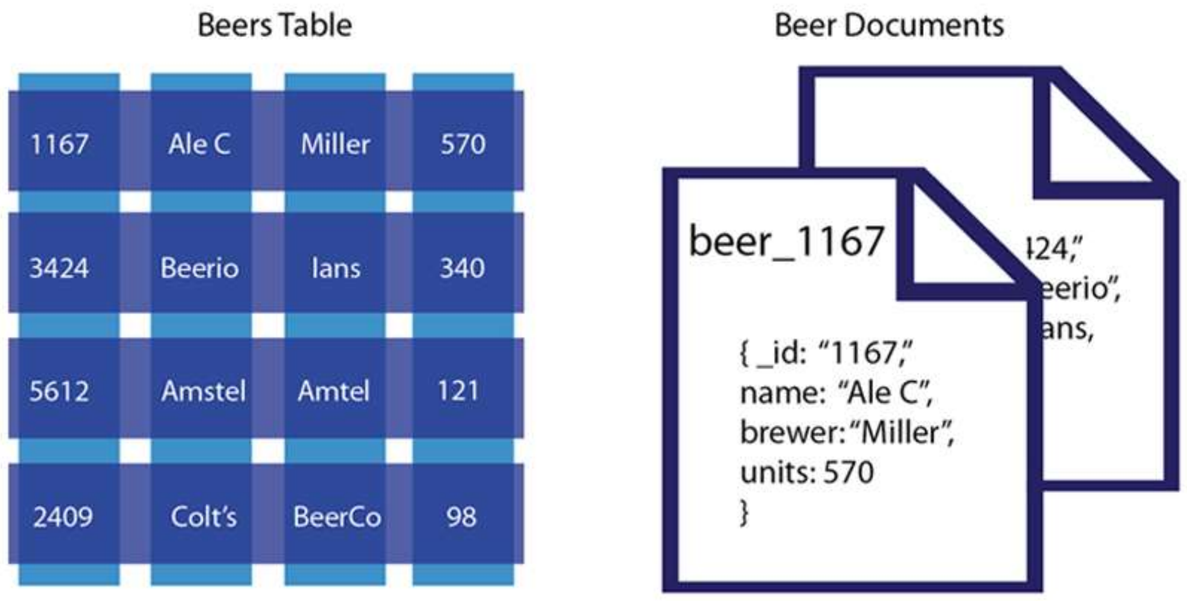


Figure 7: Comparing document-oriented and relational data [23]

In this paper we are going to survey MongoDB which the most popular representative of document data store category [34]. MongoDB (from "humongous") is a cross-platform document-oriented database [24]. This is a GPL open source data store which written in C++ and supported by 10gen.

3.2.3. Column family stores

The standard column family is a NoSQL object which contains columns of related data [17]. The approach of accessing and processing data by column rather than by rows has been used in analytic tools for quite a long time. However the subclass of NoSQL Databases called Column-family stores refers specifically to systems derived from Google Bigtable. Google researchers describe this model as "sparse, distributed, persistent, multidimensional sorted maps" [15]. In Bigtable the dataset consists of several rows, each can be addressed by a key – primary key. Some of the most popular Data stores in this category implement the basic Google Bigtable model, e.g. HBase

[16]. While others, like Cassandra extend the model by introducing super-columns, where the value can be column-family by itself [25]

The term Column family means a pair that consists of key and value, where the key is mapped to a value that is a set of columns. In analogy with relational databases, a column family can be considered as a "table", while each key-value pair represent a "row".

However using this analogy we may illustrate one of the main differences between traditional RDBMS model and Colum-family model, which is the fact that the same “table” (column family) can contain different columns and different number of columns, while the traditional relational model is absolutely strict about this.

There is also a similarity to the most basic key-value model, since the row key functions as a “key”, while the set of columns resembles the “value”. Due to the data model specifics Column-families usually do not handle complex relational logic. Hence as in case of the basic Key-value stores, if complex relational querying functionality is required, then it has to be implemented in the client side.

Figure 8 describes a layout of a single row within Column Family and a row within Super Column Family.

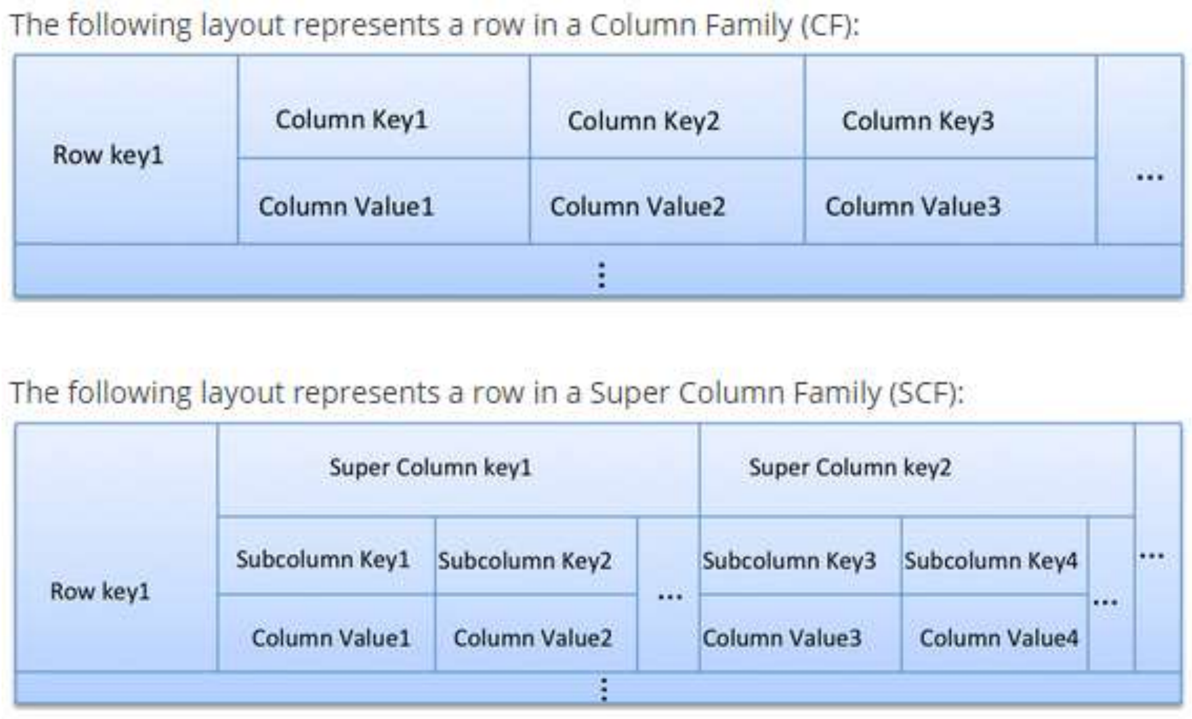


Figure 8: Column Family / Super Column family layout [26]

In this final paper we'll be focusing on Cassandra [25]. Cassandra is Column-family (or extensible record) database written in Java, and used under Apache licensing. It is heavily backed by DataStax [20], and was originally open sourced by Facebook in 2008 [18]. See section 6.1 for additional information about Cassandra.

3.2.4. Graph databases

Graph data bases is the model which has its origins in the graph theory. Moreover the data model of these NoSQL databases is based on a graph structure. Basically it means that data is stored in a form of nodes (vertices), while relations between data are presented as edges that interconnect the vertices.

These data stores considered to be NoSQL data bases, because these systems don't offer SQL support and the data model is not similar to relational. Nevertheless many of the Graph data bases, including Neo4j [19], are fully ACID-compliant.

Another difference between this sub-group and other NoSQL categories is that this model is not an extension of "key-value" concept and it's more efficient for storing interconnected data and handling relational querying. Thus naturally they are more suitable for dependency analysis problem solving and some of the social networking scenarios.

While being effective in those areas, graph databases might be less suitable in other Big Data problems handling. In particular, these data bases are not that efficient on horizontal scaling as key-value or column-families. This "weakness" stems from the fact that if related data is stored on a different servers, than traversing such a graph may be very "expensive" operation in terms of performance.

Figure 9 provides a graphical example of NoSQL Graph Database.

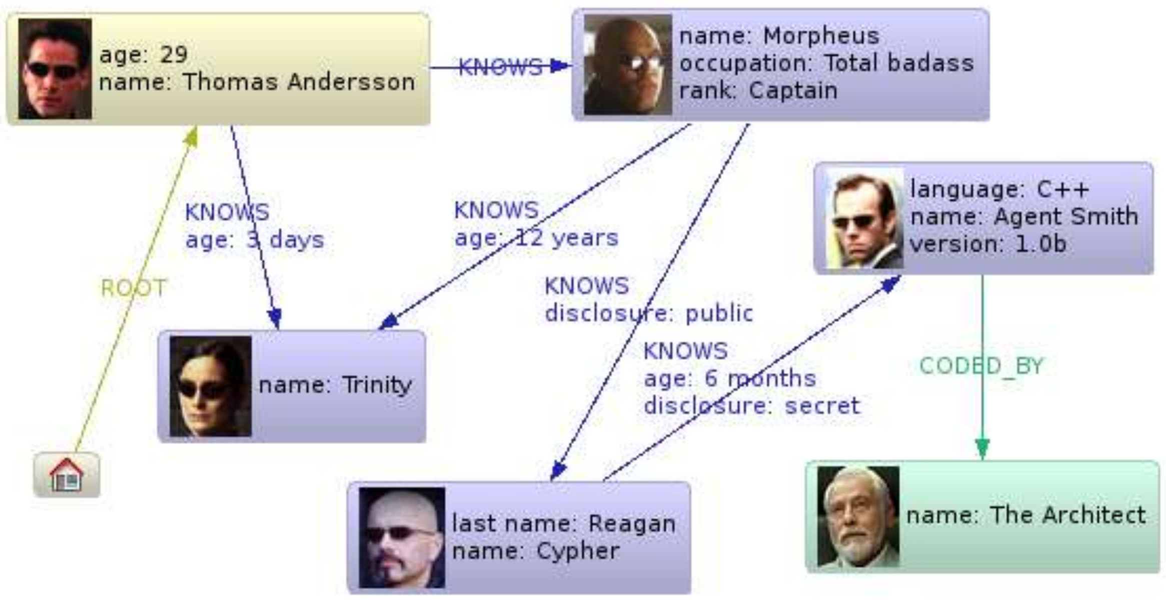


Figure 9: Graph NoSQL Database example [27]

The most popular graph database is Neo4j (see Figure 4) and we're going to analyze and survey it in the following chapter of this paper as a representative of this database category. Neo4j is an open-source graph database, implemented in Java [19]

3.2.5. NewSQL

NewSQL is another class of modern database management systems. Until recently implementing of a scale-out architecture required some NoSQL solution because old-fashioned relational databases didn't provide a good support for "horizontal" scale.

As it was mentioned above such NoSQL solutions usually didn't provide ACID and rather provided some type of eventual consistency. This tension is what inspired NewSQL movement.

NewSQL solutions by definition are based on the relational model. Besides that these databases seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads, while still maintaining the ACID guarantees of a traditional database system [28].

However even though they use SQL as main interface language and clients interact with NewSQL in traditional relational DB terms such as "tables" and "relations", the actual internal representation might be absolutely different from that of traditional DB. For example, NuoDB can store its data into key-value store [29].

The term "NewSQL" has been created by Matt Aslett from "the 451 group". He writes: "'NewSQL' is our shorthand for the various new scalable/high performance SQL database vendors. We have previously referred to these products as 'ScalableSQL' to differentiate them from the incumbent relational database products. Since this implies horizontal scalability, which is not necessarily a feature of all the products, we adopted

the term ‘NewSQL’ in the new report.
And to clarify, like NoSQL, NewSQL is not to be taken too literally: the new thing about the NewSQL vendors is the vendor, not the SQL.” [36]

According to Prof. Stonebraker the NewSQL goals are to bring the benefits of the relational paradigm to distributed architectures or to provide such good performance that horizontal scalability is no longer a necessity [28].

Table 2 provides a comparison of the main characteristics of OldSQL, NoSQL and NewSQL. For instance, only NoSQL databases are considered to be schema-less, meaning that they allow to store unstructured data without prior knowledge of the schema; traditional SQL (“OldSQL”) or NewSQL systems don’t allow it.

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance/ big volume	No	Yes	Yes
Schema-less	No	Yes	No

Table 2: The “NewSQL promise” in brief [41]

In this paper we’re focusing on the following two NewSQL data stores:

- VoltDB – a new RDBMS designed for high performance (per node) as well as scalability [37]. VoltDB is an open-source system (AGPL v3.0 license) written in Java and C++ [38].
- NuoDB – another distributed database which is SQL compliant. It’s is defined as “client/cloud relational database” [29]. NuoDB is a “closed source” system. It’s available in Amazon Web Services (AWS) marketplace as a service, as an appliance and as a stand-alone software.

4. Features comparison

This section compares NoSQL/ NewSQL database types by a number of determinative features, such as querying possibilities, concurrency control, replication, scalability, partitioning, consistency and security.

4.1. Query Possibilities

In general data models are tightly coupled with query possibilities. Therefore the analysis of the queries that will be needed for an application is an important process in order to find a suitable data base solution. Various NoSQL/ NewSQL data stores not only differ in the provided data model, but also have different APIs and interfaces to interact with them. This again is directly dependent upon the data model utilized by the data store.

For instance, key-value stores usually cannot provide value-based querying. The values in KV data stores are opaque and they only provide key-based “put”, “get” and “delete” operations. Thus any query language would be unnecessary overhead for these stores. And in systems when additional more complex query functionalities are required, they have to be implemented on the application layer, which can lead to much more system complexity and performance penalties. Therefore key-value stores shouldn’t be used in applications where complex queries or queries based on values are required.

On the other hand document stores offer richer API including queries based on the value. This type of data stores provide range queries on values, secondary indexes, querying nested documents and operations like “and”, “or” and “between”. MongoDB queries can be extended with regular expressions and besides that MongoDB provides a support to operations such as count and distinct [24].

Most of document stores also support REST interfaces [43]. However those data stores do not provide any query language with SQL-like syntax.

Column family stores usually support range queries and operations “in”, “and”, “or” and regular expressions. However those operations can be applied just on the row keys. Even if CF data store provide query language (e.g. CQL for Cassandra) – only row keys can be considered in the where clause and not the values.

Graph data bases provide query possibilities in two ways: graph pattern matching and graph traversal. Pattern matching usually refers to a mechanism which performs a search of parts of the original graph which will match the pattern. The graph traversal is another graph search technique which traverses the graph according to some description and starting from the chosen node. Traversal abilities may support various strategies as BFS (breadth first search) and DFS (depth first search).

Most of the graph data bases offer REST API and additionally in the graph databases realm there is a common language supported by multiple graph databases SPARQL [44] including Neo4j [19].

REST API is an API (Application Program Interface) that adheres to the principles of REST (Representational State Transfer). REST is an architectural style that uses simple HTTP calls for inter-machine communication. Using REST means API calls will be message-based and reliant on the HTTP standard [43].

In summary, various NoSQL data stores classes differ significantly in their query capabilities and it’s absolutely necessary to consider specific application requirements in order to choose the most suitable NoSQL data base type with appropriate data model and offered interfaces.

On the contrary, most of the NewSQL data stores are providing SQL support as one of their main features. However NuoDB is considered to be more SQL-compliant, while VoltDB has various restrictions in place, for instance: it's not possible to use "having" clause, tables can't join themselves and all joined tables must be partitioned over the same values.

4.2. MapReduce

MapReduce is a parallel programming model for large data sets processing introduced by Google. MapReduce is typically used to do distributed computing on clusters of computers [45].

In this model, a user specifies the computation by two functions, Map and Reduce:

- In the mapping phase, MapReduce takes the input data and feeds each data element to the mapper.
- In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result. In simple terms, the mapper is meant to filter and transform the input into something that the reducer can aggregate over.

The underlying MapReduce library automatically parallelizes the computation, and handles complicated issues like data distribution, load balancing and fault tolerance.

The original MapReduce implementation by Google, as well as its open source counterpart, Hadoop [46], is aimed for parallelizing computing in large clusters of commodity machines. Map Reduce has gained a great popularity as it gracefully and automatically achieves fault tolerance. It automatically handles the gathering of results across the multiple nodes and returns a single result or set. MapReduce model advantage is the easy scaling of data processing over multiple computing nodes.

Now let’s consider an example for MapReduce – WordCount application, which reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab. The WordCount is given below in pseudo code:

```
1. class Mapper
2.    method Map(docid id, doc d)
3.       for all term t in doc d do
4.          Emit(term t, count 1)
1. class Reducer
2.    method Reduce(term t, counts [c1, c2,...])
3.       sum = 0
4.       for all count c in [c1, c2,...] do
5.          sum = sum + c
6.       Emit(term t, count sum)
```

Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1.

Each reducer sums the counts for each word and emits a single key/value with the word and sum.

Figure 10 shows an example of MapReduce execution diagram.

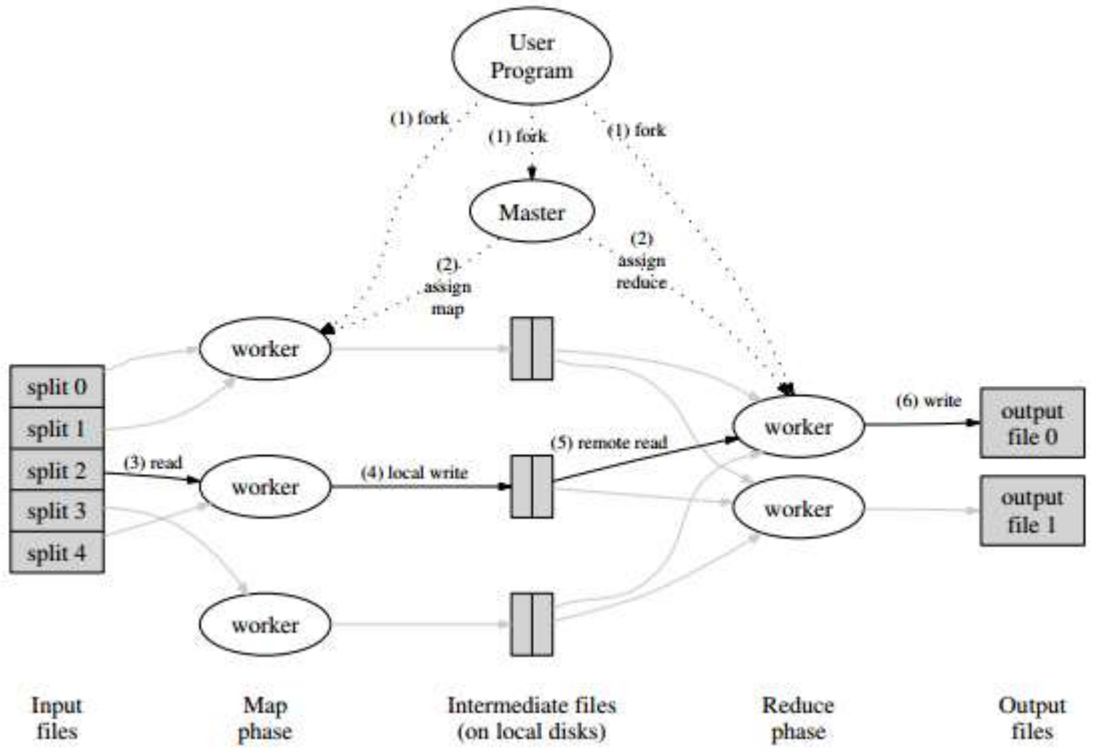


Figure 10: MapReduce Execution Overview [45]

When applied to databases, MapReduce means to process a set of keys by submitting the process logic (map and reduce functions code – see above) to the storage nodes, which in their turn locally apply the map function to the keys that they own.

Clusters of document and column family stores are able to store huge amounts of structured data, therefore queries can get very inefficient if just a single machine has to process the required data. That's the reason why all document and column family stores provide MapReduce frameworks which enable parallelized calculations over large data sets on multiple machines.

NoSQL Key-Value stores, graph databases and NewSQL usually do not support MapReduce.

Table 3 summarizes querying capabilities of all the NoSQL and NewSQL systems surveyed in this final paper.

NoSQL/NewSQL databases		***Querying capabilities***			
		Rest API	***Query Language***	***Other API***	***MapReduce Support***
Key Value Store	*Voldemort*	*Yes*	*No*	*Clients for several languages*	*Yes*
Document Store	*MongoDB*	*Yes*	*No*	*CLI and API in different languages. Support Thrift interface*	*Yes*
Column Family Store	*Cassandra*	*Yes*	*Cassandra Query Language (CQL)*	*CLI and API in different languages*	*Yes*
Graph Database	*Neo4j*	*Yes*	*Cypher, Gremlin and SparQL*	*CLI and API in different languages*	*No*
NewSQL	*VoltDB*	*Yes*	*SQL*	*CLI and API in different languages. JDBC support.*	*No*
	NuoDB	*No*	*SQL*	*CLI and drivers for most common data access APIs (JDBC, ODBC, ADO.NET). Also provides a C++ support.*	*No*

Table 3: The NoSQL/ NewSQL Querying capabilities

4.3. Scalability

Vertical scalability is an ability to scale up or to add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer.

Application scalability refers to the improved performance of running applications on a scaled-up version of the system

Traditional databases were designed primarily to scale vertically by adding more power to a single node.

Horizontal scalability is an ability to scale out or to add more nodes (servers) to the system. As computer prices have dropped and performance continues to increase, high-performance computing applications have adopted low-cost "commodity" systems for tasks that once would have required supercomputers. Size scalability is the maximum number of processors that a system can accommodate. Horizontal scalability is one of the main characteristics of NoSQL and NewSQL systems.

There are tree aspects that are considered with regards to scalability:

- scaling read requests
- scaling write requests
- scaling storage

In the next chapters we're focusing on the main strategies which have major influence on these scaling capabilities in NoSQL/NewSQL: Concurrency Control, Replication, Partitioning and Eventual Consistency.

4.4. Concurrency Control

Concurrency control is of special interest in NoSQL and NewSQL data stores, because they generally need to accommodate a huge amount of users that have access to a data source in parallel.

Traditional databases (RDBMS) use pessimistic concurrency strategies with exclusive access on a dataset. Pessimistic concurrency control, or pessimistic locking assumes that two or more concurrent users will try to update the same record at the same time. To prevent this situation a lock is placed onto the accessed entity, so that exclusive access is guaranteed to a single user operation only, other clients accessing the same object will have to wait until the initial one finishes its work. These strategies can be suitable, if costs for locking are low. However in database clusters which are distributed over large distances, pessimistic consistency strategies can lead to performance degradation, especially when applications have to support high read request rates.

Optimistic concurrency control or optimistic locking assumes that conflicts are possible, but they are not common. So before changed data is committed, every transactions checks whether another transactions made any conflicting modifications to the same datasets. If there are conflicts identified, the transaction will be rolled back. This strategy can work well if updates are very rare, and therefore chance for conflicting are relatively low. In this situation rolling back will be cheaper that locking data set exclusively as in pessimistic locking [7].

Several of the data stores including Voldemort and NuoDB implement multi version concurrency control (MVCC). In concurrent access is not managed with locks but by organization of many unmodifiable chronological ordered versions. Multiple versions are stored, but only the one is marked as current, all the rest are marked as obsolete.

Using this strategy, a read operation can see the data as it was when it started reading, while a concurrent process can accomplish write operation on the same dataset in the meantime [47].

Besides consistency cutbacks MVCC also causes higher space requirements as multiple versions are kept in parallel. So usually to work with MVCC it's necessary to have a garbage collector that deletes no longer needed versions and also some conflict resolving algorithm to deal with inconsistencies. This causes additional system complexity.

A number of NoSQL databases allow applications to implement optimistic concurrency control by providing primitives such as “check and set” (CAS). The CAS method is used to ensure that a write will be performed only if no other client operation performed another write since the data was last read.

Some of the NewSQL solutions also implement innovative approaches to concurrency control. NuoDB is primarily relying on the MVCC as discussed above. On the contrary, VoltDB has a different approach regarding the concurrency. This data store assumes that the whole data base can fit into memory, meaning that there is enough memory and transactions are short-lived. Based on these assumptions, transactions are executed sequentially in a lock-free, single – threaded environment.

Table 4 summarizes concurrency control mechanisms used in NoSQL/ NewSQL databases.

NoSQL/NewSQL databases		***Concurrency Control***
Key Value Store	*Voldemort*	*MVCC with vector clock*
Document Store	*MongoDB*	*Readers-writers lock*
Column Family Store	*Cassandra*	*Client-provided timestamps are used to determine the most recent update to a column. The latest timestamp always wins and eventually persists*
Graph Database	*Neo4j*	*Write locks are required on nodes and relationships until committed*
NewSQL	*VoltDB*	*Single threaded model, no concurrency control*
	NuoDB	*MVCC*

Table 4: The NoSQL/ NewSQL concurrency control techniques

4.5. Replication

Besides the increasing scalability and improved performance though load balancing, replication also brings better reliability, fault tolerance, and durability. As we have already mentioned in the “CAP Theorem” chapter Eric Brewer noticed, that only full availability or full consistency can be guaranteed at one time in distributed systems [6]. Thus if all the replicas in the master are updated synchronously, the system wouldn’t be available until all slaves had committed the operation. But if the messages got lost due to network problems, then the system won’t be available until all slaves had committed the write operation.

This solution might be not suitable for systems that rely on high availability, like Web retailers as Amazon [52] or internet trading systems as eBay [53]. Therefore these systems usually implement asynchronous or lazy replication which may result in a situation where reads on replicas might be inconsistent for a short period of time. This approach is use by a majority of NoSQL systems while those systems can be classified according to Brewer [6] as eventually consistent or providing BASE (as an opposite of ACID). On the contrary in synchronous or eager replication, all the changes are propagated to nodes prior to sending an acknowledgement about successful write operation. This actually means that an additional latency is introduced in execution of write operation, because it will incorporate all the propagation time. This may result in serious performance degradation and therefore this approach is rarely used in NoSQL systems.

Another determining characteristic is a mode of replication, there two types: master-slave and master-master replication.

The master-slave replication is a scheme where a single node is defined as master and this is the only node which accepts and processes write requests (see Figure 11.a). Then changes are being propagated from master to the slave nodes. Two of the four NoSQL databases considered in this final paper support master-slave replication: MongoDB and Neo4j.

In multi-master replication (shown in Figure 11.b) multiple nodes can process write requests, which is then propagated to the remaining nodes. So basically in master-slave the propagation works in one directions only: from master to slave, which in multi-master replication propagation can happen in various directions.

Figure 11 demonstrates differences between two types of replication: master-slave and multi-master.

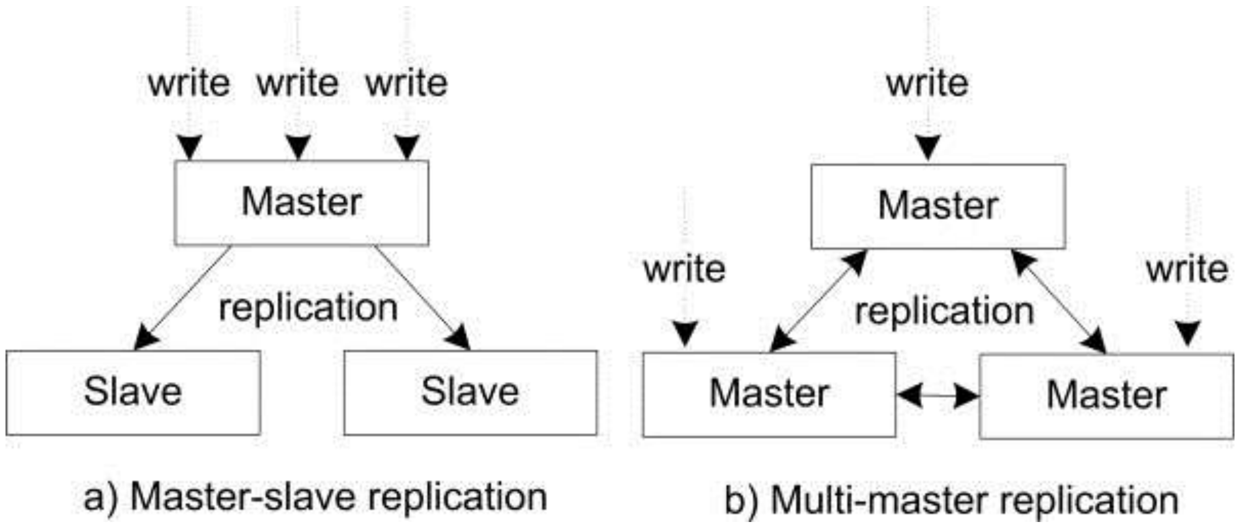


Figure 11: replication modes [7]

The remaining NoSQL databases which are surveyed in the final paper, Project Voldemort and Cassandra, support masterless replication, which is similar to multi master described above, because multiple nodes accept “write” requests. However it’s called masterless replication, because all nodes play the same role in replication system, so there is no master. Note that the data stores with masterless replication mechanism also use consistent hashing as a partitioning strategy. The replicas placement strategy is very much related to node position on partitioning ring.

In both Cassandra and Voldemort, the total number of replicas across the cluster is referred to as the replication factor [20] and it’s a configurable system parameter. The configuration tuning of RF have a major impact on the performance and we have practically observed that with Cassandra during one of the experiments we conducted as a part of the Final Project. We performed data loading using 3-nodes

Cassandra cluster: once with “replication factor” (RF) set to 1 and another time with RF set to 3, which makes the write operation fully durable. The result was interesting although theoretically predictable: loading time with higher RF setting was more than two times longer for the same size of the database (see Figure 17).

NewSQL replication models can be considered as masterless or multi-master, because any node can be receiving update statement. In NuoDB, the rows are internally represented as “in memory” objects which asynchronously communicate to replicate their state changes. VoltDB has a transaction/session manager which is responsible for getting the updates and forwarding it to all replicas to be executed in parallel.

Table 5 describes different replication techniques used in NoSQL/ NewSQL databases.

NoSQL/NewSQL databases		***Replication***
Key Value Store	*Voldemort*	*Masterless, asynchronous replication.* *Replicas are located on the first R nodes moving over the partitioning ring in a clockwise direction.*
Document Store	*MongoDB*	*Master-Slave, asynchronous replication.* *Designed for off-line operation. Multiple replicas can maintain their own copies of the same data and synchronize them at a later time.*
Column Family Store	*Cassandra*	*Masterless, asynchronous replication.* *Two strategies for placing replicas: replicas are placed on the next R nodes along the ring; or, replica 2 is placed on the first node along the ring that belongs to another data center, with remaining replicas on the nodes along the ring on the same rack at the first*
Graph Database	*Neo4j*	*Master-slave, but can handle write requests on all server nodes. Write requests to slaves must synchronously propagate to master.*
NewSQL	*VoltDB*	*Updates executed on all replicas at the same time*
	NuoDB	*Multi-master (distributed object replication).* *Asynchronous*

Table 5: The NoSQL/ NewSQL replication modes

4.6. Partitioning

When we're dealing with Big Data, while huge amounts of data and very high read and write request rates exceed the capacity of one machine, databases have to be partitioned across clusters. Traditional relational databases (RDBMS) usually do not support horizontal scalability, because of their normalized data model and ACID guarantees. Consequently doubling of number of servers in RDBMS clusters does not double the performance. This problem is one of the reasons why big Web players like Google, Amazon and Facebook started to develop their own data bases, which are designed to scale horizontally and therefore satisfy the very high requirements on performance and capacity of these systems [5].

There are multiple techniques of partitioning used by various data stores. Most NoSQL and NewSQL data stores do implement some kind of horizontal partitioning or sharding, which actually involves storing sets of rows into different segments/ shards. On the contrary the vertical partitioning involves storing sets of columns into different segments and distributing them accordingly. The partitioning strategy is strongly depending on data model. For example in Column Family data stores they usually support vertical partitioning, which is natural because partitioning segments containing predefined sets of columns.

Since data models of key-value, document data stores and column-families databases stores are very key oriented, the two major strategies related to horizontal partitioning are also key oriented.

- First is called range partitioning - this strategy distribute datasets to partitions residing on different servers based on ranges of a partition key. Initially routing server splits the whole dataset by the range of their keys. Afterwards every node is responsible for storing and read/write handling of a specific range of keys. The advantage of this approach is that query by range might become very efficient, because the neighbor keys usually reside on the same node (within the same range). However there are some disadvantages of this approach, such hot spots and load-balancing issues. For example, since the routing server is responsible load balancing, key load allocation and partition block advices - means that the processing will be always concentered in that one server (or a few servers when routing machine is replicated). So the availability of the whole cluster depends on that specific routing machine, if it fails it may seriously disturb cluster work.

- The second partitioning strategy called consistent hashing provides higher availability and simpler cluster structure comparing to range partitioning. This shared nothing technique works as follows: the dataset is represented as a ring or circle, which is divided into a number of ranges equal to a number of available nodes, and every node, is mapped to a corresponding point on the ring.

Figure 12 illustrates consistent hashing on an example with four nodes - N1 to N4.

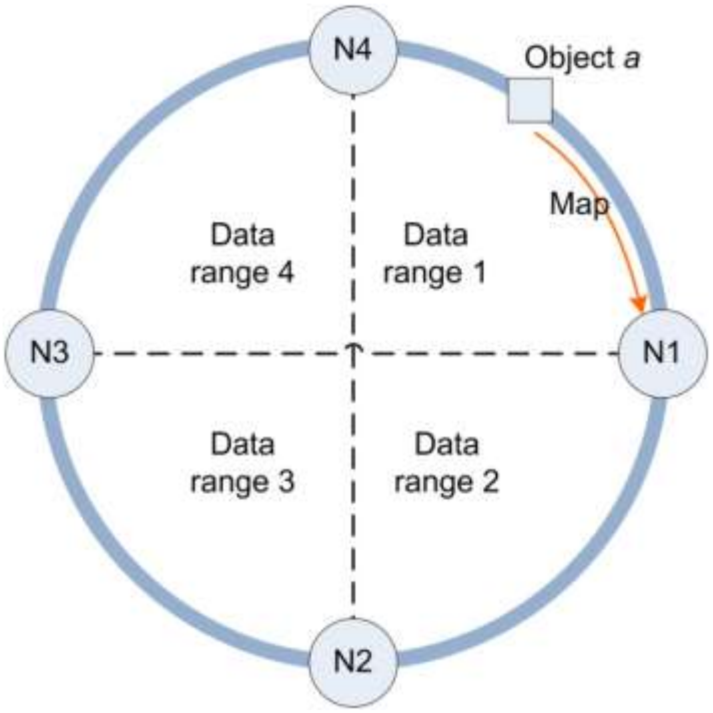


Figure 12: consistent hashing example [7]

- To determine the node where the data object should be placed, the system calculates the hash value of the object's key and finds its location on the ring. In the example from Figure 12, object a is located between nodes N4 and N1.
- After that the ring is walked clockwise until the first node is encountered, and the object gets assigned to that node. Accordingly, object a from Figure 12 gets assigned to node N1.
- Consequently, each node is responsible for the ring region between itself and its predecessor; in Figure 12: node N1 is responsible for data range 1, node N2 for data range 2, and so on [7].

One of the advantages of the consistent hashing is that there is no need for a mapping service as in range partitioning and the target location of an object can be calculated very fast. Besides that this approach is also efficient in dynamic resizing: if nodes are added to or removed from the ring, only neighboring regions are to be reassigned to different nodes, and the majority of records remain unaffected [7]. However, consistent hashing might have a bad impact on range queries because adjacent keys are distributed across a number of different nodes.

Among the systems surveyed in this paper the following use consistent hashing: Project Voldemort, Cassandra and VoltDB.

Partitioning in graph databases is much more challenging problem than in other NoSQL databases. Key-value stores, column family databases document stores usually offer partitioning base on the key value, which is relatively stable and can be looked up by using searching techniques. On the contrary, graph databases do not have stable keys and those databases perform lookups, but by exploiting relations among entities.

Therefore there are 2 contradicting requirements with regards to the graph databases: first all the related nodes of the graph should be kept on one node in order to enable efficient traversal though it, on the other hand we can't keep too many nodes on the same server, because it may cause heavy load and other performance issues. There are multiple graph data base partitioning mechanisms proposed [60], however their implementation is very limited in the actual Graph DB systems, because intensive rebalancing operations that may be a result of quickly changing data in the data store. The graph database system observed in this paper: Neo4j doesn't implement a partitioning; however it offers a cache sharding.

Sharding is a type of database partitioning that is used to separate very large databases into smaller, faster, more easily managed pieces called data shards [64].

NewSQL systems also provide various types of partitioning solutions. For instance VoltDB uses a consistent hashing approach in which each table is partitioned using a single key and rows are distributed among servers. Stored procedures can be executed on a single partition or on all of them; nevertheless, the drawback is that in VoltDB the user is responsible for selecting between these options. Another NewSQL solution analyzed in this final paper NuoDB uses a completely different approach for data partitioning [29]. A NuoDB deployment is made up of a number of Storage Managers (SM) and Transaction Managers (TM). The SMs are the nodes responsible for maintaining the data, while the TMs are the nodes that process the queries. Each SM has a complete copy of the entire data, which basically means that no partitioning takes place within the SM. Nevertheless, the underlying key-value store used by the SMs can partition the data by itself, although this is neither controllable nor viewable by the user.

4.7. Consistency

Consistency is very strongly related to the partitioning implementation discussed in the previous chapter. In general there are 2 major types of consistency that can be distinguished: strong consistency and eventual consistency.

Strong consistency ensures that when all the write requests are confirmed, the same data is visible to all the subsequent read operations. Usually strong (immediate) consistency can be obtained by synchronous replication or a complete lack of replication. However these 2 options are not acceptable for the purposes that NoSQL data stores are designed to, because it either introduces a large latency and impacts availability, or has a bad effect on scalability.

Therefore most of the NoSQL data bases, including the systems analyzed in this paper, are implementing the 2nd type of consistency – eventual consistency. In the eventual consistency model the changes propagate to replicas through the system given sufficient time. It means that some nodes may be outdated (inconsistent) for some period of time. Asynchronous replication will lead to eventual consistency, since there is a lag between write confirmation and change propagation.

While the majority of the NoSQL systems associated with the eventual consistency, many of them provide some level of configuration to set up the tradeoff between performance and consistency level.

- One of the NoSQL system considered in this paper, MongoDB can achieve strong consistency by using 2 parameters: first set to read only from the master, meaning that only one node will be accessed for read and “read scalability” will be removed. Another way is to set “write concern” parameter to “replica acknowledged", which ensures that write is successfully completed on all the replicas. These techniques actually force the data store to the synchronous replication and therefore degrade performance.

- Other two NoSQL databases: Project Voldemort and Cassandra use consistent hashing and masterless replication, those systems also use a quorum in the consistency. Quorum means a minimal number of replicas that must respond to a request for it to be considered as successful. For read operation they use "read quorum" and for write operations – it's a separated "write quorum". If the following statement is true: $R + W > RF$, where R is the "read quorum", W is the "write quorum" and RF is the Replication factor (RF was explained in the "replication" chapter above), then we end up with a strong consistency data store.
- One of the two NewSQL systems – VoltDB is strongly consistent, fully transactional DB whereas the other one NuoDB uses asynchronous replication and therefore can be defined as eventually consistent data store.

Table 6 summarizes partitioning and consistency types of all the NoSQL and NewSQL systems surveyed in this final paper.

NoSQL/NewSQL databases		*Partitioning*	*Consistency*
Key Value Store	*Voldemort*	*Consistent hashing*	*Configurable, based on quorum read and write requests.*
Document Store	*MongoDB*	*Range partitioning based on a shard key (one or more fields that exist in every document in the collection). In addition, hashed shard keys can be used to partition data.*	*Configurable.* *Two methods to achieve strong consistency:* • *set connection to read only from primary* • *set write concern parameter to "Replica Acknowledged"*
Column Family Store	*Cassandra*	*Consistent hashing and range partitioning (known as order preserving partitioning in Cassandra terminology) is not recommended due to the possibility of hot spots and load balancing issues.*	*Configurable, based on quorum read and write requests.*
Graph Database	*Neo4j*	*No partitioning (cache sharding only).*	*Eventual consistency.*
NewSQL	*VoltDB*	*Consistent hashing. Users define whether stored procedures should run on a single server or on all servers*	*Strong consistency.*
	NuoDB	*No partition. The underlying key-value store can partition the data, but it is not visible by the user*	*Strong consistency.*

Table 6: The NoSQL/ NewSQL partitioning and consistency type

4.8. Security

Security is one of the most important aspects of modern data stores. Since different types of sensitive or personal data can be stored in NoSQL and NewSQL, security has become a major concern for a companies that are going to choose what technology to adopt. In this subsection, we're going to survey the data stores and compare them with regards to the following:

- Authentication
- Authorization
- Auditing
- Encryption

In the article “Security Issues in NoSQL Databases” [8] from 2011 the researchers from Ben-Gurion University surveyed two of the systems that we relate to in this paper: MongoDB and Cassandra.

They found various issues, such as lack of encryption in communication and lack of auditing in Cassandra. Nevertheless some of those issues were addressed in the most recent versions of those data stores and those security aspects no longer provide a cause for concern.

In Tables 7 and 8 we summarize the AAA (Authorization, Authentication and Auditing) security features and the supported encryption correspondingly.

NoSQL/NewSQL databases		*Authentication*	*Authorization*	*Auditing*
Key Value Store	*Voldemort*	*No*	*No*	*No*
Document Store	*MongoDB*	*Yes, store credentials in a system collection. REST interface does not support authentication. Enterprise Edition supports also Kerberos*	*Yes, permissions include read, read/write, dbAdmin and userAdmin. Granularity of collections*	*No*
Column Family Store	*Cassandra*	*Yes, store credentials in a system table. Possible to provide pluggable implementations*	*Yes, similar to the SQL GRANT/ REVOKE approach. Possible to provide pluggable implementations*	*Enterprise Edition only. Based on log4j framework. Logging categories include ADMIN, ALL, AUTH, DML, DDL, DCL, and QUERY. Possible to disable logging for specific key spaces*
Graph Database	*Neo4j*	*No, however developers can create a Security Rule and register with the server*	*No*	*No*
NewSQL	*VoltDB*	*Yes, users are defined in a deployment file that needs to be copied to each node*	*Yes, roles are defined at the schema level, and each stored procedure defined which roles are allowed to execute it*	*Yes, logging categories, include connections, SQL statements, snapshots, exports, authentication/ authorization, and others*
	NuoDB	*Yes*	*Yes, SQL-like*	*Yes, logging categories, include SQL statements, security events, general statistics and others*

Table 7: The NoSQL/ NewSQL security: AAA

4.9. Encryption

This refers to mechanisms that encrypt data so that it will not be readable by unauthorized parties. A complete encryption solution should be present in at least three different levels:

- Data at rest
- Client to Server
- Server to Server

NoSQL/NewSQL databases		***Encryption support***		
		Data at rest	***Client / Server***	***Server/ Server***
Key Value Store	*Voldemort*	*Possibly if BerkleyDB is used as a storage engine*	*No*	*No*
Document Store	*MongoDB*	*No, however a third party partner (Gazzang) provides an encryption plug-in*	*Yes – SSL based*	*Yes*
Column Family Store	*Cassandra*	*Enterprise Edition only. Commit log is not encrypted*	*Yes – SSL based*	*Yes, configurable: all server-to-server communication , only between datacenters or between servers in the same rack*
Graph Database	*Neo4j*	*No*	*Yes – SSL based*	*No*
NewSQL	*VoltDB*	*No*	*No*	*No*
	NuoDB	*Native store does not support it. Theoretically, it could use a pluggable store that supports it*	*Yes*	*Yes*

Table 8: The NoSQL/ NewSQL security: encryption

The main observation regarding security is that NoSQL solutions are still not as mature as those included in traditional RDBMS systems.

Another interesting finding is that MongoDB and Cassandra offer additional security functionalities in their enterprise editions, because security is a particularly important concern for large companies. For example, data-at-rest encryption and auditing are available only in Cassandra Enterprise Edition.

5. Applications and scenario suitability

Each one of the articles describing various data stores [18] [19] [21] [24] [25] provides some sort of recommendation what use-case every system is most suitable for.

Jablonsky et al also provided some recommendations for the databases they were focusing on in their article [5]. In this paper we'll go over each one of the data store categories as defined in the "Taxonomy" chapter and try to evaluate what scenarios would benefit from particular features of each one of the data bases types.

5.1.1. Key-Value stores

Due to their data model key value stores are useful for simple operations, which are based on key attributes only. In order to speed up a user specific rendered webpage, parts of this page can be calculated before and served quickly and easily out of the store by user IDs when needed. There are also some other use cases for key-value based data stores, for example KV stores are widely used for the following application types:

- Caching
    - Since most key value stores hold their dataset in memory, they are oftentimes used for caching of more time intensive SQL queries.
- Queuing
    - Some K/V stores supports lists, sets, queues and more.
- Distributing information / tasks
- Keeping live information
    - Applications which need to keep a state can use K/V stores easily.
- MySQL-like Applications, dynamic queries, many data updates

5.1.2. Column Family

Column family stores only store a key value pair in one row, if a dataset needs it. It's completely different approach from the traditional relational databases that would store a null values in each column a dataset has no value for. Therefore column family are more suitable in domains with huge amounts of data with varying numbers of attributes. Thus the main use cases for CF data stores would be:

- Keeping unstructured, non-volatile information
  - If a large collection of attributes and values needs to be kept for long periods of time, column-based data stores come in extremely handy.
- Scaling
  - Column based data stores are highly scalable by nature. They can handle an awful amount of information.
- Search engines, log data analytics.

5.1.3. Document stores

Document stores offer multi attribute lookups on records which may have complete different kinds of key value pairs, that's one of the reasons those systems are very convenient in data integration and schema migration tasks

Those data stores usually also support nested data structures, links and JSON documents besides that document stores such as MongoDB are considered to be very easy to maintain and are therefore suitable for flexible web applications. Most popular use cases are:

- Nested information use cases

    - Document-based data stores allow you to work with deeply nested, complex data structures
- Real time analytics
- Logging
- Storage layer of small and flexible websites like blogs
- Different JavaScript friendly applications
    - Hence one of the most critical functionalities of document-based data stores are the way they interface with applications: using JS-friendly JSON.

Key value stores, document stores and column family stores have in common, that they do store deformalized data in order to gain advantages in distribution. As a result relationships must be handled completely in the application logic. The famous Social Networking domain problem - friend-of-a-friend and similar recommendation queries would lead to many queries, performance penalties and complex code in the application layer if one of these databases was used for such use cases [5]. The remaining NoSQL data store category – Graph databases are much more appropriate solution in this domain.

5.1.4. Graph Databases

Graph databases, such as Neo4j are suitable in scenarios as:

- Pattern matching
- Dependency analysis
- Recommendation systems
- Social networking applications
- Solving path finding problems in navigation systems

Some graph data bases and Neo4j [19] in particular are fully ACID compliant. Nevertheless they are not as sufficient as other NoSQL types reviewed above in scenarios other than handling graphs and relationships

5.1.5. NewSQL

NewSQL databases are providing ACID and supporting transactions, therefore the natural match for NewSQL would be applications that involve OLTP (on line transaction processing). For example, according to Prof. Stonebreaker [49] NewSQL would fit into the following fields:

- Web-based applications such as
    - Multi-player games,
    - Social networking sites, and
    - Online gambling networks.
- Smartphone applications that use the phone as a geographic sensor and provide location-based services.
- Machine-to-machine (M2M) applications.

6. Survey of articles discussing performance

In this chapter we provide a short survey on a few papers dealing with NoSQL/ NewSQL database performance comparison.

6.1. "Yahoo! Cloud Serving Benchmark" [2]

First we must provide a brief description of the YCSB benchmarking suite used in the experiments described in all the articles reviewed in this part of the paper. YCSB acronym stands for Yahoo! Cloud Serving Benchmark. YCSB system has been developed by a team of developers in 2010 and its main goal was to come up with an extensible and generic framework for the evaluation of key-value stores [2].

YCSB allows to generate synthetic workloads which are defined as a configurable distribution of CRUD (create, read, update and delete) operations on a set of records. Records have a predefined number of fields and are logically indexed by a key. This generic data model can easily be mapped to a specific key-value or column-based data model. Figure 13 describes YCSB client architecture as it was given in the original article by Brian Cooper [2]:

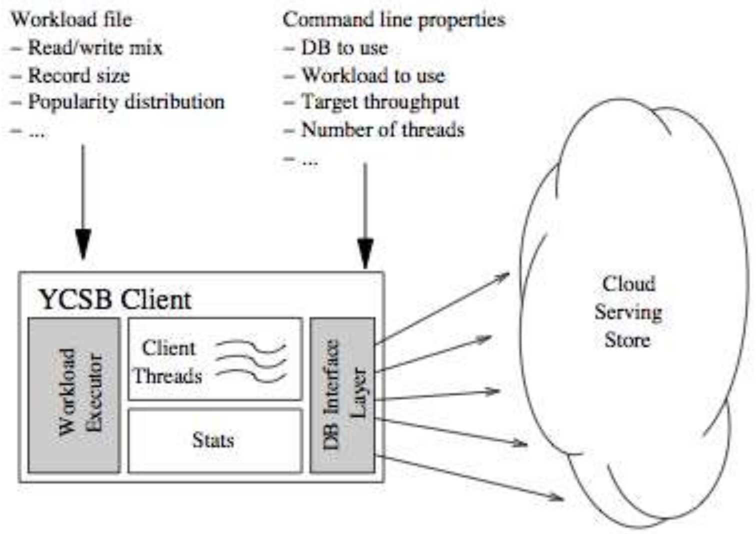


Figure 13: YCSB high-level design [2]

The reason for YCSB's popularity is that it comprises a data generator, a workload generator, as well as drivers for several key-value stores, some of which are also used in this evaluation.

6.2. "A comparison between several NoSQL databases with comments and notes" [51]

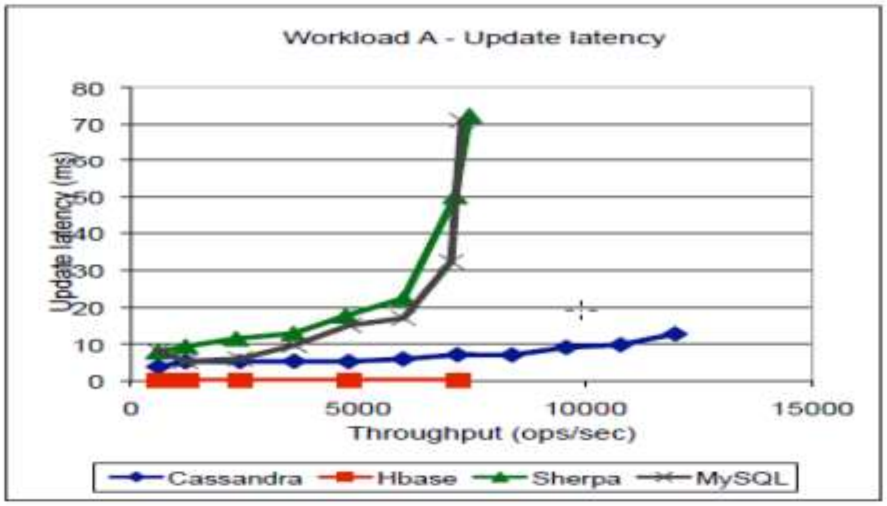
The authors of that article are trying to comment on the various NoSQL (Not only Structured Query Language) systems and to make a comparison between them. Initially they provide a definition for the term NoSQL, ACID and BASE and indicate some of the differences between NoSQL approach and the traditional RDBMS. Further they are trying to determine "what to compare" and come up with the taxonomy, which is similar to the one used in this paper, but slightly different:

- Wide Column
- Document Store
- Tuple Store
- Eventually Consistent Key Value Store
- Graph Databases

Further they propose two approaches for the comparison - one is A QUALITATIVE and a QUANTITATIVE.

- QUALITATIVE evaluation is done by comparing what features are available for the NoSQL databases taken into account.
- For QUANTITATIVE evaluation criteria they used two different sets, one related to size and one related to performance.

The performance measurements listed in the last few paragraphs and included in the figures is obtained from [2] which is describing a laboratory based benchmark which uses YCSB (Yahoo! Cloud Serving Benchmark) as a measurement tool. The benchmark was run on 120 million records of small size (1kB), 6 node, and 0.12 TB equivalent installations of the three products. Figures 14 and 15 show the results of the experiments.



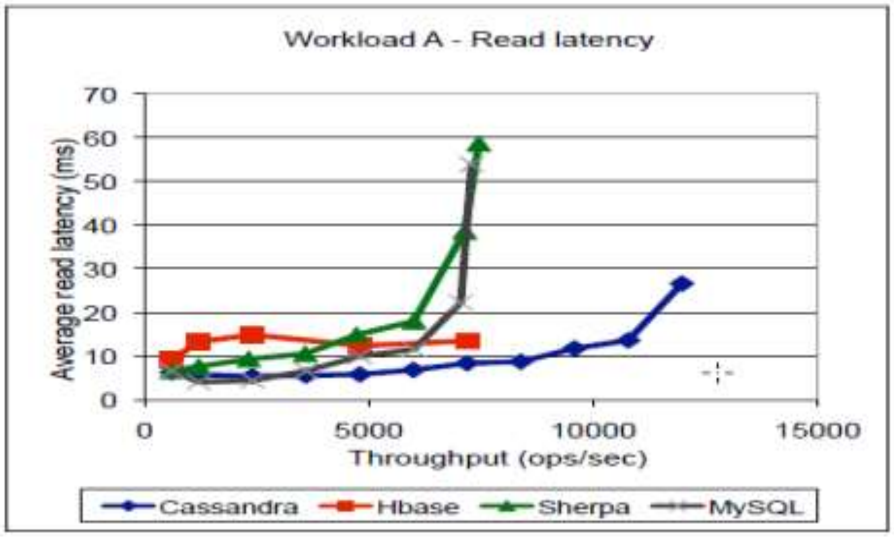
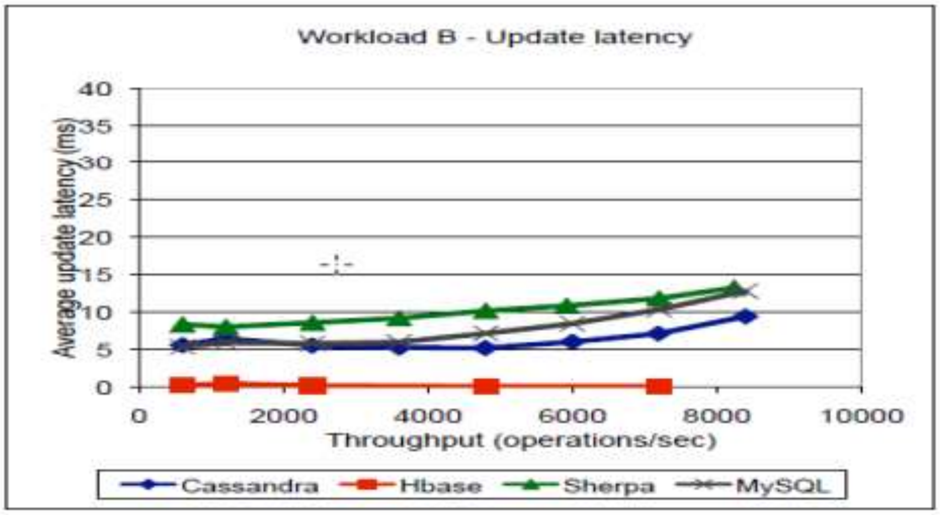


Figure 14: workload A latencies



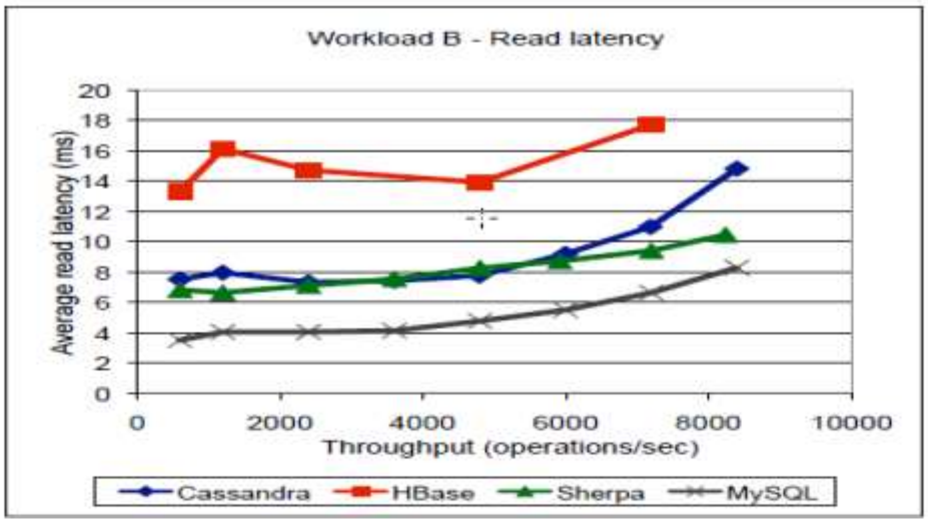


Figure 15: workload B latencies

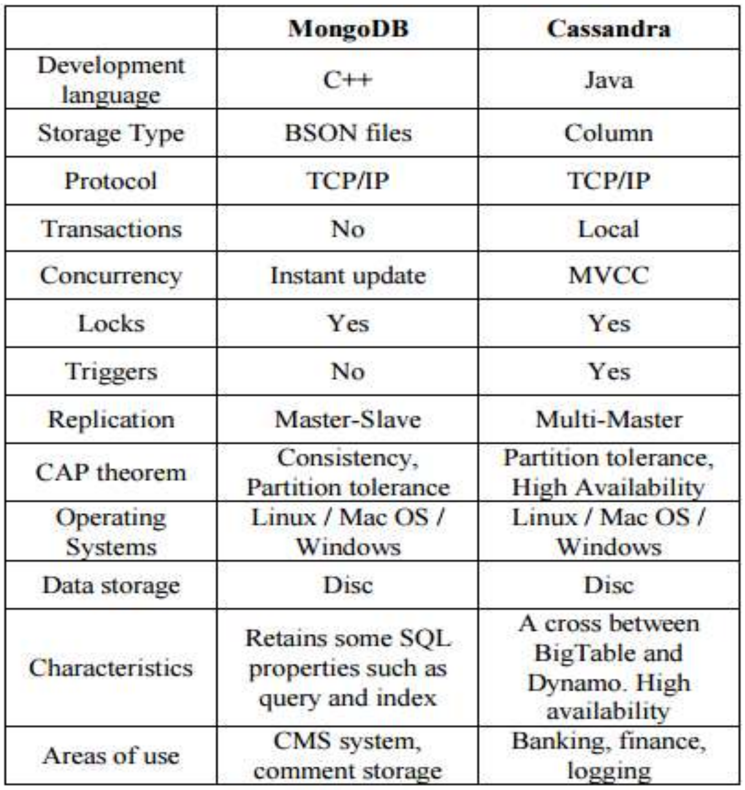
Finally the conclusion that the authors come to is that even though the SQL and the NoSQL databases have some shared features, but they are not similar in their behavior. Therefore they cannot be used interchangeably for solving any type of problem and for a particular given problem one shall consider both types and choose between them.

6.3. ”NoSQL Databases: MongoDB vs Cassandra” [55]

This article is mainly focusing on the comparison two of the most popular NoSQL databases: MongoDB and Cassandra. In the related work chapter the authors relate to the same famous article describing YCCB in action [2] and they point out that 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 [2]. In their paper they are prioritizing different execution parameters.

Further the paper provides a definition for the terms NoSQL, BASE and define the taxonomy which is exactly the same as we use in this final paper.

The article also gives a brief description for MongoDB and Cassandra and summarizes both data stores features as follows in Table 9:



	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

Table 9: MongoDB and Cassandra features

The paper describes the experiments that were conducted using YCCB (Yahoo! Cloud Serving Benchmark). YCSB has a client that consists of two parts: workload generator and the set of scenarios. The scenarios, known as workloads, are combinations of read, write and update operations performed on randomly chosen records. The workloads that were used in the experiments described in the article are the following

- 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.

Eventually after executing multiple experiments using YCCB and the workload mentioned above the authors come to the following conclusions:

- With the increase of data size, MongoDB started to decrease performance

- Cassandra just got faster while working with an increase of data
- Cassandra is faster than MongoDB, providing lower execution time independently of database size used in our evaluation.

Thus, as overall analysis turns out that Cassandra show the best results for almost all scenarios.

6.4. "Solving big data challenges for enterprise application performance management" [56]

This article provides a comprehensive performance evaluation of six modern (open-source) data stores: Apache Cassandra, Apache HBase, Project Voldemort, Redis, VoltDB, and a MySQL. They evaluated these systems with data and workloads that can be found in application performance monitoring, as well as, on-line advertisement, power monitoring, and many other use cases.

Initially the paper provides a background on a large scale enterprise systems and APM. Large scale systems are heterogeneous and have many interdependencies which makes their administration a very complex task.

Application Performance Management (APM) tools, provide a more sophisticated view on the monitored system, their purpose is to give administrators an on-line view of the system health. These tools instrument the applications to retrieve information about the response times, as well as about failure rates, resource utilization, etc.

The article then point out that APM has similar requirements to current Web-based information systems such as weaker consistency requirements, geographical distribution, and asynchronous processing. Because of this similarity of APM storage requirements to the requirements of Web information system applications, obvious candidates for new APM storage systems are key-value stores and their derivatives (NoSQL and NewSQL).

In the next chapter the authors provide a more thorough description of the use case and big data challenge, as following: in a highly distributed system, it is difficult to determine the root cause of performance degradations especially since it is often not tied to a single component, but to a specific interaction of components. System components are heterogeneous and there is no unified code base and often access to the entire source code is not possible. Thus, an in depth analysis of the components or the integration of a profiling infrastructure is not possible. To overcome this challenges, application performance management systems (APM) have been developed.

In general the APM data is in general relatively simple. It usually consists of a metric name, a value, and a time stamp. With regards to the storage the queries can be distinguished into two major types:

- Single value lookups to retrieve the most current value and
- Small scans for retrieving system health information and for computing aggregates over time windows.

Based on these properties they define a benchmark. Similar to other articles surveyed above in the chapter the benchmarking system used in the experiment was YCSB benchmark. For the experiment they chose each two of the following classes according to the classification presented by Cartell in [4]:

- Key-value stores: Project Voldemort and Redis
- Extensible record stores: HBase and Cassandra
- Scalable relational stores: MySQL Cluster and VoltDB

In Table 10 we've summarized all the various workloads presented in the paper.

Workload Type	Read	Write	Scan
Workload R	95%	5%	0%
Workload RW	50%	50%	0%
Workload W	1%	99%	0%
Workload RS	47.5%	5%	47.5%
Workload RSW	25%	50%	25%

Table 10: workloads details

The graph in Figure 16 below shows some of the results that were obtained using those workloads.

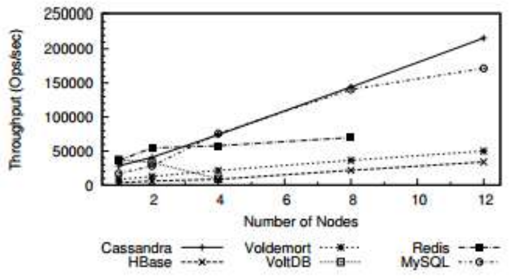


Figure 6: Throughput for Workload RW

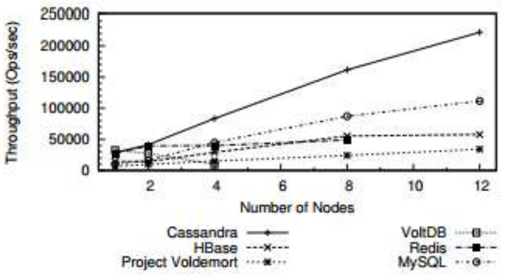


Figure 9: Throughput for Workload W

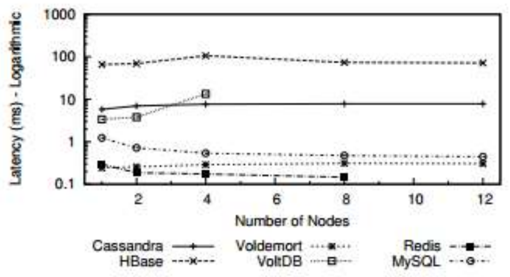


Figure 7: Read latency for Workload RW

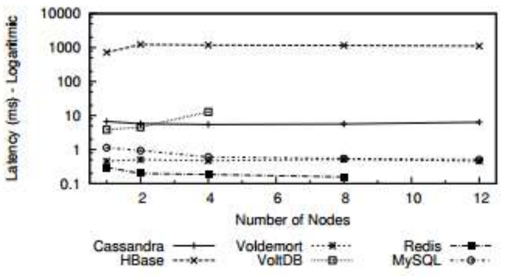


Figure 10: Read latency for Workload W

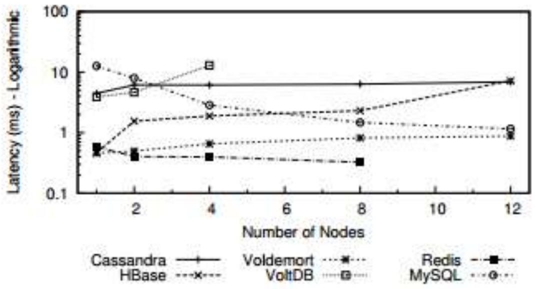


Figure 11: Write latency for Workload W

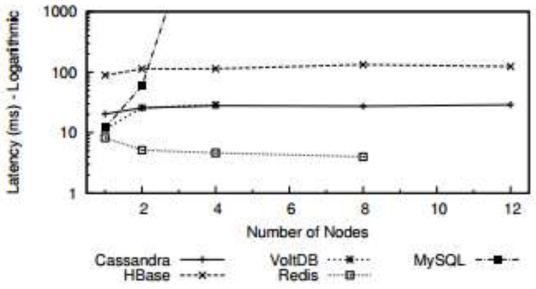


Figure 13: Scan latency for Workload RS

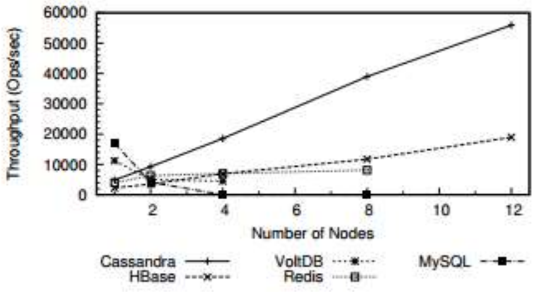


Figure 12: Throughput for Workload RS

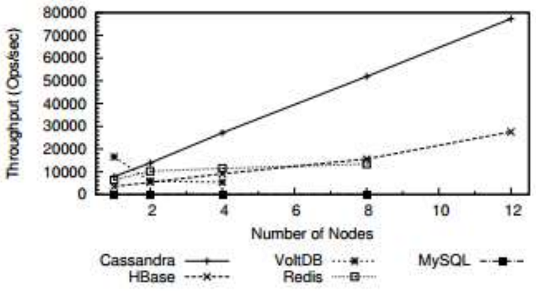


Figure 14: Throughput for Workload RSW

Figure 16: latency and throughput for various workloads

Finally, the paper reports on details of their experiences with these systems from an industry perspective. They point out that Cassandra's setup was relatively easy, since there are quick-start manuals available at the official website. Another observations was a linear scalability for Cassandra, HBase, and Project Voldemort in most of the tests. Cassandra's throughput dominated in all the tests, however, its latency was in all tests peculiarly high.

The configuration of Project Voldemort was easy for the most part. However, in contrast to the other systems, we had to create a separate configuration file for each node. Regarding the performance - Project Voldemort exhibits a stable latency that is much lower than Cassandra's latency.

The VoltDB configuration was mostly inspired by the VoltDB community documentation that suggested the client implementation and configuration of the store [37]. The

VoltDB developers benchmarked the speed of VoltDB vs. Cassandra themselves with a similar configuration, but only up to 3 nodes – I've described that experiment in the seminar report (Appendix A). Performance wise the observation is the following: VoltDB exhibited a high throughput for a single node, however the multi-node setup did not scale well.

It is worth to mention that the results presented in the article are valid also for related use cases, such as on-line advertisement marketing, click stream storage, and power monitoring. Unlike previous work, this paper focused on the maximum throughput that can be achieved by the systems. One of the future plans mentioned in the article is to determine the impact of replication and compression on the throughput in the APM use case.

7. Final Project: NewSQL/ NoSQL in Social Networking

In this chapter we provide a description of the Final Project (see Appendices B and C).

7.1. Background

As part of the Final Project implementation we have performed Practical evaluation of NewSQL/ NoSQL systems using advanced benchmarking system called BG. BG is a benchmarking system that rates data store by processing interactive social networking actions [9]. It has been inspired by prior benchmarks that evaluate cloud services such as YCSB [2] [62] and YCSB++ [63], e-commerce sites, and object-oriented and transaction processing systems [61].

We used Cassandra as an example of NoSQL data store. This data store can be considered as a definitive representative of NoSQL movement since it's one of the most popular NoSQL databases [34] and it's built using various concepts of two major initial NoSQL projects: Dynamo [1] and BigTable [15].

We have chosen NuoDB as NewSQL data store, since it's an independent project which became widely used in the Cloud environment. Internally it has implemented various techniques that were touched base above, such as MVCC [47] and asynchronous replication.

We used BG because of the conceptual data model (Appendix B) which is more complex than that of the YCSB and YCSB++. Besides the schema, BG main contributions are two folds. First, it emphasizes interactive social actions that retrieve a small amount of data. Second, it promotes the amount of unpredictable data produced by a solution as a first class metric for comparing different data stores with one another [59].

BG populates data base with a social graph consisting of a fixed number of members, friends per member, and resources per member. It consists of social networking actions such as extend a friendship invitation to a member and view a member profile. See Table 11 for a list of the BG actions considered in this study.

	f	g+	🐦	in	You Tube
View Profile (VP)	✓	✓	✓	✓	✓
List Friends (LF)	✓	✓	✓	✓	✓
View Friend Requests (VFR)	✓	⊖	⊖	✓	⊖
Invite Friend (IF)	✓	*Add to Circle*	*Follow*	✓	*Subscribe*
Accept Friend Request (AFR)	✓	⊖	⊖	✓	⊖
Reject Friend Request (RFR)	✓	⊖	⊖	✓	⊖
Thaw Friendship (TF)	✓	*Remove from Circle*	*Unfollow*	✓	*Unsubscribe*
View Top-K Resources (VTR)	✓	✓	✓	✓	✓
View Comments on a Resource (VCR)	✓	✓	✓	✓	✓
Post Comment on a Resource (PCR)	✓	✓	*Reply to a Tweet*	*Send a Recommendation*	✓
Delete Comment on a Resource (DCR)	✓	✓	*Delete Reply to a Tweet*	*Withdraw Recommendation*	✓

Table 11: Social Actions

We use BG to establish two important metrics:

1. Database population time, i.e. measure the time to fully load BG database including 10000 members with 100 friends per user and 100 resources per user. Load is done using 10 loading threads.

2. Social Action Rating (SoAR): this rating compute the number of concurrent actions performed by a system when a fixed percentage of requests (say 98%) observe a latency equal to or lower than a pre-specified threshold (say 100 msec) with the amount of unpredictable data less than a fixed threshold (say 0.01%) for some fixed duration of time (say 10 minutes). The values in the parenthesis are inputs to BG. BG's output is the SoAR rating of its target data store [59].

7.2. Implementation

In the following chapters we describe briefly our implementation of BG data model schema in Cassandra and NuoDB data stores. A detailed description is given in the final project report.

7.2.1. Cassandra Schema Description

- Data is stored in the following 3 Column Families:
  - Members
  - Resource
  - Manipulation
- Profile and thumbnail images for a user are stored as Blobs (Binary Large OBjectS) columns within Members column family.
- PengingRequests and ConfirmedFriendship are stored as a collection-type columns within Members column family. Those columns contain sets of user ids of the corresponding users.
- We decided to implement Manipulation as a separate Column Family and not as a collection-type column within Resources, because of the following 2 reasons:
  - The length of the collection-type field is limited (CQL3 limitation is 64K)

    - In Cassandra 1.2 they do not support non-primitive type as a member of collection column. According to Datastax: it might be possible to support such functionality in the future if that is deemed useful, but it will require additional work [20]
- Primary Keys
    - "Members" primary key is userid
    - "Manipulation" primary key is manipulationid (mid)
    - "Resource" has a compound primary key of (walluserid, rid)
- Indexes
    - The "Resource" column family is also indexed on creatorid
    - The "Manipulation" column family is also indexed on resourceid (rid)

7.2.2. NuoDB Schema Description

- Data is stored in the following tables:
    - users
    - friendship
    - resources
    - manipulation
- Friendship is stored in one table, "friendship", as two records for every confirmed friendship.
- Aggregates are calculated by issuing aggregate queries.
- Profile and thumbnail images for a user are stored in the file system.
- Indexes

- The "users" table is indexed on userid.
- The "friendship" table is index on friend1 and friend2.
- The "resources" table is indexed on resourceid and walluserid (the userid for the wall, where the resource is posted on).
- The "manipulation" table is indexed on manipulationid and resourceid.

7.2.3. Social Actions

We've noticed that the design and implementation of the required social actions for NewSQL database as NuoDB which is almost fully SQL-compliant [29] usually requires much less effort than of implementation of corresponding NoSQL client. Table summarizing social actions implementation can be found in the final project report (Appendix B).

Experiment setup and execution details including screen captures can be seen in the attachment for final project report (Appendices B and C).

That's an interesting outcome of this investigation is the observation that NuoDB operates much faster during the initial Social graph data load (data population).

This is an obvious advantage of NoSQL databases related to the fact that there are less relation between column families than in tables within relational schema.

7.3. Results

More detailed explanation on the experiment setup (database installation/ access), configuration parameters (SLA, RF) as well as the table providing parameters for various workloads (VeryLowUpdate, LowUpdate and High Update) can be found in the final project report (Appendix B).

Figure 17 displays data base loading time for the configurations described above.

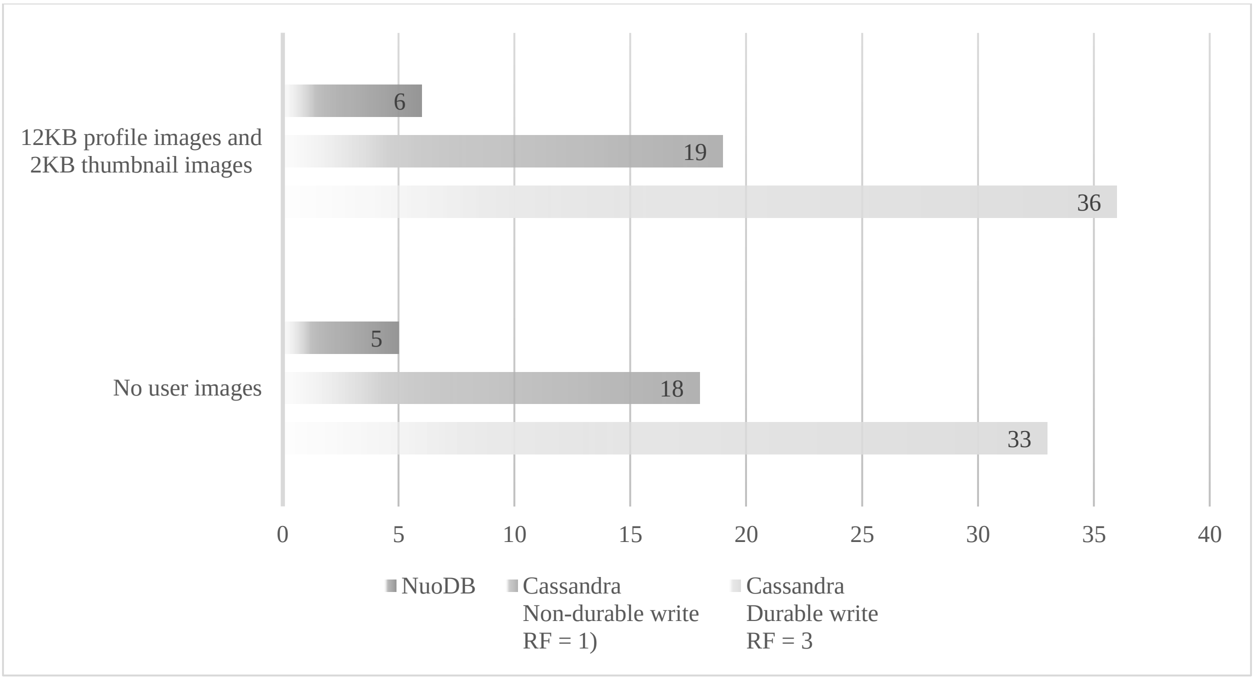


Figure 17: Database loading

Relational databases in general are more suitable for data loading when relations are known up front and reflected in the schema accordingly. Besides that we have noticed that when Cassandra is configured with “non-durable write” (replication factor less than the number of nodes), then the loading performance is much higher comparing to “durable write” (replication factor equals to the number of nodes).

On the other hand Cassandra has shown a better performance during the experiment emulating interactive social actions.

Figure 18 shows experimental results obtained for the configurations described above.

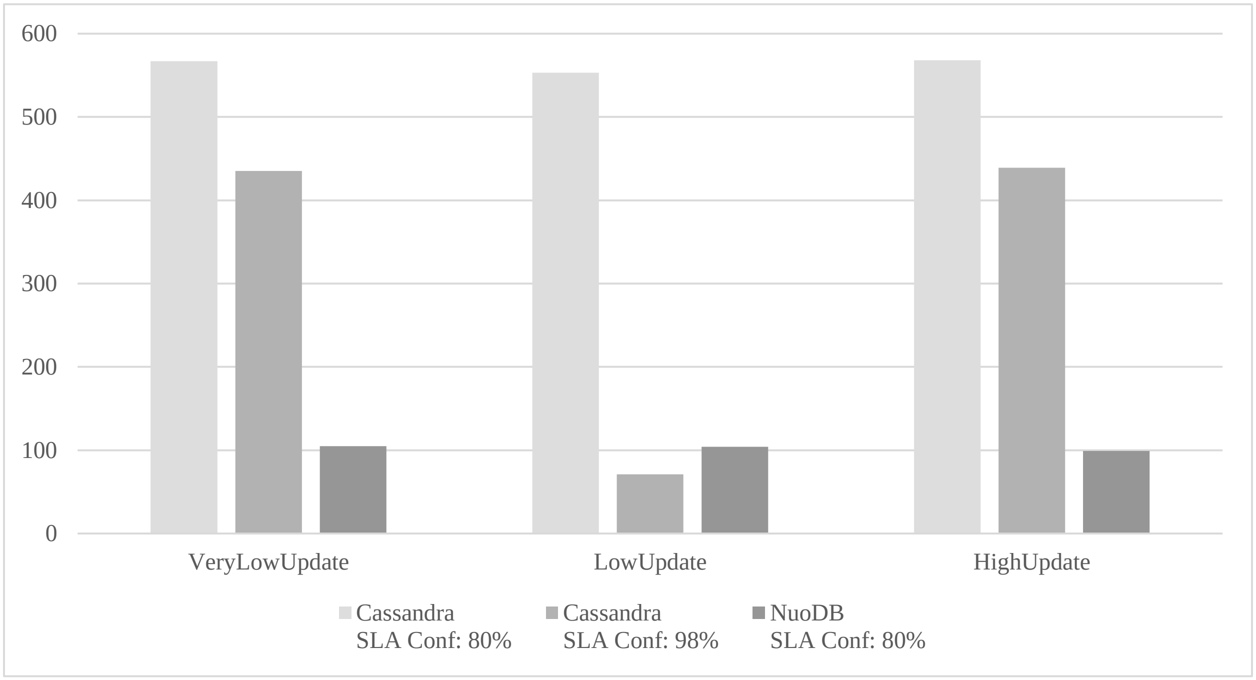


Figure 18: SoAR rating of databases using different workloads

It can be explained by decentralization features as well as lack of normalization [18] which makes NoSQL database easily scalable and showing better results in distributed application when multiple clients perform concurrent actions. The latest is a result of special architecture characteristics contributing to Cassandra’s write: eventual consistency, lack of locking mechanism and usage of in memory structures called memTable. Same theoretical observation has been made by Grolinger et al. in their article “Data management in cloud environment” when they classified Cassandra use case as “heavy write load environment” [7].

8. Conclusions

In the recent past, cloud computing along with distributed web and mobile applications dictated a new storage and processing requirements for data bases. Those demands were more challenging than the traditional database techniques can handle and as a result new architectures started to emerge. Two major categories that define themselves as alternatives to the traditional SQL databases are NoSQL and NewSQL stores.

In this paper, we did a detailed comparative study of NoSQL and NewSQL data stores on several parameters, both technical and non-technical. We performed comparison based on data model, querying capabilities, concurrency control, replication, scalability, partitioning strategies, consistency models and security features. Furthermore we have discussed use-cases suitability and possible applications in which each type of NoSQL/ NewSQL databases can be used.

Additionally, we have also surveyed a number of articles dealing with performance analysis and quantitative comparison of various data stores based on the results obtained by using YCSB benchmarking system.

Finally we have presented the experiments that we conducted as a part of the Final Project implementation. The project was dealing with an evaluation of performance of NoSQL/ NewSQL data stores in the environment emulating social actions network (BG benchmark suite). The practical results obtained during the Final Project were comprehensively analyzed in this paper.

In summary, there is number of NoSQL/NewSQL databases types with differing capabilities. There are a variety of existing and newly-emerging applications that can benefit from principles and techniques provided by those systems. The comparison among the most popular NoSQL/NewSQL data stores along with the description of the possible use cases provided in this paper may provide additional assistance for practitioners on choosing the best data storage solution for their application needs.

References

[1] DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.

[2] Cooper, Brian F., “Yahoo! Cloud Serving Benchmark”, http://research.yahoo.com/files/ycsb-v4.pdf, (unpublished)

[3] Laney, Douglas. "3D Data Management: Controlling Data Volume, Velocity and Variety" . Gartner. Retrieved 6 February 2001.

[4] R. Cartell. Scalable sql and nosql data stores. SIGMOD Record, 39(4):12–27, 2010.

[5] Robin Hecht. NoSQL evaluation: A use case oriented survey. In Stefan Jablonski, editor, Proceedings of the International Conference on Cloud and Service Computing, pages 336–341, 2011.

[6] Brewer, Eric A.: Towards Robust Distributed Systems. Portland, Oregon, July 2000. – Keynote at the ACM Symposium on Principles of Distributed Computing (PODC) on 2000-07-19. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[7] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, Miriam AM Capretz; “Data management in cloud environments: NoSQL and NewSQL data stores”; Journal of Cloud Computing: Advances, Systems and Applications 2013.

[8] Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes, Jenny Abramov, "Security Issues in NoSQL Databases," Proceedings of Trustcom, pp.541-547, 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011

[9] http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[10] Eric Brewer, “CAP twelve years later: How the "rules" have changed”, IEEE Explore, Volume 45, Issue 2 (2012), pg. 23-29

[11] Stonebraker, Michael ; Madden, Samuel ; Abadi, Daniel J. ; Harizopoulos, Stavros ; Hachem, Nabil ; Helland, Pat: The end of an architectural era: (it’s time for a complete rewrite). In: VLDB ’07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, p. 1150–1160

[12] Michael Stonebraker: SQL databases v. NoSQL databases. Commun. ACM 53(4): 10-11 (2010)

[13] Michael Stonebraker, NewSQL: An Alternative To NoSQL And Old SQL For New OLTP Apps, CACM Blog, 2011

[14] Katsov, Ilya (1 March 2012). "NoSQL Data Modeling Techniques". Ilya Katsov. Retrieved 8 May 2014.https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

[15] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.

[16] Hbase http://wiki.apache.org/hadoop/Hbase

[17] Standard column family https://en.wikipedia.org/wiki/Standard_column_family

[18] A. Lakshman and P. Malik. Cassandra -- A Decentralized Structured Storage System. In Proc. of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS '2009), Big Sky, MT, October 2009.

[19] Developers, Neo4J. "Neo4J." Graph NoSQL Database http://neo4j.com/

[20] http://www.datastax.com/

[21] Project Voldemort: A distributed database. http://project-voldemort.com/.

[22] Gao, Roshan Sumbaly Jay Kreps Lei, and Alex Feinberg Chinmay Soman Sam Shah. "Serving Large-scale Batch Computed Data with Project Voldemort."

[23] CouchBase Deveveloper Guide. http://docs.couchbase.com/developer/dev-guide-3.0/compare-docs-vs-relational.html

[24] MongoDB. http://www.mongodb.org/

[25] Cassandra", http://cassandra.apache.org,

[26] eBay Tech Blog: http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/

[27] Neo4j Blog http://neo4j.com/blog/the-top-10-ways-to-get-to-know-neo4j/

[28] Michael Stonebraker, NewSQL: An Alternative To NoSQL And Old SQL For New OLTP Apps, CACM Blog, 2011

[29] NuoDB Technical Whitepaper http://go.nuodb.com/white-paper.html

[30] Amazon Web Services http://aws.amazon.com/

[31] Stack Overflow http://stackoverflow.com/

[32] Database Administrators Stack Exchange http://dba.stackexchange.com/

[33] LinkedIn: https://www.linkedin.com/

[34] DB-Engines Ranking http://db-engines.com/en/ranking

[35] Ho, Ricky: NOSQL Patterns. November 2009. – Blog post of 2009-11-15. http://horicky.blogspot.com/2009/11/nosql-patterns.html

[36] 451 group blog https://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/

[37] VoltDB Technical Overview and Whitepapers www.voldb.com/resources/datasheets

[38] Stonebraker, Michael, and Ariel Weisberg. "The VoltDB main memory DBMS. "IEEE Data Eng. Bull 36.2 (2013).

[39] Stonebraker, Michael: The “NoSQL” Discussion has Nothing to Do With SQL. November 2009. – Blog post of 2009-11-04. http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-withsql/fulltext

[40] Michael Stonebraker: SQL databases vs. NoSQL databases. Commun. ACM 53(4): 10-11 (2010)

[41] NewSQL: what’s this? http://labs.sogeti.com/newsql-whats/

[42] No to SQL? Anti-database movement gains steam. Computerworld. June 2009. – http://www.computerworld.com/article/2526317/database-administration/no-to-sql--anti-database-movement-gains-steam.html

[43] https://en.wikipedia.org/wiki/Representational_state_transfer

[44] SPARQL - RDF query language https://en.wikipedia.org/wiki/SPARQL

[45] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008.

[46] Apache Hadoop https://en.wikipedia.org/wiki/Apache_Hadoop

[47] Multiversion concurrency control

https://en.wikipedia.org/wiki/Multiversion_concurrency_control

[48] Vector Clock https://en.wikipedia.org/wiki/Vector_clock

[49] https://en.wikipedia.org/wiki/Michael_Stonebraker

[50] https://www.digitalocean.com/community/tutorials/

a-comparison-of-nosql-database-management-systems-and-models

[51] 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.

[52] Amazon http://www.amazon.com/

[53] eBay http://www.ebay.com/

[54] High-performance computing https://en.wikipedia.org/wiki/Supercomputer

[55] Abramova, Veronika, and Jorge Bernardino. "NoSQL databases: MongoDB vs cassandra." Proceedings of the International C* Conference on Computer Science and Software Engineering. ACM, 2013.

[56] Rabl, Tilmann, et al. "Solving big data challenges for enterprise application performance management." Proceedings of the VLDB Endowment 5.12 (2012): 1724-1735.

[57] http://hortonworks.com/blog/7-key-drivers-for-the-big-data-market/

[58] http://nosqltips.blogspot.com

[59] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. Proceedings of 2013 CIDR, January 2013.

[60] Soni Madhulatha T (2012) Graph partitioning advance clustering technique. Int J Computer Sci Eng Surv 3(1):91-104
10.5121/ijcses.2012.3109

[61] Transaction Processing Performance Council, "TPC benchmark C standard spec. 5.11," Feb 2010, http://www.tpc.org/tpcc/spec/tpc-c_v5-11.pdf.

[62] Brian F. Cooper , Adam Silberstein , Erwin Tam , Raghu Ramakrishnan ,Russell Sears, Benchmarking cloud serving systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing, June 10-11, 2010, Indianapolis, Indiana, USA

[63] Swapnil Patil , Milo Polte , Kai Ren , Wittawat Tantisiriroj , Lin Xiao , Julio López , Garth Gibson , Adam Fuchs , Billie Rinaldi, YCSB++: benchmarking and performance debugging advanced features in scalable table stores, Proceedings of the 2nd ACM Symposium on Cloud Computing, p.1-14, October 26-28, 2011, Cascais, Portugal

[64] Shard https://en.wikipedia.org/wiki/Shard_(database_architecture)

[65] Clark, BJ: NoSQL: If only it was that easy. August 2009. – Blog post of 2009-08-04. http://bjclark.me/2009/08/04/nosql-if-only-it-was-that-easy/

Appendices

Appendix A – Seminar report

Report for the seminar “Traditional SQL, NoSQL or NewSQL?” has been prepared within research-seminar course “Database Systems and Data Mining” (22953 – 2014a).

Appendix B - Final Project report

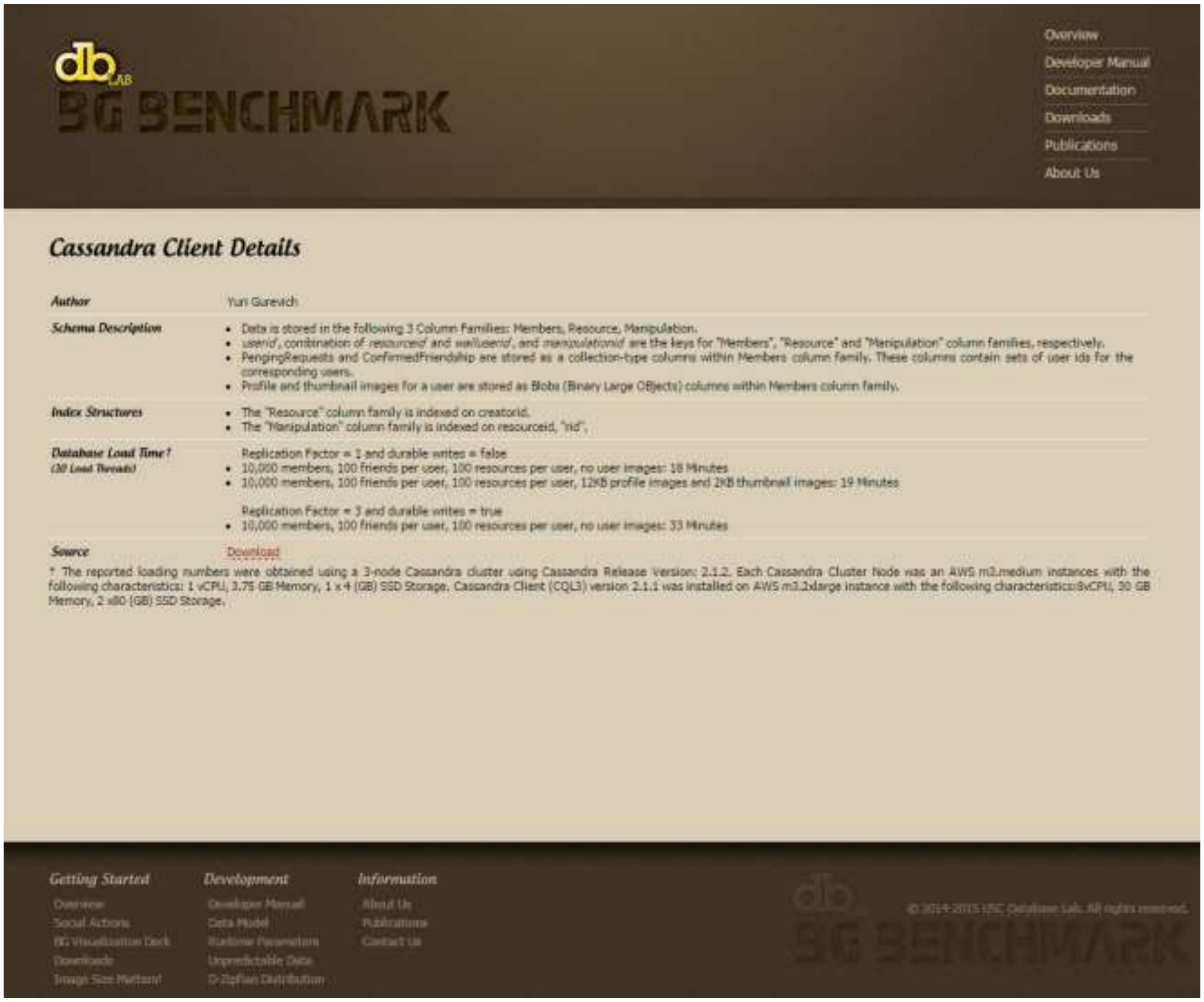
Report on the Final Project “Practical evaluation of NoSQL/NewSQL systems”, Prepared under the supervision of Prof. Ehud Gudes (February 2015)

Appendix C - Final Project implementation

Our Cassandra Client implementation has been evaluated by the creators of BG benchmark in the University of Southern California and they requested to post it on the official BG benchmark Web page in.

That is the URL with the description and download details for Cassandra Client that was implemented as a part of Final Project:

http://bgbenchmark.org/BG/CassandraClient.html



Cassandra Client Details

Author	Yuri Gurevich
Schema Description	• Data is stored in the following 3 Column Families: Members, Resource, Manipulation. • *userid*, combination of *resourceid* and *walluserid*, and *manipulationid* are the keys for "Members", "Resource" and "Manipulation" column families, respectively. • PengingRequests and ConfirmedFriendship are stored as a collection-type columns within Members column family. These columns contain sets of user ids for the corresponding users. • Profile and thumbnail images for a user are stored as Blobs (Binary Large OBjects) columns within Members column family.
Index Structures	• The "Resource" column family is indexed on creatorid. • The "Manipulation" column family is indexed on resourceid, "rid".
Database Load Time † (30 Load Threads)	Replication Factor = 1 and durable writes = false • 10,000 members, 100 friends per user, 100 resources per user, no user images: 18 Minutes • 10,000 members, 100 friends per user, 100 resources per user, 12KB profile images and 2KB thumbnail images: 19 Minutes Replication Factor = 3 and durable writes = true • 10,000 members, 100 friends per user, 100 resources per user, no user images: 33 Minutes
Source	Download

† The reported loading numbers were obtained using a 3-node Cassandra cluster using Cassandra Release Version: 2.1.2. Each Cassandra Cluster Node was an AWS m3.medium instances with the following characteristics: 1 vCPU, 3.75 GB Memory, 1 x 4 (GB) SSD Storage. Cassandra Client (CQL3) version 2.1.1 was installed on AWS m3.2xlarge instance with the following characteristics:8vCPU, 30 GB Memory, 2 x80 (GB) SSD Storage.