Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Storing and structuring big data with business intelligence in mind.

by

Fredrik Andersson

LIU-IDA/LITH-EX-A--15/050--SE

2015-09-15

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Final thesis

Storing and structuring big data with business intelligence in mind.

by

Fredrik Andersson

LIU-IDA/LITH-EX-A–15/050–SE

September 15, 2015

Supervisor: Dag Sonntag

Examiner: Jose Peña

Abstract

Sectra has a customer database with approximately 1600 customers across the world. In this system there exists not only medical information but also information about the environment which the system runs in, usage pattern and much more.

This report is about storing data received from log files into a suitable database. Sectra wants to be able to analyze this information so that they can make strategic decisions and get a better understanding of their customers' needs. The tested databases are MongoDB, Cassandra, and MySQL. The results shows that MySQL is not suitable for storing large amount of data with the current configuration. On the other hand, both MongoDB and Cassandra performed well with the growing amount of data.

Acknowledgements

I would like to thank Sectra Imtec for the opportunity to perform this project and a special thanks to Alexander Sjöberg for all the help and support during the project. I would also like to thank Dag Sonntag and Jose Peña at Linköping's University for all the help and feedback I received during the project.

Furthermore, I like to thank Jonas Björk at Spotify and Cecilia Ernvik at Region Östergötland for taking the time to meet me.

Abbreviations

DBMS	Database management systems
JSON	JavaScript Object Notation
RDBMS	Relational database management systems
REST	Representational State Transfer
SQL	Structured Query Language
WCET	Worst Case Execution Time

Contents

Chapter 1

Introduction

Would it not be interesting to know how other people think?

At least it would be desirable for the designers and constructors who build products that are supposed to be used by others than themselves. If the product is well designed and constructed for a specific purpose but instead used for something different, the designers and constructors have failed. The reason for such a failure is often the lack of knowledge about the users and their needs. One example is Alfred Nobel's invention of the dynamite. The purpose of the invention was to find a less dangerous way of transporting nitroglycerin, for industrial use. Nevertheless, the military used the dynamite as a weapon. Instead of saving peoples' life the invention was used to kill people.

Fortunately, the effect is not usually that extreme when designing and constructing software products. The misuse of a product usually lead to a slower workflow for the user. To avoid those misuses it is necessary to understand and analyze the user. One way to obtain such information is to analyze how the user interacts with the product. A product with many users naturally produces a lot of information and this information could be difficult to store. The storing problem is the main objective for this report.

1.1 Background

Sectra has approximately 1600 customers across the world and they store medical information like for example patient information and images from different medical studies. However, they also store information about the environment which the medical system runs in, customer usage pattern and much more. One of the great challenges for Sectra today is how to analyze the data in order to improve their products and make their customers more satisfied.

The information about the customers is stored in log files usually structured as separated rows containing a time stamp column and a column with

the massage for the event. An example of two log rows is shown in Table 1.1. The main purpose for these files is to help customer support to get a better understanding of problems that are reported about the products. If an error occur in the product the customer support can analyze the log files and deduce the cause for the error.

Time	Massage
2015-05-18 10:53	Button X was clicked.
2015-05-19 14:56	User Y logs in.

Table 1.1: Example of a log file.

These files could also be used for analyzing how the customer interacts with the product. The files are not stored at Sectra but can be collected through a system called Sectra Monitoring. This report's focus is not on collecting the log files, it presumes that the log files exist locally at Sectra.

This report is written as a 30 credits, educational points, master in computer science and engineering final thesis at Linköping university with Jose Peña as the examiner and Dag Sonntag as supervisor. It was performed at Sectra Imtec AB with Alexander Sjöberg as supervisor. The audience should have a basic knowledge in computer science. The book *Fundamentals of Database Systems* [10] gives a good overview of the basic terms and theory mentioned in this report.

1.2 Goals

This report is about how to store large amount of information in a sustainable and efficient way. The information could then be analyzed to get a better understanding how the products are used. With such information it is easier to make strategic decisions that benefits Sectra and their customers. This information could for example be how often certain functionality is used and what functionality is never used.

One part of the goal is to create a system that can store the information from the log files. This is achieved by implementing a program that takes the log files as input, transform and store them in the database. The other part is finding a suitable database and that is achieved by testing and comparing different databases.

The main goal is not to analyze and visualize the data for business intelligence purpose. Business intelligence involves sieving through large amounts of data, extracting information and turning that information into actionable knowledge. However, analyzing the data is an extra goal if time is available.

1.3 Problem statement

The problem statement of the thesis is stated as:

- How can the information from Sectra's users be structured?
- What data storage is suitable for storing large amount of data with the two aspects scaling out and query performance in mind?

1.4 Research approach

Initially, a study of literature was performed in order to find what techniques and methods might be possible to use. Papers about big data was the first step to get a better understanding and knowledge in the problem area. Reading papers that compared different storing solutions were also helpful.

The approach to the problem was to look at how the data was organized in the log files and what kind of tools to collect and visualize the data that existed at Sectra. With that in mind, the idea was to search for different databases and tools that could solve the storage problem. Then the different databases were evaluated with focus on different aspects like for example performance. The different database types, relational based and NoSQL, were compared. Potential candidates for a NoSQL database were MongoDB, Cassandra and Raven.

1.5 Outline of the report

In Chapter 2 there is a description of the method used in this thesis and a list of the tools that were used. Chapter 3 gives an overview of the term database. The chapter also describes the NoSQL database and compare it to the relational database. A comparison between MongoDB and Cassandra occurs as well. An overview and the design of the system is shown in Chapter 4. Moreover, a description of the visualization tool and the structure of the data occurs in that chapter as well. The results from this thesis can be found in Chapter 5 which also explains the dataset and configurations of the databases. Then the chapter describes the results form structuring the data and from the queries. A discussion occurs in Chapter 6 about the results. The last Chapter 7 gives suggestions on future work. Furthermore, the conclusions are included in Chapter 6 and Chapter 7.

Chapter 2

Method

This chapter gives an overview of how the master thesis was conducted.

First an investigation of the existing data processing solution at Sectra was conducted. Knowledge about the solution was obtained by attending meetings with employees at Sectra. Moreover, a demonstration was held to obtain an even better understanding of the product.

The next step was to perform a literature study in order to find what techniques and methods might be suitable to use. Papers about storage solutions in general were read in order to get a good overview of the area. Later in the literature study papers that compared the relational database against the NoSQL database were read. Furthermore, to obtain information about NoSQL databases a study of papers that compared different NoSQL database solutions against each other was performed. An inspection of the homepage for the NoSQL databases [9] was conducted for those databases that seemed promising and interesting. In this report MongoDB and Cassandra were selected as the NoSQL databases. Moreover, MySQL was selected as the relational database with InnoDB as the storage engine.

The third step was to install the selected databases and implement the insertion program. All three databases were installed as single servers with no replica. Furthermore, Visual Studio was used to implement a program that inserted the data into the databases. The computer that was used has the specification shown in Table 2.1 and the versions of software shown in Table 2.2.

Component	Specification
CPU	Intel Xeon, 8 cores 3.40 GHz
RAM	16 GB
Disk	SATA 1 TB
OS	Windows 8.1, x64

Table 2.1: Specification table.

Software	Version
MongoDB	2.6.7
MongoDB shell	2.6.7
MongoDB C#/.NET driver	2.0
Cassandra	2.0.14
Cassandra C#/.NET driver	2.5
MySQL	5.6.24
MySQL C#/.NET	6.9.6

Table 2.2: Table of versions for software.

When the installation and implementation were completed, a set of tests were conducted. The purpose for those tests where to evaluate the performance and the usability of the different databases. Different aspects were tested like insertion time, final storage space, and query time. Moreover, a study of how to organize and structure the data in the database was conducted and tested. Papers concerning similar structuring of data were also considered.

The last step was to extract and visualize the data. There were several different ways to solve this and two solutions were tested. These experiments where performed to determent the most suitable solution for this project. The existing visualization tool at Sectra was QlikView which then naturally became the target system for the extraction.

Chapter 3

Theory

This chapter introduce the term database. A study of different data models, indexes, and scaling methods occurs. The chapter describes the most used and researched system for storing large amount of data. Moreover, a comparison of MongoDB and Cassandra, which are two different NoSQL databases, is performed.

3.1 About databases

The definition of a database is anything that can store data for later retrieval. However, this report only focuses on those that stores their data on a hard disk.

There has always been a need for storing information. Paper has been the traditional storage but when the computer became popular people could store information more efficiently. Today computer databases are common in the society and they store large amount of information. Therefore it is a natural choice to store the information about Sectra's users in a computer database.

A database system or database management system (DBMS) is a computer software that allows users and other applications to interact with the database. Examples of interactions are defining how the database should be structured or queried, i.e. retrieving information. For a database with personal information a query could be to find how many persons have the first name Dave.

When evaluating a system of any kind a good starting point is to locate the bottlenecks. A bottleneck is often a subsystem that is fully utilized and cannot increase its performance when the system is heavily used. An example is a bridge that has a speed limit. If the speed for the cars cannot increases due to the limit and more cars tries to cross the bridge the throughput of cars will not increase above the capacity for the bridge. Even if the highway to the bridge has a higher capacity, the road is bounded by

the bridges capacity. When evaluating databases a good starting point is to search for potential bottlenecks.

In a database system the hard disk is a bottleneck when considering the performance aspect. The main memory in a computer is in contrast to the hard disk a smaller but much faster memory, in reads and writes. The hard disk can be fifteen times slower than the memory bus which is a channel between the main memory and the central processing unit (CPU). Furthermore, the hard disk can even be forty times slower than the Peripheral Component Interconnect (PCI) 3.0 bus which is a fast bus for attaching hardware devices in a computer. In conclusion, the hard disk has an important role when evaluating a database system.

To solve the problem with the hard disk bottleneck many operating systems have implemented page caches. A page is a block of memory that an application can access. The purpose of page caches is to transparently optimize away the cost of disk access by storing contents of files in main memory pages mapped to the disk by the operating system. If the page is synchronized with the hard disk data, reads can be done from the page instead of the hard disk. The write operations modifies the page data but not necessarily write to the hard disk itself. The pages are stored in the main memory which leads to that reads and writes to the same local part on the disk will avoid interaction with the slow hard disk. This is usually beneficial for databases [23].

3.1.1 Different data models

A data model defines the format and the relations between data elements in a database. There are fundamentally three categories of data models: relational, key value, and hierarchical. The different data models have benefits and drawbacks depending on the requirements of the database. It is important to know those properties because it affects the outcome of how the system performs.

The relational model was very popular in the 1980s and 1990s due to that it allowed the database designer to minimize the data duplication within a database through a process called normalization. This was desirable because the hard disk was considered rare and expensive. However, lately the cost of disk storage has decreased making the benefit of low disk usage less important. Another drawback is that the relational model does not scale well with larger amount of data. The relational model is using the well-known Structured Query Language (SQL). The queries are typically required to access multiple indexes, joining and sorting result vectors from multiple tables. Such a query will not work well with datasets larger than one terabyte (TB) [23].

Another technique for storing data is to use key-value model. As the name implies, a key is associated to a value and works similar as a hashmap. The benefits are the fast, flexible, and easily understood storage model.

However, a drawback with this method is the absence of a schema which is a blueprint of how a database is constructed. The absence of a schema means that consistency checks are not performed and application logic is more complicated.

The last data model is the hierarchical model and is shown in Figure 3.1. The hierarchical model is ergonomic since the data is stored and retrieved in the same way as the object in the application. All relevant information for an object is stored in a single record and the records are structured into a tree-like structure. A record in the hierarchical database model corresponds to a row in the relational database model. Furthermore, the hierarchical model is the most flexible model for the programmer since the application data structures are the same as the data structures on the disk. The benefit of a single record is less overhead when extracting data compared to relational databases where information about an object can be located in different tables. However, a drawback with hierarchical model is that in order to retrieve a record in the low level of the tree the whole tree needs to be traversed starting from the root node.

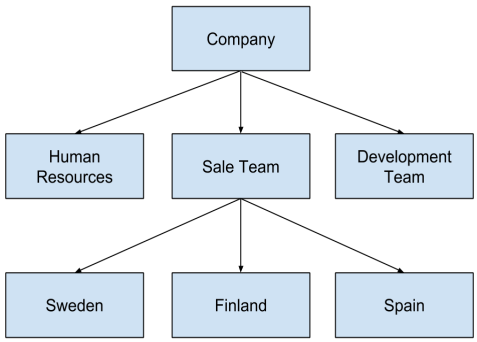

Figure 3.1: Example layout of a hierarchical based data storage.

Beside the three data models there are different ideas on how to store and retrieve the data. Two of the broadest categories of potential schemes for storage and retrieval are row based and columnar based. These two schemes can be applied to the models described above. For example, in the key-value model the row based model could be implemented by retrieving the key and the value at once. However, the columnar based model could also be implemented by retrieving all the keys first and then retrieve all the values. The same idea could be implemented for the hierarchical model in its records.

The row based schema stores the data row by row in a compact data structure. The rows themselves are contiguous in memory which means that it is optimized for fetching regions of entire rows of data at once. Figure 3.2

show an overview of how a row based schema can look like.

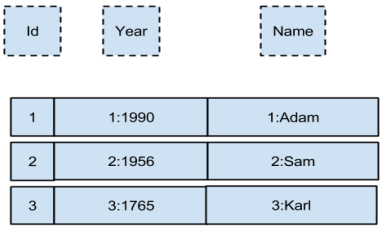

Figure 3.2: Example layout of a row based data storage.

Columnar based data storages are similar to the row based but now the columns are stored contiguously in memory. One advantage with this model is that columns have necessarily similar data structure which have a positive effect on the compression, increasing the amount of data that can be stored and retrieved over the bus. Moreover, breaking up the data into multiple files, one per column, can take advantage of the parallel reads and writes. One drawback with columnar based data storage is the inflexibility since the tightly packed data in columns. Furthermore, updates and insertions requires more coordination and calculation to find the data for a row. The Figure 3.3 shows how a column based data storage looks like.

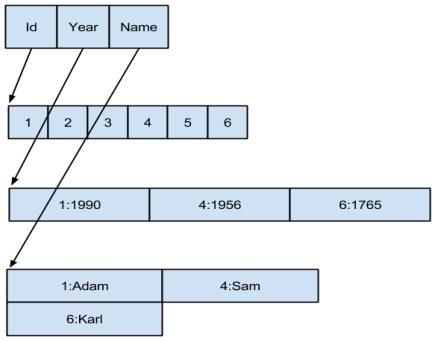

Figure 3.3: Example layout of a columnar based data storage.

3.1.2 Index

Index structure has an important role in a database since it improves query performance. Indexes are extra information added to the data in a database.

As mentioned by Arnab in *Fundamentals of Database Indexing and Searching* [3, Chapter 3.1], index structures are data structures that organize database objects for efficient searching and retrieval. Four properties are desirable of an index structure.

- Efficient access, the index structure should enable efficient searching and querying. I.e., it should take less time to scan through the entire database for an object than a linear scan.
- Small update overhead.
- Small space overhead.
- Correctness, which is the most important property. Using indexes should always answer the asked query correctly.

3.1.3 Vertical and horizontal scaling

In this project the scaling aspect of the database is important since large amount of data will be stored in the database. The test data from a hospital is around 2 gigabytes (GB) of raw data per month. This means that approximately 1 TB of data will be produced from all Sectra's customers per month. The amount of data to be stored, the storage, is of course depending on how long the data should be saved.

The methods of scaling a databases can be divided into two main categories, the vertical scaling and the horizontal scaling.

Vertical scaling is about upgrading the existing hardware on a computer or a node, i.e. a device in a network. Upgrading the hard disk with more memory or adding more main memory are examples of vertical scaling. The benefits are that it is easy and simple to upgrade the system without any configuration. The problem occurs when the system reaches the upper bound on the performance of the hardware. The hardware needed to upgrade such a system will be expensive, if it even exist.

The other measurement is horizontal scaling. In contrast with vertical, horizontal scaling is about adding more nodes to the system rather than upgrading a single node. The benefit is that there exists no upper bound for upgrading the system since more nodes can simply be added. Some other benefits are that the nodes often are cheap and can be replaced if failure occurs. However, the problem with this method is to implement the communication and structure of the nodes efficiently.

Since Sectra has several customers which generates large amount of data and the aim is to analyze the historical data, the horizontal scaling measurement seems to be suitable for this project.

3.1.4 Replica schema

The last database theory aspect in this report is replica schema. A replica schema defines how and where the data should be stored. The purpose of

the schema is to protect the stored data against a server failure. The basic method is to replicate the data on several servers instead of one. Furthermore, there are two basic replication schemes. The schemes are master slave and master master.

The replica schema master slave is constructed with master nodes that writes their data to slave nodes. The other one is multi master where all nodes can process update operations and propagate these to the other masters.

3.2 NoSQL

This section gives a short overview of NoSQL. These databases are interesting because they are designed to handle large amount of data.

NoSQL stands for "not only SQL" and not the misleading no SQL. NoSQL databases addresses the following points : [9]

- Non-relational.
- Distributed.
- Open source.
- Horizontally scalable.

Furthermore, some other characteristics that applies to NoSQL are: schema-free, easy replication support, simple API, eventually consistent/BASE, huge data amount, and more.

There exists several different classifications of NoSQL databases. Column, document and key-value are examples of NoSQL classifications. Column and key-value are described in Section 3.1.1 and the document classification is similar to the hierarchical model but focuses on storing all information of an object in a single record.

3.2.1 Horizontal scaling

The horizontal scaling method is the most suitable scaling method for this thesis as mentioned in Section 3.1.3. Furthermore, the authors Kuznetsov and Poskonin [17] explains the horizontally scalable aspect as the need to improve the performance by adding new computational nodes to the existing ones. This means that rather than upgrading memory and processors, more disks and processors are added to the database system. Benefits gained by this approach are a more scalable and cheaper system. In conclusion, if the system has performance issues it is easier and cheaper to add separated resources than to upgrade the existing hardware.

3.2.2 Distributed

In the case of distributed aspect there are two basic techniques used. The first approach is sharding. In this technique each node contains a piece of data called a shard and the node performs its operations on that piece. The second technique for distribution is replication where the same data is stored on several nodes of the network. A distributed environment of several nodes is called a cluster. The reliability, in both individual nodes and clusters, is improved in the system with the replication method. Read operations can also be improved since the system can have different search algorithms on the nodes. If the database should for example search for a time stamp, one algorithm could start from the earliest time stamp and the other one from the oldest time stamp.

3.2.3 ACID and BASE

One major difference between NoSQL databases and relational databases is that NoSQL does not fully support the ACID transactions. Those transactions are crucial for guarantee that the database transactions are processed reliably. ACID stands for:

- Atomicity; all operations in the transaction completes, or none.
- Consistency; before and after transactions the database is in a consistent state. A consistent state is when all constraints in the set of integrity constraints are satisfied.
- Isolation; operations cannot access data that is currently modified.
- Durability; data is not lost upon completion of a transaction.

Instead NoSQL databases justify the tradeoff, for not guarantee the ACID properties, by invoking the CAP theorem that stands for:

- Consistency; all nodes contain the consistent data at any time.
- Availability; when certain nodes fail, the remaining nodes continue to operate.
- Partition tolerance; if the system splits into disconnected groups of nodes, each group continues to operate.

Eric Brewer, who defined the CAP theorem [5], states that a distributed system can only guaranteeing two of the tree properties. He also introduced the BASE concept that stands for:

- Basically Available; an application works basically all the time.
- Soft State; the state of the system could change over time even during times without input.

- Eventual Consistency; the system will eventually become consistent once it stops receiving input.

The BASE concept is focusing on the two properties availability and partition tolerance in the CAP theorem. However, the ACID concept is focusing on the two properties consistency and availability which are more suitable for a single and central server [24].

In conclusion, the relational database favors the consistency and availability in the CAP theorem by guaranteeing the ACID properties. When the data can be stored on a single server the ACID properties are desirable. Although, when the amount of data increase the ACID properties do not scale as needed. However, the lesser strict BASE concept does this. This is one of the main reasons why NoSQL databases have increased in popularity over the last years.

3.3 Relational vs NoSQL

In this section relational and NoSQL databases are compared. It is good to know the differences between the two concepts when choosing the database for this project since it affects the performance of the overall system.

To compare the two different database management systems is not an easy task since there exist several different implementations for each system. PostgreSQL and MySQL are examples of relational database management system (RDBMS) commonly used today. However, there exist about 150 different NoSQL databases [9]. Furthermore, NoSQL databases can store its data in several ways like for example key value pairs, documents, columns, and more, as mentioned in Section 3.1. Comparing relational databases with NoSQL databases is not very straight forward because of the big structural differences between them. Hence, the comparison considers the overall characteristics of NoSQL databases and relational databases.

RDBMSs have been around since the 1980s and are well-documented and well optimized systems. They have a complete pre-defined schema and complies with the ACID transactions. Furthermore, RDBMSs use the Structured Query Language (SQL) which is well known to the community. Rick Cattell [6] proposed that many programmers are already familiar with SQL and that it is easier to use the high-level SQL rather than the lower-level commands provided by NoSQL systems. He also proposed that RDBMSs have successfully evolved to different load problems, for example in-memory, distributed, and horizontally scaled databases.

Moreover, Rick Cattell [6] also mentions some positive arguments for using NoSQL. Applications may require a flexible schema to allow different attributes for objects in the same collection. In most NoSQL databases this is not a problem but with RDBMSs this can be tricky due to the strict schema. Some RDBMSs allow new attributes at runtime and some allow missing attributes with an efficient data handling, although it is uncommon.

Another argument in favor of NoSQL is the absence of good benchmark tests which shows that relational databases can achieve comparable scaling with NoSQL systems, which may be out of date since the article [6] is from 2010.

The more flexible schema for NoSQL would be suitable for storing the users' behavior in Sectra's products since the log files may be structured different from case to case. Even a basic event like logging out from the product could differ. A user can for example push the log off button which triggers one type of event, however a user can also logout by being inactive for a particular amount of time which triggers another type of event.

Furthermore, the article by Adam Jacobs [15] conducts an experiment on the scalability of the RDBMS PostgreSQL. The test data was a representation of the world's population, 6.75 billion rows with 10 columns each. The total size of the test data is approximated to 100 GB. Since the inflation/unpacking for RDBMS caused to long execution time the data was reduced to 1 billion rows with three columns, which occupied more than 40 GB on the local disk. Then the query shown in Listning 3.1 was executed with different value for the maximum number of rows returned.

Listing 3.1: SQL query

```
SELECT country, age, sex, count(*)
FROM people GROUP BY
country, age, sex;
```

The execution time increased rapidly when the number of rows grew past one million. For one million rows the execution time was a couple of seconds but for ten million it was more than 100 seconds. When trying with the complete test data, 1 billion rows, the execution time was more than 24 hours. Unfortunately, the article does not mention anything about how long the execution time would have been for a NoSQL database. One of the conclusions in the article by Adam Jacobs was that it is easier to get the data in, as long you have enough disk space, than to get it out from a RDBMS. One other conclusion was that one must be willing to abandon the purely relational database model to achieve acceptable performance for highly ordered dependent queries on truly large data. Jacobs also describes the term big data as:

> "Data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time."

Furthermore, Adam Jacobs mentions that horizontal scaling is cheaper and more replicable than vertical scaling. That is one reason why NoSQL databases are popular. Moreover, from the beginning RDBMS did concentrate on optimizing for vertical scaling but have adapted to the horizontal scaling concept. Although, Jnan Dash [14] mentions that adding such a different concept to an already well defined system can be a challenge and the result may be expensive.

Additionally one comparison between RDBMS and NoSQL has been done in the article by Agapin, Boicea, and Radulescu [13]. The RDBMS

was an Oracle database and the NoSQL database was MongoDB. In the experiments MongoDB was shown to have better performance. Nevertheless, the article does not mention how the two databases were initially configured or which version. The result would have been more interesting if the Oracle database was supporting horizontal scaling, which presumably it did not.

Altogether, the NoSQL database seems to be the suitable solution for this project.

3.4 NoSQL-implementations

Given the conclusion from Section 3.3 two different NoSQL DBMSs were tested, MongoDB and Cassandra. These two were chosen because they are popular and they store data differently compared to each other. Cassandra uses the column based method and MongoDB uses the document based method for storing the data. Both methods are described in Section 3.1.1.

3.4.1 MongoDB

This section gives a short overview of MongoDB's features and functionality.

MongoDB is an open source project and it is a NoSQL implementation [18]. An open source project is a project were the source code is open for the public. Anyone can download the code, modify it, and publish their version to the community. Furthermore, MongoDB is written in the program language C++ and is a document oriented DBMS, as mentioned in Section 3.1.1. The documents have JavaScript Object Notation (JSON) data structure with field-value pairs. One example of a JSON document is shown in Listning 3.2.

Listing 3.2: JSON document

```
{
 field1: value1,
 field2: value2,
 ...
 fieldN: valueN
}
```

MongoDB stores documents on disk in the binary JSON (BSON) serialization format, which is a binary representation of the JSON document that contains more data types than JSON. Moreover, a document can not only contain field value pair but also nested documents and arrays. MongoDB joins documents into collections which is equivalent to an RDBMS table. For example, a table that describes patient information. A row would specify a patient and in MongoDB this would be a document instead of a row. A query in MongoDB targets a specific collection of documents and can be modified to impose limits, skips, and sort orders. The different types of data modifications are create, update, and delete.

One other feature in MongoDB is indexes which enhance the performance of common queries and updates. MongoDB also supports secondary indexes. These indexes allow applications to store a view of the portion of the collection in an efficient data structure. Moreover, MongoDB uses shard keys to distribute documents of a collection between nodes to achieve scalability. The asynchronous master slave replication, mentioned in section 3.1.4, is used. There exist two modes to change the consistency model. The first one is the non-blocking mode which does not wait for the confirmation from nodes that they have updated successfully. The second mode is the blocking mode which waits for a predetermined number of nodes.

Two more features in MongoDB are Map-Reduce and Aggregation-Framework which are techniques for forming a query from a sequence of steps, such as aggregation, projection, sorting, and more. MongoDB also supports several programing languages such as C, C++, C#, and Java [19]. Moreover, MongoDB also supports the operating systems OS X, Linux, Windows, and Solaris. The Representational State Transfer (REST) interface is supported as well, which is a software architecture style for scalable web services [22].

In conclusion, MongoDB is a popular and well-documented NoSQL implementation. It is easy and flexible to store data in MongoDB since the JSON format provides those two attributes. Furthermore, MongoDB supports C# which is used as the programing language in this thesis. Altogether, MongoDB seems like a suitable storage solution.

3.4.2 Cassandra

Cassandra is the second NoSQL database implementation that was tested in this project and this section will give a short overview of Cassandra's features and functionality.

Cassandra is an open source project, under the Apache license, written in the programing language Java. Furthermore, the structure for storing data is columnar based, as described in section 3.1.1. The data model is structured as three containers as the book *Mastering Apache Cassandra* [21] describes. The outermost container is the keyspace which is similar to the database in a RDBMS. The next container is the column family which can be seen as a table in a RDBMS. A column family consists of a unique row key which is similar to the primary key in a RDBMS. The row key contains a number of columns which is structured as a key and value pair. The Cassandra data model is shown is Figure 3.4.

Cassandra supports a query language similar to SQL, Cassandra Query Language (CQL). It also supports Map-Reduce and secondary indexes. Moreover, Cassandra was designed to ensure good scalability and reliability on a large number of inexpensive computers. That is achieved by the clusters in Cassandra which have a peer to peer structure which is equal to a multi master system, mentioned in Section 3.1.4. Furthermore, Cassandra supports several drivers for communicating with the database. Those drivers

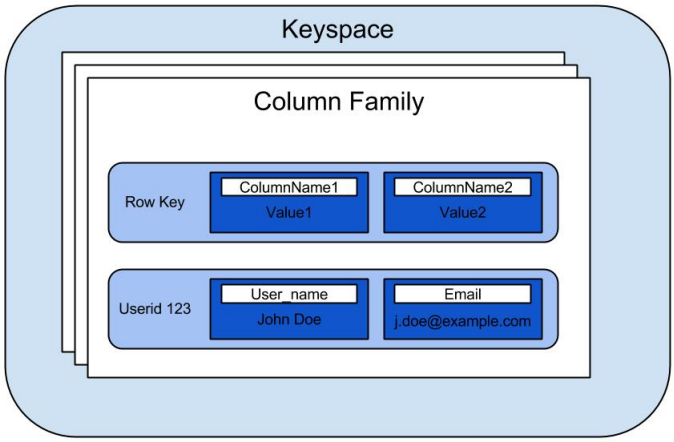

Figure 3.4: The layout of Cassandra's data model.

supports programming languages such as Java, and C# [11].

In conclusion, Cassandra is a popular and well-tested NoSQL implementation. In contrast with MongoDB, Cassandra's structure is more similar to a relational database. Furthermore, Cassandra's structure is less flexible than MongoDB's structure. The row keys are defined when the column family is created and can not be changed afterwards. Despite this, Cassandra seems like a suitable storage solution for this thesis since these remarks are minor difference.

3.4.3 MongoDB vs Cassandra

This section compares MongoDB and Cassandra and inspects related work. The purpose is to achieve a greater knowledge about the two databases and discover potential bottlenecks or performance issues.

Cassandra and MongoDB are chosen because they are popular among the NoSQL databases. Furthermore, both databases are open source projects with an active community. There have been 193 commits to the master branch by 42 authors for MongoDB on GitHub in one month [20]. For Cassandra there have been 112 commits by 24 authors on master in one month [12]. Although, the numbers do not compare the databases against each other since commits can be more or less important and the amount of added functionality can vary. The numbers shows that the communities are active and that features are added. Furthermore, C# driver exists for both databases which is preferable because there exists good competence within

Sectra in the .NET environment.

Disk usage

One difference between Cassandra and MongoDB, mentioned in section 3.2, is that MongoDB is document oriented and Cassandra is column oriented. Which one to prefer depends on how the information data is structured which is discussed in the book *Model-Driven Engineering Language and Systems* [16]. The book performs a benchmarking test with several NoSQL databases, including MongoDB and Cassandra, on how they performs in both time and disk usage for large and heterogeneous models. The Tinker-Pop Blueprints interface was used to make a more just comparison since NoSQL databases have different interfaces. It provides third-party drivers and APIs to several NoSQL implementations. Note that the results for Cassandra is the results for TitanDB who had Cassandra as its back-end database. However, the results should be similar to the results form only testing Cassandra since TitanDB only adds more functionality on top of Cassandra.

Because the test data in the benchmarking includes relations between objects, a workaround had to be made for MongoDB. An extra document was added for each relationship to represent the relation between two documents which increases the disk usage for MongoDB. The result shows that MongoDB had a much higher disk usage than Cassandra, approximated five times higher. However, the relation could be represented with an array inside the document instead of a separated document which would decrease the disk usage in MongoDB. The results from testing the insertion-time were similar for MongoDB and Cassandra with an approximated 15 % lower insertion-time for Cassandra with the larger data set. Moreover, the traversal performance were also tested in the benchmarking. The result shows that MongoDB had a much lower execution time than Cassandra, approximated to be 26 times lower.

Suggested by the book *Model-Driven Engineering Language and Systems* [16] was that MongoDB suffers at model insertion-time since MongoDB automatically creates indexes to address each document. The indexes for MongoDB have a negative effect on the disk usage since they are stored in the database. However, the indexes are beneficial as the traversal execution time shows. In conclusion, the article suggests that MongoDB scales well for querying large models.

Replication

Another difference between the two databases is that MongoDB is master slave oriented and Cassandra is master-master or multi master oriented, as mentioned in Section 3.1.4. Both methods are equally good in replicating the data. In the book *Performance Evaluation of NoSQL Databases* [1] a benchmark test was conducted using Yahoo! Cloud Serving Benchmark as

the framework. MongoDB was configured to use shards to achieve replication. A shard is a group of one or more nodes. The replication factor is determined by the number of nodes in a shard. Each member of a shard contains a copy of the data. Only one master is able to perform reads and writes in a shard, the rest are considered slaves and can only perform reads. However, the number of replicas for Cassandra can be controlled by a parameter set by the user.

Several different aspects were tested in the benchmarking. One of them was changing the number of cores on a single node. Cassandra and MongoDB had similar throughput using up to eight cores with a small advantage for MongoDB. Furthermore, Cassandra had a lower update latency and MongoDB had a lower read latency. In this project the read latency is more important than the update latency since updates are rare. Another test with different number of nodes had almost the same result as the test with different number of cores. Although, MongoDB had a better throughput with an increasing number of nodes and threads than Cassandra. For example with 8 nodes and 16 threads Cassandra got a throughput of around 8 000 operations per second and MongoDB got approximated 13 000 operations per second.

The number of replicas was also tested. Cassandra had a lower latency in both read and update. However, MongoDB had a higher throughput compared to Cassandra when the replica factor was set to one. With a factor set to two and three both databases were similar in throughput. However, one observation was that Cassandra got a steady degradation in throughput when the replica factor increased. On the contrary, MongoDB's throughput did increase when the replica factor was increased from two to three.

Conclusion of comparison

When comparing the disk usage for the two databases it seemed that Cassandra performed better than MongoDB. However, the result depended significantly on the data structure which Section 5.2 indicated. One other conclusion from the comparison was that Cassandra had a lower insertion time than MongoDB. Although, results from the experiments conducted in Section 5.3.1 indicated the contrary. The comparison with the replication aspect showed that both databases seemed to perform approximated equally with a small favor for MongoDB.

Chapter 4

Design and Implementation

This chapter gives an overview of the existing system and describes the design decisions of the modifications made.

4.1 System overview

This report focuses on the Sectra product picture archiving and communication system (PACS). A simplified setup of a PACS consists of a server and a client. The server stores pictures and corresponding data and handles request from the client. One example of a request could be a series of images of a brain scan for a specific patient. When a user interacts with the client its actions are logged to a log file. These files are stored locally at the hospital and are removed after a configurable time.

The Figure 4.1 shows an overview of the system. The stored log files are fetched by a system called Sectra Monitoring. When a pull request is received, Sectra Monitoring sends the log files to a server located at Sectra's office in Linköping. Then the program Command Usage Tool reads the files and writes Extensible Markup Language (XML) files as output. The purpose of the Command Usage Tool is to transform the log files into XML files since the visualization program QlikView, explained in Section 4.2, can't import the original files. However, another purpose of the program is to modify and restructure the data according to some requirements. The last step in the system is to configure QlikView to read the XML files and create charts that is visualizing the data.

One drawback with the system in Figure 4.1 is that all the steps are executed manually. First someone must collect the log files and then execute the program Command Usage Tool. Then it is necessary to run and configure QlikView to read from the XML files that Command Usage Tool

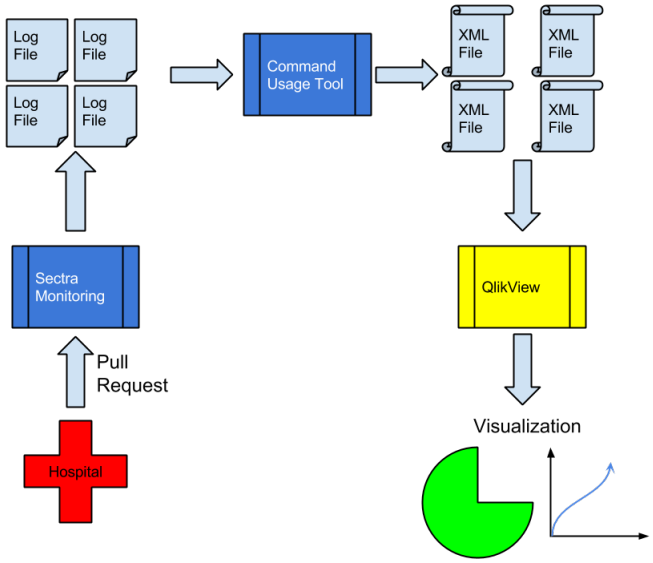

Figure 4.1: Overview of the existing system.

has produced.

The main focus for this thesis is to replace the XML files with a database solution. Figure 4.2 shows the new overview of the system. The major difference is that the Command Usage Tool stores the information from the log files into a database instead of XML files. There are benefits for storing the data in a database instead of in a file structure. One benefit is that a database is designed to store similar data structures. Moreover, by forcing the data to comply with certain requirements and by structuring the database in a suitable way, one can compress the data remarkably. This leads to more information per byte in the database compared to the file system. This is important when the data grows in size.

However, one drawback with a database compared to the XML files is that some sort of connector is necessary since QlikView needs to import the data. The problem is that QlikView has a relational view of the internal data. However, there exist several connectors for both MongoDB and Cassandra.

Chapter 5 discusses the positive effects of having a database instead of the system shown in Figure 4.1. One positive effect is that the database reduces the storage space with more than half of the original size as Table 5.2 shows. Another benefit is that the data is more accessible due to the database management system. Instead of a manual search through the file system for some specific information a query could be sent to the DBMS.

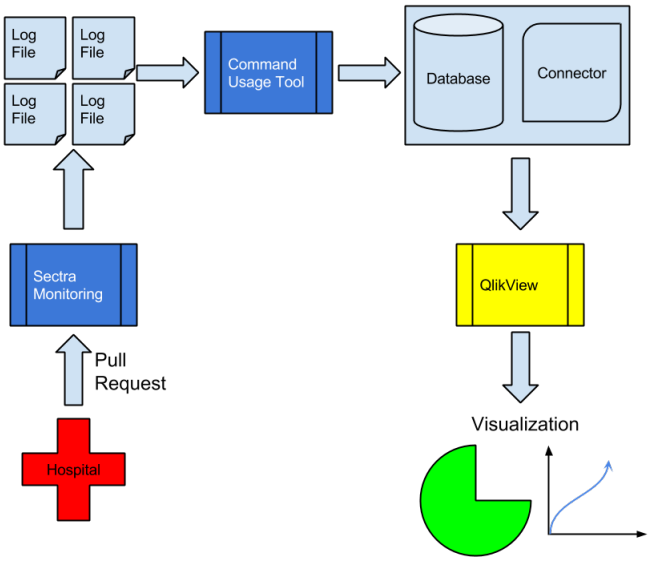

Figure 4.2: Overview of the new system.

4.2 Visualization tool

The visualization tool used in this thesis is QlikView [2]. QlikView is a business discovery platform that supports in-memory technology. In-memory means that all the data is stored in main memory. The benefit with in-memory is that aggregations can be calculated without any overhead from a hard disk. This leads to that QlikView has interacting data. For example, if there is a table listing different hospitals the analyst can select a hospital with a simple click and the charts will then only show data corresponding to that hospital.

However, a bottleneck with in-memory is the capacity of the main memory. QlikView can only handle as much data as the main memory can hold. However, QlikView compresses the data to minimize the bottleneck and in the best case down to 10 % of its original size [2].

In addition to QlikView, other visualization tools were considered. One of them was D3.js [4]. D3.js is a code library for the program language JavaScript which is used to create web sites. A problem with this solution was that it stored the data, from the database, as plain JSON. The plain JSON data was not compressed, which made it difficult to analyze the data. Therefore, D3.js was excluded as a possible visualization tool.

4.3 Structuring the data

The data structure in a database is crucial for the performance. Query performance can be improved with a good data structure. Furthermore, compression of the data can also be improved and those improvements are interesting for this thesis.

The structure of a log file at Sectra is that one log event corresponds to one row. Moreover, a row has some connecting attributes, the most interesting ones for this project are shown in Table 4.1. Normally, the log files are not as structured as in Table 4.1. The *Message* attribute for instance, does not always contain only the event code, but also less relevant information such as the license associated with the user.

Time	Host	User	Massage
2015-03-18 10:53	Workstation 1	Doctor Y	Login
2015-03-18 10:56	Workstation 1	Doctor Y	Select patient X
2015-03-18 11:00	Workstation 2	Doctor X	Login
2015-03-18 11:01	Workstation 1	Doctor Y	Rotate screen
2015-03-18 11:06	Workstation 1	Doctor Y	Logout

Table 4.1: Example of a log file from Sectra.

4.3.1 Unstructured

There are several ways to structure a log file. The simplest one is to assign a row in the log file to a corresponding row in the database. However, this method has some drawbacks. For one thing, much data will be redundant like for example the host and user information. This information is repeated in every row and can be aggregated to only appear once. Another drawback with unstructured data is that unnecessary events will be stored in the database like for example the scroll function. When a user scrolls trough a stack of images every scroll will correspond to one event. Since a stack can contain hundreds of images, lots of scroll events will be logged.

4.3.2 Structured

There exist no silver bullet for structuring a dataset. The quality of the structure is strongly dependent on the original information and the intended use. One possible structuring process is discussed in the article by Doru Tanasa and Brigitte Trousse [8]. The article describes the preprocessing of information from Web pages and the problems are similar to the ones in this project. There are some users that interacts with a system which generates log files. Furthermore, the log files also contain unstructured data with timestamps and events. They make the user anonymous which is important for protecting the user's integrity. The anonymity is also important for

Sectra users and should be implemented. Besides, it is more interesting for the analyst to look at a group of users than a specific individual when performing analyzes.

Data cleaning is another step in the preprocessing described in the article. Tanasa and Trousse clean the data by removing image and Web robot requests. However, such data does not exist in the log files from Sectra there are some data that can be removed. Two example are the failed login attempts and the scroll events. At the same time it is important not to remove information that could be useful for the analysis.

The next step in the process is data structuring. One interesting concept that the article mention is user session. A user session is a number of events that starts with a login and ends with a logout. Defining a session like that is not always accurate, since the user might not log out immediately after he or she has stopped using the application. Unfortunately, the forced logout setting is optional. For Sectra products with this setting switched off it becomes more complicated to determine the end event for a user session.

Figure 4.3 shows an overview of how a session can be structured. However, the structure in the databases will not be exactly the same since the layout in the database can differ. None of Cassandra or MongoDB support foreign key constraints. However, the same behavior can be achieved by implementing matching identifications when inserting the data in this project. For example, sessions have an integer attribute named *User_Id* which has the same value as the integer attribute *Id* in a user document and thereby connecting them. Nevertheless, this implementation in the NoSQL databases does not have all the features that a relational database has like for example consistency checks.

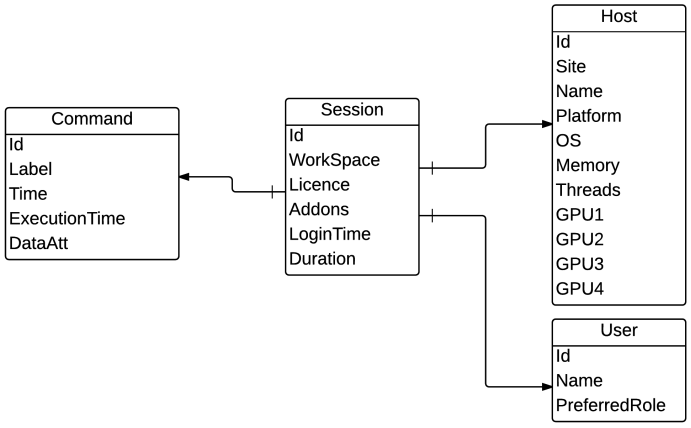

Figure 4.3: Overview of a session.

Chapter 5

Results

This chapter shows the results from the experiments that were performed using the modified system described in Chapter 4. The databases that have been evaluated are MongoDB, Cassandra, and MySQL.

5.1 Dataset and configuration

The dataset that has been used for the evaluation is the log files from a specific hospital for the period of one month. The dataset contains approximately 4 380 000 rows and is 1 316 megabytes large on the file system.

MongoDB and Cassandra were not configured from the default configuration throughout the tests except when indexes were added. Moreover, both databases is setup to use one node with no replicas. On the other hand, MySQL had to be configured due to the slow insertion time. Moreover, the buffer pool for MySQL's storage engine InnoDB was increased from the default value of one GB to four GBs. This lead to that MySQL could use more main memory to read, write, and store data before using the disk. Furthermore, the increased capacity of the main memory effects the page caches positive since more pages can be stored in main memory.

One other configuration that was changed for MySQL was how it should flush its log file to disk. Flush means that the changes done in the log file are written to disk and then cleared from the log file. The default setting was to flush to disk at every commit, which is similar to each query. This was changed to once per second. Additionally, the XA two-phase commit support, which reduces the number of disk flushes due to synchronizing on disk data and the binary log, was disabled for InnoDB.

The effect of the flush and XA two-phase commit support configurations is that MySQL does not apply strictly to the ACID properties. If the server crashes between two flushes the changes from the last flush are not written to disk.

5.2 Storage capacity

This section shows the result of how well the different databases managed to store the data. The storage capacity is not the most important aspect for a scalable database since hardware is cheap. However, the results shows how the database grows in size over time which is interesting when choosing the storage capacity on the server.

The setup of the experiments were simply done by inserting the dataset with the specified structure into the databases and then extract the information about the storage size from the databases.

5.2.1 Unstructured

Figure 5.1 shows the result of storing the dataset with the unstructured method mentioned in Section 4.3.1. The label *Original* in the graph refers to the size of the unmodified log files. Cassandra managed to compress the storage size of the dataset with a factor of 0.50 while MongoDB on the other hand increased the storage size with a factor of 1.58. MySQL increased the storage size with a factor of 1.40.

In conclusion, the results from this experiment indicates that Cassandra compresses the data well within a table. A reason for the good compression by Cassandra could presumably be that Cassandra internally compresses the columns. If a column has a low covariance an aggregation of the column could reduce the data usage significantly. This shows the benefit of using a column structure for storing data. Both MongoDB and MySQL stores the data with a row structure which indicates that row structure is not suitable for storing unstructured data.

5.2.2 Structured

To evaluate which was the most suitable data structure for this project another experiment was performed. In contrast to the unstructured data used in the previous experiment the data was structure as sessions, mentioned in Section 4.3.2. The results from structuring the dataset as sessions are shown in Figure 5.2. To achieve relations between objects in the database a 32 bits integer was used for all databases. Furthermore, MongoDB was tested with two different structures. One with all information in a single document, named *MongoDB* in the figure. In the other structure, named *MongoDB Norm* in the figure, both host and command information were extracted and inserted into a separated collection as described in Section 5.2.3. The result from the normalized MongoDB experiment indicates that the compression is benefiting from the normalization.

The results from the experiment also shows that both Cassandra and MySQL decreased their storage usage. In contrast to the unstructured experiment MongoDB did, with normalized structure, compress the data better than Cassandra.

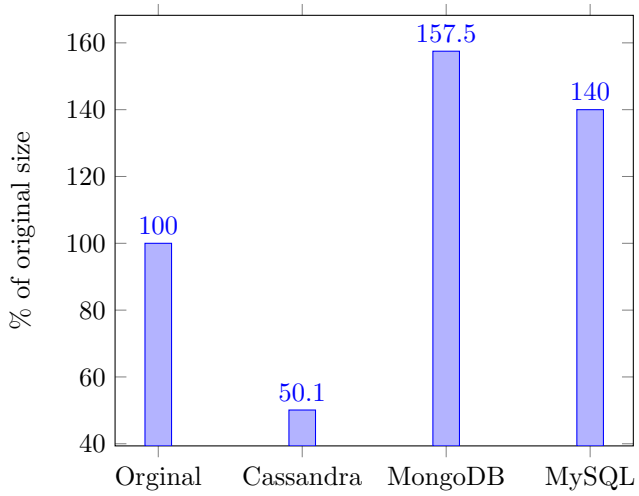

Figure 5.1: Storage graph, structuring as assigning a row in the log file to a row in the database.

In conclusion, this experiment shows that structured data is more suitable than unstructured data when considering the storage use aspect. The experiment also indicates that it is important how the data is structured.

5.2.3 Datreregation

When storing large amount of data it can be beneficial to aggregate the information in the database to decrease the storage capacity. Table 5.1 shows the results from structuring and aggregating the data with the structure described in Section 4.3.2 and shown in Figure 4.3. MongoDB was used as the test database for this experiment due to its flexible data structure. However, the results are dependent on the dataset and database and should be interpreted as guidelines.

Different techniques were used to structure and aggregate the information. Only structuring the data does not solve all the problems with disk usage. The difference results between *MongoDB* and *MongoDB Norm* in Figure 5.2 shows the benefits of aggregating the structured data.

The first idea was to aggregate the information that is repeated in a session. Both host and command information applies to that. The second idea was to remove unnecessary information by either removing events or replacing the date type with a smaller information type.

To extract the host information, a pointer was created from a session in the session collection to the corresponding host in the host collection. The pointer is a 32 bits integer which has the same value as the identifier attribute at the host. The integer then has the same functionality as a foreign key

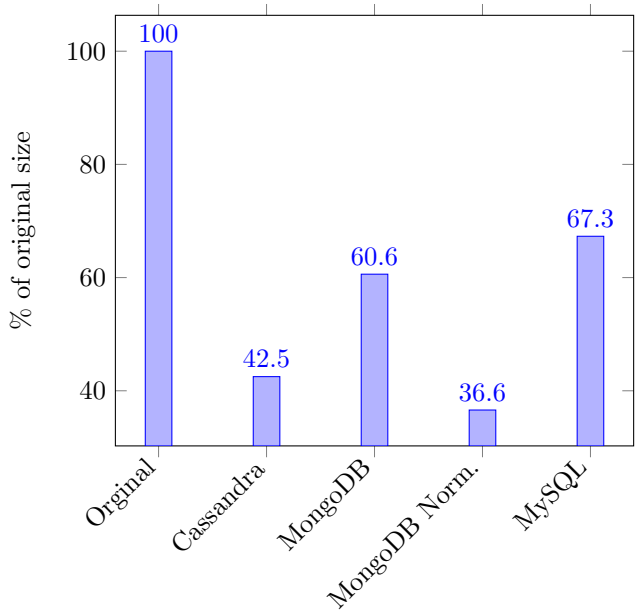

Figure 5.2: Storage graph, structuring as sessions.

in a relational database. The host extraction does not only reduce the storage capacity, it also simplifies host specific queries. Instead of iterating through the entire session collection to collect the hosts' information, a query will iterate through the smaller host collection. Moreover, the extraction is similar to the normalization procedure in a relational database with the goal to reduce duplicated data.

A similar extraction was performed for the commands. However, the command document has an attribute that differ for the same command. Therefore, not all information could be extracted from a command document to remove duplication. Since a session can contain several commands and a command can occur in several sessions this is a many to many relation. In a relational database a table is created to represent this relation. However, in MongoDB this relation can be achieved by a document array in the session document and then create a pointer to the redundant information which is stored in a separated collection. A document in the array contained an integer as pointer and a text value which contained the specific information about the command. In conclusion, it can also be noted that the reduce factor was better for extracting the commands compared to hosts since there are more commands.

Not all information in a log file is valuable for analytic purpose. The log files were analyzed in order to find unnecessary events. The analyze method was to calculate how often a command was repeated. The results showed

that two commands were common, which both corresponded to scroll events. It was discussed with a Sectra employee with the conclusion that those two commands could be removed which reduced the storage amount with 17 %.

The last reducing method was to replace the time stamps in a command document. The time stamp was replaced with an offset that was calculated from the first event in a session. The consequence was that a 32 bits offset determined the time for a command instead of a 64 bits time stamp. It would be preferable to use an integer with fewer bits but MongoDB's smallest integer is 32 bits long.

In conclusion, both structuring and aggregating the information in a database have an impact on the storage capacity. The greatest impact was the aggregation of command information since this information is heavily repeated.

Structure	Storage Capacity (MB)	Reduce %
Original, single document	798	0
Extract host information	778	2.5
Extract command information	501	37.2
Remove unnecessary events	663	17.0
Offset on time	787	1.4

Table 5.1: Storage capacity for different storage solutions with MongoDB as test database with the dataset discussed in Section 5.1.

5.3 Query performance

One of the most important aspects when considering a database is the query performance. It is not only the storage capacity that needs to scale with the larger amount of data, the query performance should scale as well. To measure how the database's queries performs different experiments were conducted.

The procedure for executing the experiments was to first start and configure the database. After that a stopwatch was started and then the query was executed. When the query returned with the result the stopwatch was stopped and the time was measured. Furthermore, the stopwatch was implemented in the program to achieve a more correct measurement and the query was executed several time and then the average value was calculated.

5.3.1 Inserting data

Table 5.2 shows the execution time of the insertion queries for the databases. The insertion in MongoDB is done by grouping the queries in a bulk and then executing the entire bulk. The purpose of a bulk insertion is to decrease the execution time. However, both Cassandra and MySQL supports

a preparation query which looks like a regular query but the values are variables that can be changed for each query. The benefit of a preparation query is achieved when it is used several times but with different values. Only the values are sent the second time which reduces overhead. A preparation query was defined for Cassandra and MySQL.

The result indicates that MongoDB has a lower insertion time than Cassandra and MySQL. Nevertheless, insertions will not occur manually or with a high frequency so the insertion time is less significant than the extraction queries. However, both Cassandra and MongoDB insertion time were constant with the growing capacity in the database. This was not the case for MySQL since its insertion time grows with the growing amount of data in the database. One explanation to MySQL's problems was that it tried to store all the data in main memory and when it was unable to fit the dataset in memory MySQL had to read and write to the hard disk which decreased the performance significantly. Moreover, when inserting the dataset for the fifth and sixth time in MySQL the insertion time took approximated 9 hours and 22 hours.

Time (m:s)	Cassandra	MongoDB	MySQL
Insertion with row structure	13:25	3:15	22:00
Insertion with session structure	13:28	0:22	13:01

Table 5.2: Execution time of insertion queries. The dataset is the one mentioned in Section 5.1.

5.3.2 Extracting data

The database stores more data than a computer can store in the main memory. A consequence is the necessity to extract only a subset of the data in the database. After consulting with employees at Sectra three main types of queries were specified for extracting the subset. The first query was to extract only the information regarding a specific hospital. The second one was to extract sessions that had performed a specific command. The last query was to only collect information over a specified period of time.

Hosts

The goal for the host query was to find and count all rows that had a predefined value for the host attribute. The query was executed ten times and the average value was calculated. The results from the search queries are shown in Figure 5.3. The dataset that was used is the one shown in Section 5.1 and it is structured as the simple row structure mentioned in Section 4.3. However, the dataset has been multiplied by two to achieve a larger dataset. Furthermore, the databases have been indexed on the host

attribute. The horizontal axis in Figure 5.3 shows the number of hosts that exists for the different search values.

The results indicate that MongoDB performs better with queries that increase in size compared to the other databases. One reason for this result could be that MongoDB's indexes works better than the other databases' indexes and thereby reducing the search time for the search algorithm. This could be beneficial to the system that should extract a subset of the data in the database. A filter system performs the extraction and the results indicate that such a filter would be faster in MongoDB compared to Cassandra. Furthermore, the performance of the filter is dependent on the corresponding filter system. If the filter system for MongoDB is not configured and implemented well the increase in query performance is lost.

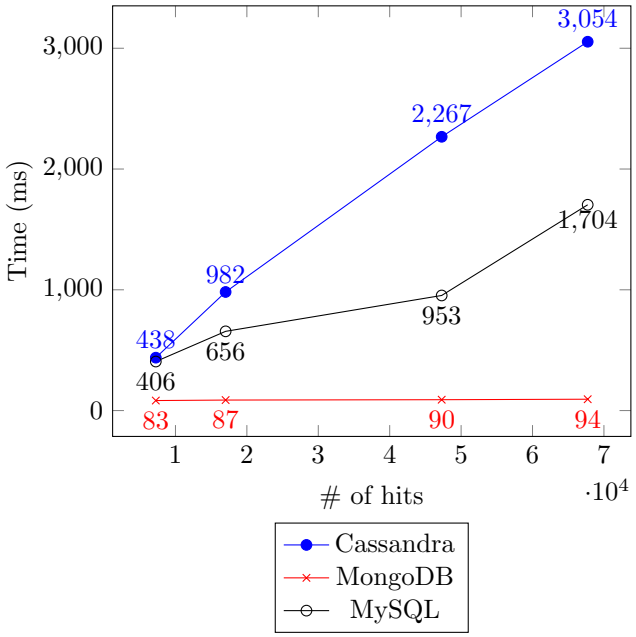

Figure 5.3: Chart of the search time for host queries with unstructured data. The horizontal axis shows the number of hosts found by the query. For example, the smallest query found around 17 000 rows with that specific host.

Commands

The command query aspect was tested with a command that existed in every fourth session which means that 25 % of the total data in the database was extracted. The results from the queries are shown in Figure 5.4 and the dataset was structured. The horizontal axis shows how many times the

dataset was inserted in the database. For example, the number two indicates that it was multiplied twice which means that the database stored twice the amount of data. The exact numbers can be found in Table 5.3.

The results presented for MySQL was when the target data fits in MySQL's buffer pool. When this was not the case MySQL had to load data from disk which increased the query time. The worst case execution time (WCET) for MySQL escalated with the growing amount of data in the database. Furthermore, the result shows that when MySQL can't store all necessary data in main memory its performance decreases.

In conclusion, this experiment indicates that Cassandra and MongoDB performed similar for extracting the structured data. The experiment also indicates that the current configuration for MySQL does not scale well with the growing amount of data in the database. Section 5.3.1 also describes this problem with big data for MySQL and the main issue was the strong dependency of storing the entire dataset in main memory. Both Cassandra and MongoDB does not depend on storing the entire dataset in the main memory which was beneficial when the data outgrow the capacity of the main memory.

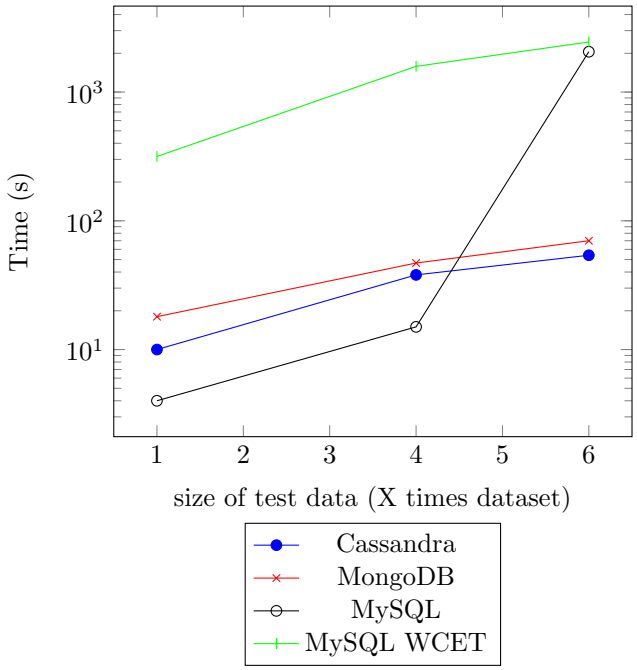

Figure 5.4: Chart of the search time for command queries with different size of the structured dataset.

Dataset X	Cassandra (s)	MongoDB (s)	MySQL (s)	MySQL WCET (m:s)
1	10	18	4	5:16
4	38	47	15	26:24
6	54	70	2057	40:49

Table 5.3: Table of execution time for command queries. The left column shows how many times the dataset were inserted.

Time

Figure 5.5 shows the results from extracting data for a specified time interval. The dataset was multiplied, structured, indexed on the timestamps, and a month was added with each multiplication. The time interval for the query was one month.

The result shows that both Cassandra and MongoDB scales well with the growing amount of data. The query performances from the two NoSQL DBMSs were nearly constant despite the increase of data in the databases. This indicates that both the databases' indexes works well. However, MySQL was only tested with the dataset multiplied with six due to growing insertion time mentioned in Section 5.3.1.

One conclusion from this experiment was that both Cassandra and MongoDB were more preferable compared to MySQL since their query performance scaled well with the growing amount of data in the database. The problem for MySQL was the dependency on storing the entire dataset in main memory which the previous results also proves. MySQL had to write and read to the slower hard disk when the main memory was maximized which decreased its performance significantly.

5.4 Extracting data into QlikView

This section shows the results of extracting the data from the databases into QlikView. These results were important for the thesis because they indicate the total amount of time the analysts waits before they can start working. One benefit with QlikView was that it saved the data to an internal file which significantly reduced the load time for the second start up. Although, if the analyst wanted another time interval it was necessary to perform a new extraction. Therefore, the results, shown in Figure 5.6, were measured when QlikView loaded the data for the first time. The horizontal axis shows how many months the query asked for and the query asked for a sequence of months, for example January, February, and Mars.

The connector between the databases and QlikView was created and implemented from scratch since the connectors on the market did not meet the requirements. The procedure for extracting the data to QlikView included the following steps. First the analyst entered the desirable time interval to the connector which then queried the database. The result from the query

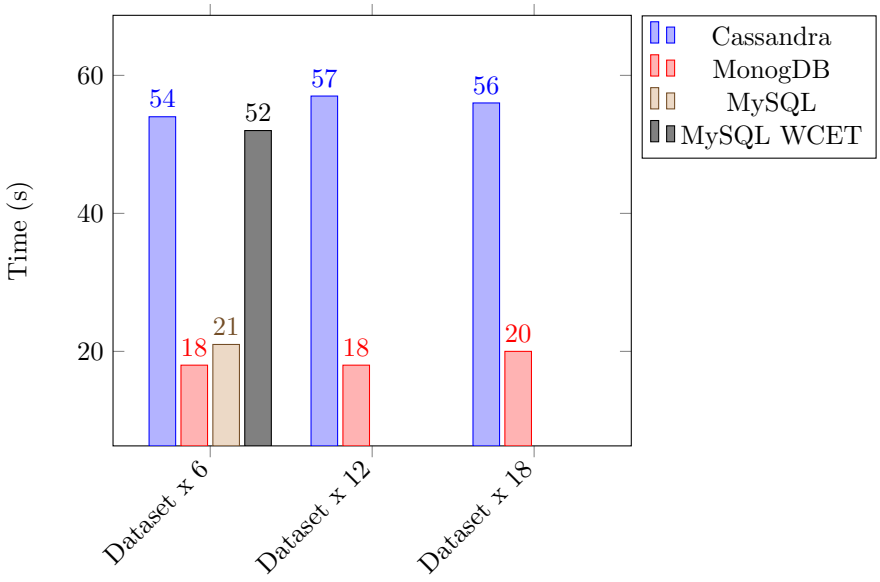

Figure 5.5: Chart of the search time for timestamps queries with different size of the dataset. The timestamps attribute is indexed.

was then written to XML files. The next step was to load the XML files into QlikView which was performed by QlikView's internal load system. The last step involved structuring the data in QlikView according to the analyst's demands.

The setups for the databases were configured accordantly to Section 5.3.2 and the same dataset was used. The dataset was inserted 18 times and with each insertion the date attribute for the session was incremented with one month. In other words, the database contained an even distribution of sessions for 18 months to simulate a real world scenario. The databases were indexed on the date attribute in a session. MySQL was excluded from this experiments since the poor insertion time.

In conclusion, the results show that MongoDB was slightly faster than Cassandra. One explanation for the result was the better query performance of MongoDB compared to Cassandra shown in Figure 5.5.

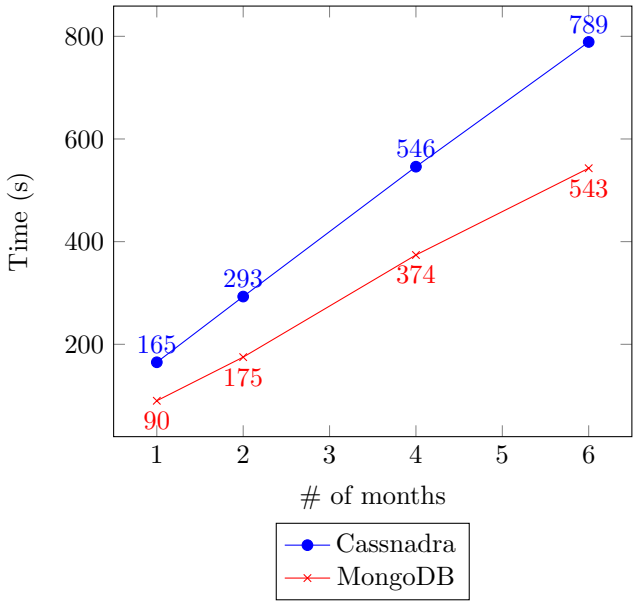

Figure 5.6: Chart of the execution time for extracting the data into QlikView.

Chapter 6

Discussion and Conclusions

In this section the results and methods are discussed. Other possible solutions are also studied and discussed.

6.1 Method

The method used in this thesis could be improved. First of all, the dataset was constructed of real data from a hospital which made the results domain specific and less general. In order to achieve the same results as this thesis the same dataset should be used which is hard since the dataset was classified. Although, using real data made the results more realistic and interesting for Sectra and the application domain at hand.

To achieve a wider perspective of the big data area some other companies were consulted. The companies were Spotify, which provides a music streaming service, and Region Östergötland, who is responsible for the health care in Östergötland. They gave interesting thoughts about how to store big data. The meetings could have occurred earlier in the project in order to have a larger impact. However, the meetings were held as early as possible in the project but scheduling problems postpone the meetings.

One aspect that was not tested was a solution with several servers. The test could indicate how the databases handles the internal division of data and the communication between the servers. However, the book *Performance Evaluation of NoSQL Databases* [1] tested Cassandra and MongoDB with different number of nodes and it gives some guidelines on how the databases performs in that area.

6.2 Results

The results were significantly dependent on the configuration of the database and the test data. Therefore, the results from this thesis were dependent on the current configuration and on the hardware of the test computer.

An overview of all the results conducted in this report shows a favor for NoSQL databases compared to relational databases. However, relational databases have other benefits compared to NoSQL databases, like for example constancy checks. This drawbacks for NoSQL databases are not an issue for Sectra.

6.2.1 Storage results

Figure 5.1 in Section 5.2 shows that Cassandra stored the unstructured dataset well compressed in contrast to MongoDB and MySQL who both increased the storage capacity. This indicates that the column based structure used by Cassandra works well with compressing unstructured data compared to the row based structure used by MongoDB and MySQL. One explanation to this is the similarity between the values in a column. For example, if a column stores integers an average value can be calculated and then the database only needs to store the offset from the average value. In other words, instead of storing several bits per value the database only stores some few bits that is defining how far from the average the value is. This explanation is a hypothesis and should be treated as such.

On the other hand, Figure 5.2 shows that structuring the data had a significant impact on the storage capacity. MongoDB compressed the structured data much better than the unstructured and even more compressed than Cassandra. MySQL did also compress the structured data better.

In conclusion, Cassandra is the best choice for unstructured data with the storage capacity in mind. However, in the case when the data is structured both MongoDB and MySQL closed the gap to Cassandra significantly. With the modifications shown in Table 5.1 MongoDB got a better result than Cassandra. Therefore, MongoDB is the best choice when storing structured data when it comes to storage size. One reason for MongoDB's better result is its flexible data structure.

6.2.2 Query results

Query performance was an important aspect for the databases since they should scale well with the growing amount of data in the database. Altogether, the results indicated that MongoDB had the best overall performance and that MySQL had problems when that data capacity grew.

Insertion query

The insertion time, mentioned in Section 5.3.1, was not the most important aspect of the database because it should not occur often. However, Table 5.2 shows that MongoDB had a faster insertion time than both Cassandra and MySQL. Another interesting result was that MySQL had problems when the storage capacity grew. One explanation to MySQL's problems was that it tried to store all the data in main memory and when it was unable to fit the dataset in memory MySQL had to read and write to the hard disk which decreased the performance significantly.

Extraction query

The results in Section 5.3.2 shows that MongoDB's queries performed better than the other databases' queries for unstructured data as shown in Figure 5.3. However, MongoDB's performance advantage came with a higher disk usage cost, which Figure 5.1 shows.

The command extraction, displayed in Figure 5.4, was another proof that MySQL had problems with the growing amount of data. The performance of MySQL decreased remarkably when the data capacity were larger than four times the original dataset since the whole dataset did not fit in main memory. However, both Cassandra and MongoDB performed better than MySQL with a small advantage for Cassandra.

Figure 5.5 shows that both Cassandra and MongoDB scaled well with the growing amount of data. This is an important result since it indicates that both Cassandra and MongoDB indexes works well even if the data capacity grows. MySQL was only tested with the first dataset since the insertion time grow exponentially.

Data into QlikView

The results in Figure 5.6 were influenced by the results in Figure 5.5 since the query was the same. However, the insertion time for QlikView was added to the result and it grow linearly with the growing data size. The query for both databases did also grow linearly which Figure 5.4 reflects as well. In conclusion, both Cassandra and MongoDB performed well with extracting a time interval from the database into QlikView. However, MongoDB had a slight advantage compared to Cassandra.

It can be noted that the connector between the database and QlikView can be optimized to achieve a lower execution time. One improvement could be to parallelize the writes to the XML files. Another improvement could be to change the output format to a simpler one. The XML format introduces overhead since it adds tags to the values. However, XML files are easier for a human to read and was therefore chosen. Comma-separated values (CSV) is a simpler format that could potentially decrease the execution time for the connector.

6.3 Other solutions

There exists several other possible DBMSs for this thesis and it is worth mention them to get a wider perspective.

6.3.1 Hadoop

Spotify stores large amount of data since they have over 50 million users. Spotify's methods are interesting to this report since they also stores log files from users and analyze them. They store parts of their data unstructured in a Hadoop cluster. Apache Hadoop is an open source framework that allows distributed processing of large data sets across clusters of computers using simple programming models. Hadoop adopts the horizontal scaling method and supports Map-Reduce functionality. In contrast to this report, Spotify has other requirements for their system like for example the massive insertion speed. To handle that problem they have teams that work with getting the data into the cluster at a high speed and there is no time for modifying the data. Hence, it is up to the analysts to collect and structure the data from the cluster in order to analyze.

In conclusion, the benefit of storing the unstructured data in a Hadoop cluster is that the analysts can structure the data exactly how they want. However, some drawbacks with a Hadoop cluster is that it needs constant maintenance of the servers and requires more work from the analysts in order to collect the data and structure it.

6.3.2 Google BigQuery

Another possible DBMS is Google's BigQuery product which provides servers and interfaces to interact with the data. Google is a multinational technology company which provides Internet services and products. One advantage with BigQuery is that Google handles the servers with replicas and clusters, in contrast to Hadoop where this is done manually. Furthermore, Google charges for data storage and for querying data, but loading and exporting data are free of charge. This is an important aspect when comparing BigQuery to Hadoop. The cost for a Hadoop cluster would be the servers and wages for the employees maintaining the cluster. The cost for BigQuery is only determined by how large the dataset is and the query frequency.

Chapter 7

Future work

This section discusses different topics in the thesis that can be further investigated.

7.1 Databases

In the thesis only three databases were tested. There exists several other databases that can be tested to achieve a wider prospective and maybe find a more suitable database. It would be interesting testing other types of solution as well, like those mentioned in Section 6.3.

The setup and configuration of the database is another aspect that could be investigated more. The large social media Facebook is using MySQL [7] as part of their storage solution which indicates that it is possible to use MySQL for storing big data.

7.2 Data structure

One aspect that should be further investigated is the structuring of the data. There are benefits and drawbacks with both unstructured and structured data. The results from the thesis shows that structuring the data can lead to compression. However, Cassandra could store the unstructured data fairly compressed in contrast to MongoDB and MySQL. Then a Map-Reduce job could collect the data from Cassandra and structure it according to the analyst's demands.

Bibliography

[1] W. Knottenbelt R. Osman A. Gandini, M. Gribaudo and P. Piazzolla. Performance evaluation of nosql databases. In *Computer Performance Engineering*, 8721, pages 16–29. Springer International Publishing, 2014.

[2] QlikTech International AB. `http://www.qlik.com/us/explore/products/qlikview`. Collected 2015-4-15.

[3] A. Bhattacharya. *Fundamentals of Database Indexing and Searching*. Taylor & Francis Group, 2015. eBook.

[4] M. Bostock. `http://d3js.org/`. Collected 2015-4-16.

[5] E. Brewer. Towards robust distributed systems. `http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`, 2000.

[6] R. Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, December 2010. Newsletter.

[7] Oracle Corporation. `https://www.mysql.com/customers/view/?id=757`. Collected 2015-7-8.

[8] Tanasa D. and Trousse B. Advanced data preprocessing for intersites web usage mining. *Intelligent Systems*, 19:59–65, Mars 2004.

[9] S. Edlich. `http://nosql-database.org/`. Collected 2015-3-3.

[10] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2011.

[11] Apache Software Foundation. `http://wiki.apache.org/cassandra/ClientOptions`. Collected 2015-3-4.

[12] Apache Software Foundation. `https://github.com/apache/cassandra`. Collected 2015-3-6.

[13] A. Boicea I. Agapin Laura and F. Radulescu. Mongodb vs oracle - database comparison. In *Third International Conference on Emerging Intelligent Data and Web Technologies*, 2012.

[14] Dash J. Rdbms vs. nosql: How do you pick? http://www.zdnet.com/article/rdbms-vs-nosql-how-do-you-pick/. Collected 2015-3-6.

[15] A. Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36–44, August 2009.

[16] R. Paige L. Rose K. Barmpis, D. Kolovos and S. Shah. A framework to benchmark nosql data stores for large-scale model persistence. In *Model-Driven Engineering Languages and Systems*, volume 8767, pages 586–601. Springer International Publishing, October 2014.

[17] S. Kuznetsov and A. Poskonin. Programming and computer software. *NoSQL Data Management Systems*, 40:323–332, November 2014.

[18] Inc MongoDB. http://docs.mongodb.org/manual/. Collected 2015-3-4.

[19] Inc MongoDB. http://docs.mongodb.org/ecosystem/drivers/?_ga=1.21005371.1689776219.1424343476. Collected 2015-3-4.

[20] Inc MongoDB. https://github.com/mongodb/mongo. Collected 2015-3-6.

[21] N. Neeraj. *Mastering Apache Cassandra*. Birmingham : Packt Publishing, 2013. Chapter 1, eBook.

[22] RestApiTutorial.com. http://www.restapitutorial.com/. Collected 2015-3-4.

[23] R. Richardson. Disambiguating databases. *Communication of the ACM*, 58(3):54–61, 2015.

[24] P. Vanroose and K. Van Thillo. Acid or base? - the case of nosql. http://www.abis.be/resources/presentations/gsebedb220140612nosql.pdf, June 2014. Collected 2015-3-3.