Analysis of Database Systems

Implementing a database system for the advertisement engine Adengi

LUND INSTITUTE OF TECHNOLOGY
Lund University

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Mohammad Rahimpur
Raghed Ali

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2011

Abstract

Adengi is an advertising system for mobile applications. The system was developed by *Crunchfish AB* with the objective to be used globally by advertising agencies. The system requires an efficient scalable database system to maintain the availability of information in the server at any hour of the day. It can be done in particular by distributing information on various servers around the world. Unfortunately, the current database does not meet this requirement without compromising the performance on the server.

This thesis describes and analyzes the databases of interest to Crunchfish AB. The purpose is to convert the current relational database to the selected non-relational database.

The analysis consists of two parts:

1. *An analysis of the current database and why it does not meet the system requirements.*
2. *A comparative analysis of the non-relational databases Neo4J, MongoDb and Apache Cassandra.*

With the information that was held from the analysis and advice from Crunchfish AB, Apache Cassandra was chosen.

The database is describes in depth and also fully implemented. For communication with Apache Cassandra database server an API in PHP was developed, which was integrated with the web-based interface.

Keywords: Adengi, Advertising system, Database, Apache Cassandra, PHP, API

Sammanfattning

Adengi är ett annonseringssystem för mobila applikationer. Systemet utvecklades av Crunchfish AB med målet att det ska användas globalt av reklambyråer. Behov uppstår då av att upprätthålla tillgängligheten av informationen i servern dygnet runt. Det kan göras bland annat genom att distribuera information på olika serverar runt om i världen. Dessvärre uppfyller inte den nuvarande databasen detta krav utan att försämra prestandan på servern.

I denna rapport beskrivs och analyseras de databaser som är av intresse för Crunchfish AB i syfte att omvandla den nuvarande relationsbaserade databasen till den utvalda databasen.

Analysen består av två delar:

1. *En analys av den nuvarande databasen och varför den inte uppfyller systemets krav.*
2. *En jämförelseanalys på de icke-relationsbaserade databaserna Neo4J, MongoDb och Apache Cassandra.*

Med den information som erhölls av analysen och med råd från Crunchfish AB, blev valet av databasen Apache Cassandra.

Denna databas beskrivs djupgående i rapporten och implementeras i systemet Adengi. För kommunikation med Apache Cassandra databasserver implementerades ett API i PHP som integrerades med det webbaserade gränssnittet.

Nyckelord: Adengi, Annonseringssystem, Databaser, Apache Cassandra, API, PHP

Foreword

This bachelor thesis has been made in collaboration with Crunchfish AB during spring 2011 at Lund University, LTH School of Engineering. The project includes analysis of database systems and the timeline equals to 15 weeks of work.

We want to thank our examiner Mats Lilja and tutor Christian Nyberg for their support and valuable comments. Also a special thanks to Paul Cronholm and Thomas Gårdängen for great guidance and aid.

I would also like to thank my family who has been involved throughout my education and listened and encouraged me.

- *Mohammad Rahimpur*

I would like to thank my fiancé and family for their support and help.

- *Raghed Ali*

Helsingborg, June 2011

List of contents

1 Introduction

Crunchfish AB is a Swedish company that develops mobile applications for iPhone, iPad, Windows 7 phones and Android. The company has been in existence for just over a year and has already managed to become big in the mobile application world. The company's passion is developing innovative applications focusing on usability and high performance. This has made them a leading application developer in Scandinavia.

The competition and the development of applications for smart phones have grown tremendously. It takes no more than a week before a new app store has opened and new download records have been set. One idea that certain firms have been thinking about is the so called *app in advertising*, marketing directly to applications. Analysis shows that in 2014, 1.5 % of global marketing campaigns will be directed towards mobile advertising and this equals 5.7 billion dollars of the total amount spent on advertising [1].

With this idea in mind, Crunchfish AB desires to implement an efficient and user friendly system that is oriented towards *app in advertising.*

1.1 Problem description

Crunchfish AB has implemented a system that handles advertising in applications called *Adengi*. The current system is in its early stage and needs improvements in efficiency, usability and availability.

The main idea of this system is to offer advertisers the possibility to reach mobile phone users with great accuracy. Advertisements can be presented depending on the user's geographical location, weather, time and categories of interest. In this way the ads can be directed towards the right crowd and people.

Due to the loads of data that needs to be written and read quickly and efficiently, the current relational database model is not the right choice.

Therefore the problem of this thesis is to:

- *Analyze and compare the non-relational database systems against Crunchfish AB needs of efficiency and high availability.*
- *Choose and analyze in depth one of the non relational databases.*
- *Rewrite the current database model to the chosen non relational data model.*

1.2 Goals

The primary goal of the thesis is to find a database that satisfies the needs of a scalable, highly available and efficient database system.

A secondary goal is to convert the current database model to the non-relational database model and integrate the converted database into the Adengi system.

1.3 Delimitations

This thesis will only give suggestions for a solution to the database and API. The product is not intended to provide a high commercial quality.

2 Work Method

The work of this thesis is divided into categories and phases.

Planning

The first step was to set up a plan for the fifteen weeks of work in form of a Gantt-chart, figure 1.

Analysis

The analysis will be done around the architecture of Adengi and determine which databases are of interest for further analyzing.

Phase 1

The aim of this phase will be to analyze and to understand the data model of the chosen non-relational database.

Phase 2

This will be the development phase. The main focus will be at converting the existing database model to the new data model.

Phase 3

The main focus in this phase will be to test the database against a web based interface and start implementing an API for the communication between database and web interface.

Phase 4

Revision test and an acceptance test of the system including the web-based GUI and the Android application.

2.1 Time Plan

Week number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Calendar week	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Planning	▨	▨														
Analysis		🟦	🟦													
Phase 1			🟦	🟦	🟦	🟦										
Phase 2						🟦	🟦	🟦								
Phase 3									🟦	🟦	🟦					
Phase 4												🟧	🟧			
Report writing		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		
Reporting to tutor			🟥		🟥		🟥		🟥			🟥				
Reporting to examiner														🟥		
Preparation for presentation														🟥	🟥	
Presentation of the thesis																🟫

Figure 1: Time plan described in Gantt-schema

When we set up a time plan for our project we decided to use a modified version of the *water fall* process model. This is because the main purpose of this project is to analyze and understand new technologies. We decided to do this in phases.

The *agile* process model will be used when implementing the API for communication with the database. The reason for using the *agile* process model is because we will implement a function, then test it and if needed re-implement the function. This will make it easier to integrate the database with the Adengi webpage.

3 Analysis

This chapter will cover the analysis part of this thesis. The analysis will be on the databases that we chose together with Crunchfish AB.

The following questions will be answered in this analysis:

- ***What*** *are the disadvantages with the current relational database model?*
- ***How*** *does the data model work for the different databases?*
- ***What*** *are the advantages and disadvantages of the databases?*
- ***Which*** *database did we decide to choose and why?*

3.1 The current database

Today the Adengi database is based on the RDBMS *MySQL,* which is a limited and non efficient database model for storing large amount data. The main reason for not using MySQL is because of the weakness for handling of scalability.

3.1.1 Scalability

Scalability in a database management system is the ability to handle an increasing amount of requests to the servers. There are two options for scalability, either adding servers (*horizontal scaling*) or upgrading the hardware on the existing servers (*vertical scaling*) [23]. When working with relational databases it is preferred to upgrade the existing hardware. This is because the relational databases are not supposed to be used as a distributed database system. The problem with vertical scaling is when the data increases and exceeds the limitation of the hardware and there are no other ways than distributing the data across other machines. Another disadvantage when using vertical scaling is that it is impossible to upgrade the hardware without bringing the server down, which affects the uptime.

Other strategies for achieving scalability are by applying "*sharding*" or "*replication*":

Sharding
Sharding is a method of horizontal scaling in which data is divided into portions (shard) and distributed across different servers within the cloud. There are different strategies for handling partitions across servers. The least complex strategy is to move a heavily used table to another server. But the problem with this strategy is that it has to be implemented to the system from the start.

If this is not done from the start, the system will end up losing uptime because it is time consuming to move data from one shard to another shard. The sharding process is not really supported in MySQL, there are some tools and libraries under development and these can help to understand the sharding process but, in the end the user ha to implement sharding by himself [25].

Replication
Replication is a technique that stores redundant copies of data in several servers. A standard MySQL strategy is to create a master and slave node where the slave node(s) can hold replicas of the data. A “write” has to be sent through a master-node while “reads” can be directly read from a slave-node. The problem appears when there are more writes than reads which will affect the performance drastically, because the master-node will be a bottleneck. One way to avoid this would be by writing to any server in the cluster, but this cannot be done in MySQL. The reason for that is because of the nature of MySQL. The writes have to go through a master-node before being counted as a successful write.

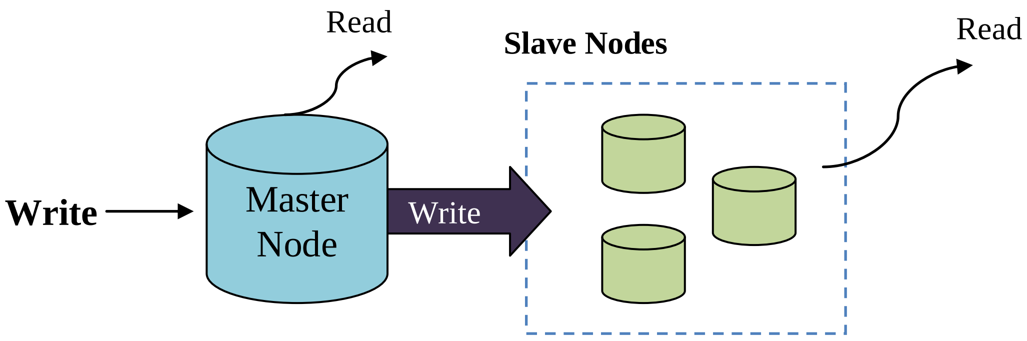

Figure 2: Writing and reading to a MySQL cluster

Figure 2 illustrates the master-node problem within the MySQL database. Here it is shown that the writes have to go through a master-node before being sent to a slave-node. The read operations are directly being read from the slave nodes.

The problem with the master-slave model is that it is optimized for reading data. This allows data to be read from any node within the cluster. But when replicating data it is a one way communication from the master to the slaves. In this model the master nodes acts as authoritative sources for data, and the slave only synchronize their data against the master’s data.

3.1.2 Normalization

To achieve high performance in a relational DBMS the tables need to be normalized [26]. This can be an issue when new features have to be added to the system, resulting in new tables that need to be normalized. It increases the chance of de-normalizing the existing tables, which means that the re-normalization of the old and the new tables must be done once again.
This process is time consuming and is not desirable when expanding the existing system with new features.

3.2 Brewer’s CAP Theorem

The CAP (Consistency, Availability, Partition Tolerance) theorem states that within a large-scale distributed data system it is impossible to simultaneously provide these three guarantees [24].

1. *Consistency*
   All clients (nodes) have access to the same data at the same time, meaning a query gives the same value/update to all nodes.
2. *Availability*
   If a node fails or gets locked in the cluster it should not affect the availability of a certain data or resource.
3. *Partition tolerance*
   The system should continue to function, regardless of failure within the system. E.g. if the nodes within the system cannot communicate with each other due of a network failure, the result of querying either partition should return the correct data.

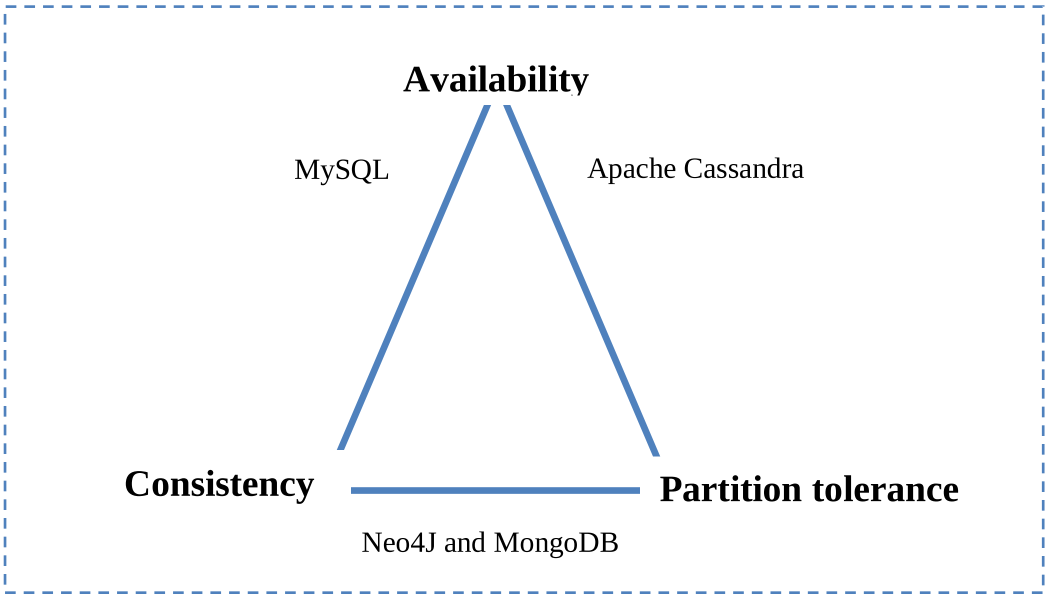

Figure 3: Where different databases appear in Brewer’s theorem

The figure above shows how the database systems focus on these three factors. This does not mean that they ignore the third factor, only prioritize the two that are the most important.

3.3 Databases

Together with Crunchfish it was decided to analyze the three NoSQL databases, *Neo4J, MongoDB* and *Apache Cassandra*. These three databases could replace the current relational database to fulfill Adengi system's needs of scalability.

3.3.1 Neo4J

Neo4J is an open source graph database that stores data as structured graphs. It has been in production for over seven years and has been in commercial development for ten years [7].

Neo4J is designed for network-oriented data and is based on nodes, relationships and properties. These primitives of large networks are called a *node space.*

3.3.1.1 Data Model

The data model is based on collection of nodes with edges that connect pairs of nodes. Each node contains a value and the node name is the key for that node. Relationship connects two nodes and can hold information in them.

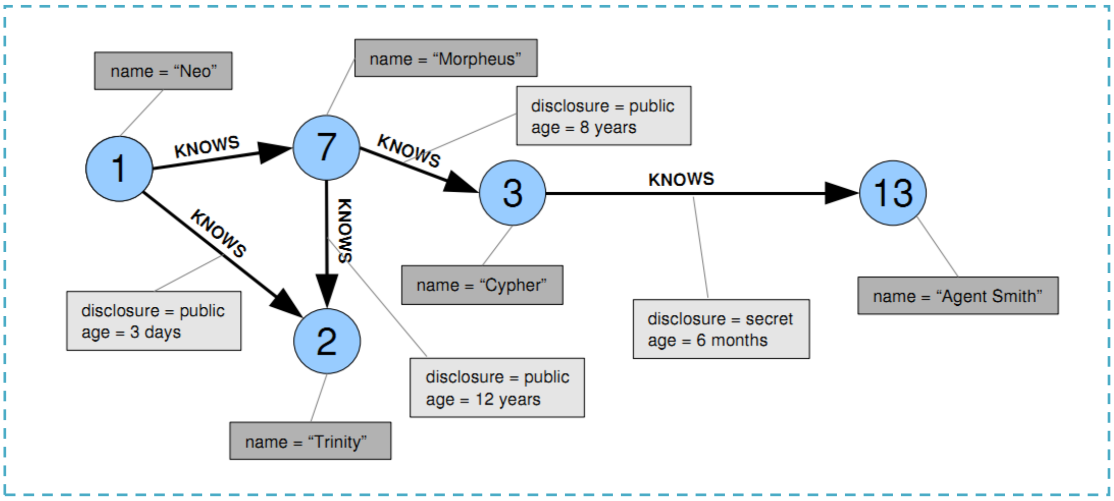

Figure 4: A graph in Neo4J [7]

Node

A node in a graph represents an object, e.g. a person including a value and a name. Each node contains a set of properties defining that object, in a key – value set. In the example above the node two has only one property with 'name' as key and 'trinity' as value.

Relationship
There is a relationship between two nodes. Like in a RDBMS, they can hold value and the relation becomes the primary key. All relationships have a value, through this value you can access the data that the relationship holds or the values of nodes that are related.

3.3.2 MongoDb
MongoDb is an open source, document-oriented and high performance database written in the C++ language [9].

With MongoDb, there is less "normalization" when designing a relational schema since there are no server-side joins. Instead focus is on having a database collection for each of the top level objects.

3.3.2.1 Data Model
A MongoDb system holds many databases that can be sharded and replicated as shown in the picture below [8].

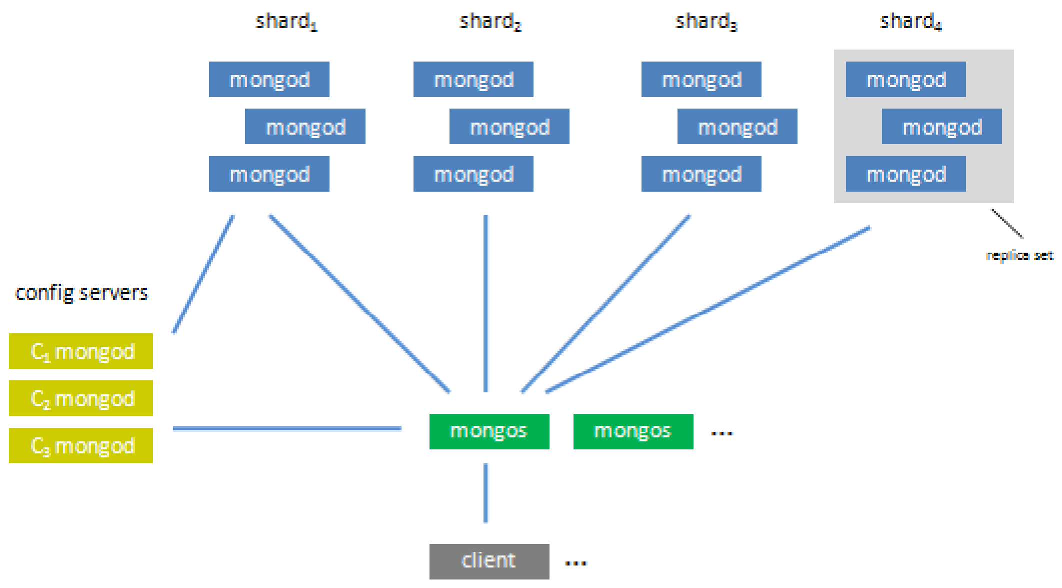

Figure 5: Shards in a MongoDb system [8]

In figure 5 we see how data is divided into portions and then distributed across multiple shards. There is also the option of having one or multiple replicas that hold a backup for a shard. In MongoDb there is always one primary server and the rest are secondary. When the primary goes down, one of the secondary will then be promoted to become a primary. Which secondary that will be promoted depends on which secondary is the one most recently updated.

Collection
A collection is like a table in a relational database. The only difference is that it holds documents instead of rows. Also, instead of having a collection for every class they have embedded objects within a collection.

Documents
A document is basically a set of fields where every field holds a key which is associated with a value. This is called a key-value pair. Every document must have a key-id that identifies the document within the collection.
In MongoDb a document can hold another document as value, called "*embedded document*". They are structured as an ordinary document where the only difference is that it is embedded into another document.

Keys
Every embedded document or a value has to have a key so that they can be identified in the document. A key is a normal string, any UTF-8 characters are allowed, but there are some minor exceptions when choosing characters.

Value
The values are the information for the associated key and can be all from basic data types as string, integers, timestamps etc, to embedded documents and arrays.

3.3.3 Apache Cassandra
Apache Cassandra is an open source column-oriented database developed by Facebook to solve their inbox search problem, in which they had to deal with large volumes of data. In March 2009 Cassandra was moved to an Apache Incubator project and since then it has been maintained by them [4].

Since Apache Cassandra was the main choice to implement, this database is described in depth in a different chapter, [Chapter 4].

3.4 Conclusion

In *Appendix B* there is a table that compares the three database types. These three have almost the same features and the focus is on distributed systems. What this table does not cover is the amount of time a user has to spend to understand the concept of the databases model. With the time we had to test how easy it was to set up the databases, we noticed soon that apart from Apache Cassandra, the two other ones required more time. This was mainly because of the amount of information available for Neo4J and MongoDb. This was one of the factors that made us choose Apache Cassandra. Another reason for choosing Cassandra is because it focused mostly on the availability and the partition tolerance [Chapter 3.2], and as we were informed in the beginning of this project to find a database that mostly focused on having the most uptime and that it could keep its uptime even if a server within the cloud would crash. The other two databases did not focus on availability like Apache Cassandra did. Instead they focused on the consistency and partition tolerance, and this was the main objective why they were ruled out when deciding which database to implement the Adengi system on.

The three databases are all capable of handling scaling well and puts their focus on distributed systems, but with the time we had we could not determine which of these three databases we should use. This is because all three databases are relatively new and we felt that the information that was available was not sufficient. To determine which database to suit the system best we had been forced to use and test the databases and then compare them in sense of efficiency and availability when making different operations, but that is a thesis project in itself. Therefore, because of lack of time and the information that was hold, we were advised by Crunchfish to go ahead with Apache Cassandra database model and try to convert the relational model into a non-relational model.

4 Apache Cassandra

This chapter describes Apache Cassandra in depth and will cover up the theoretical part that is needed before implementing [3].

The main features of Apache Cassandra are:

Distribution	Cassandra is capable of running on multiple machines while appearing to the end users as a single node.
Decentralization	There is no single point of failure (SPF). Unlike database models like MySQL and Bigtable, Cassandra does not need a master node to organize other nodes within the cluster.
Scalability	Refers to a special property of horizontal scalability. It means that the cluster can scale up and down by accepting new nodes or removing nodes [23].
Availability	Replacing failed nodes with other nodes in the cluster without affecting the uptime is done by replicating data to multiple nodes.
Consistency	Returns the most recently written data by comparing timestamps within the replicas. More about consistency at [Chapter 4.5].

These features make it relatively easy to construct a data model that can handle changes. Because Apache Cassandra does not require normalization, meaning data structures can be added without affecting the performance.

4.1 Data model

The Cassandra data model is structured for distributed data on a very large scale. The model trades ACID-compliant data practices for advantages in performance and availability. The data is stored in a 5-dimensional hash map and is very flexible and easy to work with in many programming languages [5].

Keyspace
Keyspace is an outermost container for column families and data in Cassandra. Like in a RDBM schema a keyspace has a set of attributes that defines its behavior. These attributes are *Replication factor, Replica placement strategy* [Chapter 4.4.2] *and Column families.*

Column Family
Column family contains columns of related data. The structure is a tuple of a key-value pair where each key (row) is mapped to a set of values (columns).

Row

Each column family has an unspecified number of rows. The rows are sorted by their data type. Each row consists of a row key which the columns within that row are referenced with.

Column

It is a triplet that holds a name, a value and a timestamp. The timestamp is used when updating a value or retrieving the newest updated value. In this way a read will give the most recently written value in a cluster. Timestamps are provided by the client, not by Cassandra itself.

Super Column Family
A super column family contains a set of *super columns*.

Super Column
Each super column holds an unbounded number of columns. The main difference between a column and a super column is that columns values are *string* and in super column the value is a *map* of columns. Besides that, super columns do not have any timestamp.

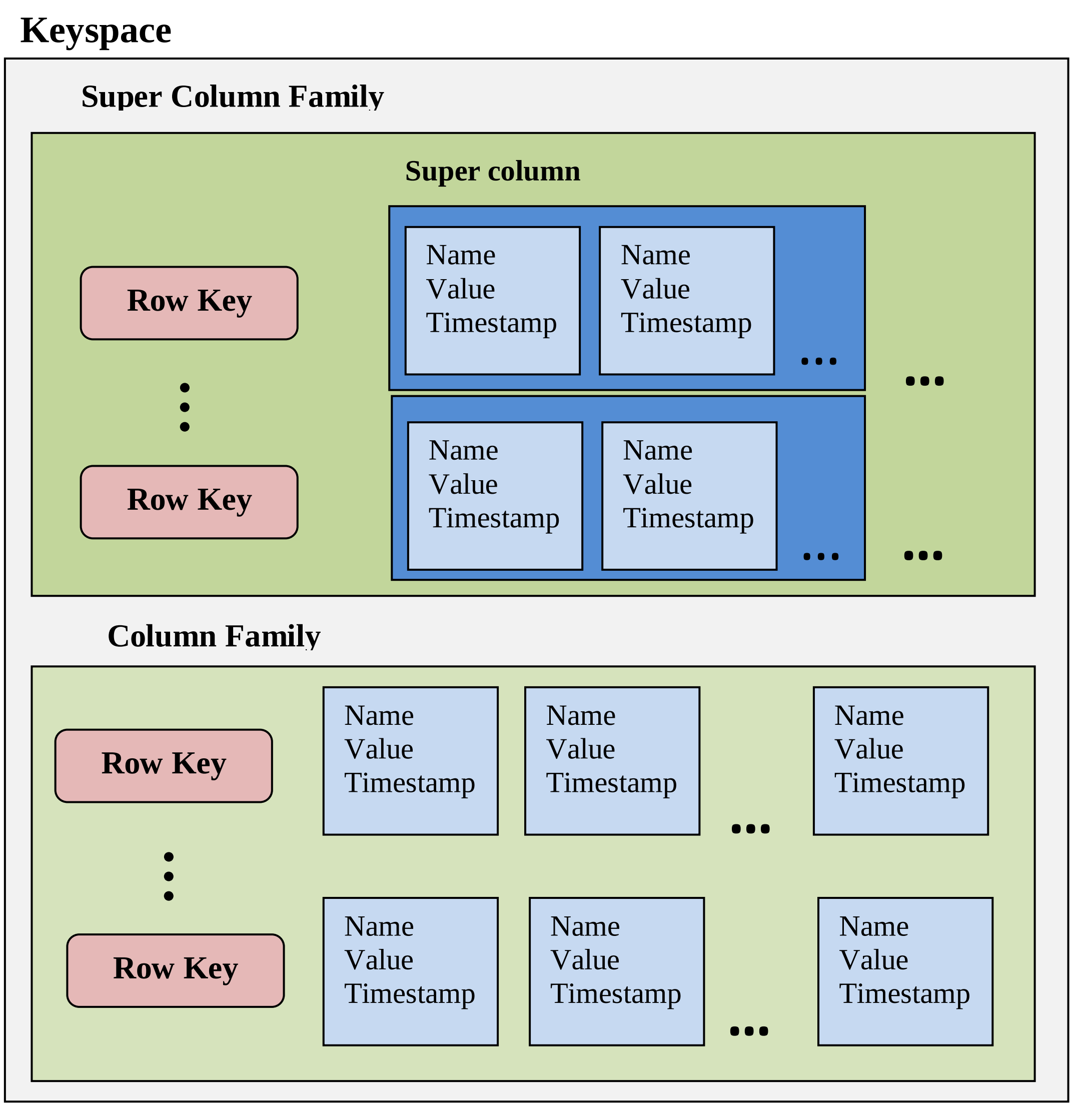

Figure 6: Data model in Apache Cassandra

Figure 6 illustrates an example of a data model in Cassandra. In this example we have:

- **Keyspace** – a wide based namespace containing a set of (super) column families.
- **Super column family** – contains rows as keys and super column as values
    - *Super column* – contains a set of columns.
- **Column family** – contains rows as keys and column as values.

An example in JSON (JavaScript Object Notation):

```
Accounts:                          ColumnFamily
    Administrator:                 RowKey
        Name: Admin                ColumnName:Value
        AccessRights: r/w          ColumnName:Value
        Password: ******           ColunName:Value
    User:                          RowKey
        Name: Guest                ColumnName:Value
        AccessRights: r            ColumnName:Value
```

In the example above we have a column family labeled *Accounts* associated with two rows, *Administrator* and *User*. The Administrator row has three columns while User has only one. In Cassandra that is accepted. Instead of setting the value to *null*, which resource an amount of memory, the null-value is never written.

4.2 Security

Cassandra allows use of an authentication mechanism for configuration of the database. The default authenticator is *AllowAllAutenticator*, which does not require credentials for clients. If you want to configure credentials, another alternative is to use *SimpleAutenticator*. Beside of these two there are options for writing a new authentication mechanism.

The use of the second authenticator allows you to set different accounts for a keyspace. To be able to access data, a user has to authenticate itself by writing a username and password. The passwords that are stored for each username can be MD5 encrypted or saved as plain text.

4.3 Storage

Cassandra uses a different architecture for internal data storage for achieving high performance when making read and write operations. Writes are stored directly in the primary memory (RAM) as *Memtables*. When the size of data in the Memtable grows and reaches a threshold, the data is flushed to the secondary memory as an *SSTable* [6].

	Reading	**Writing**
Apache Cassandra	~ 15ms	~ 0.12ms
MySQL	~ 350ms	~ 300ms

Figure 7: Performance comparison between Cassandra and MySQL [14]

Figure 7 shows a comparison that was made by Apache to test the performance of Cassandra. This test was made with 50GB of data. The reason for the huge difference for Apache Cassandra and MySQL is the features Cassandra offers. Scaling reads to a relational database is difficult because of master and slave properties and scaling writes are virtually impossible, which is mentioned in chapter 3.1.1. Offering features like a write can be done to any node in the cluster and using the storage architecture mentioned below increases the performance drastically.

4.3.1 Storage Architecture

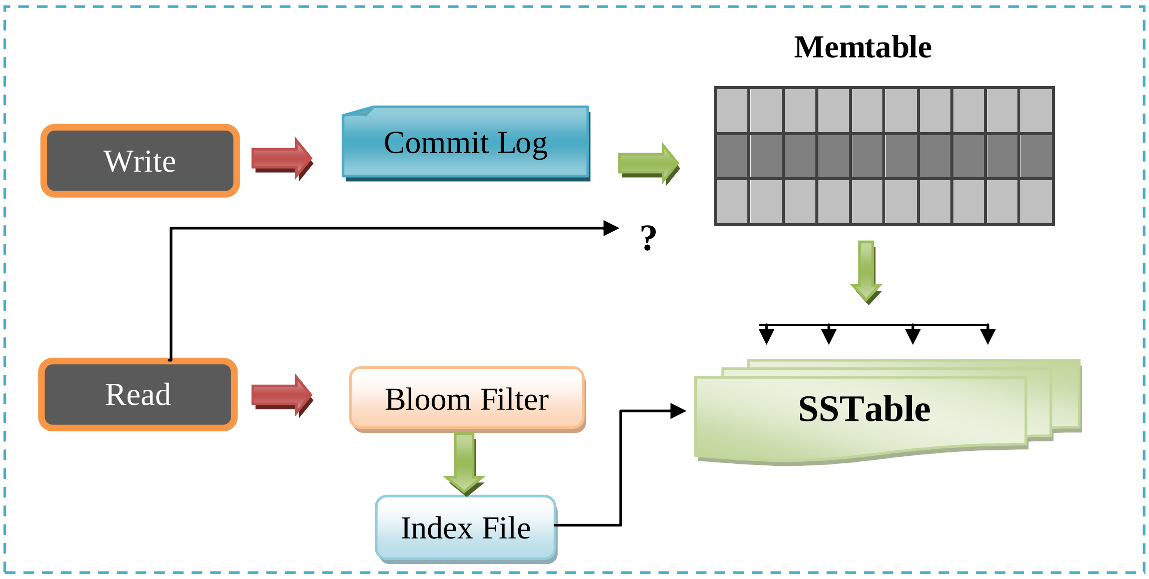

Figure 8: Writing and reading path for Apache Cassandra

Figure 8 shows the steps when performing a:

Write
A write is first saved into the commit log. After a check the data will be stored in a Memtable, and if needed flushed to the SStable on the disk.

Read
The system will start searching for the key in Memtable. If the key does not exist in the Memtable the search will continue looking for the key with the help of Bloom Filter [15], if the answer is false-positive [Chapter 4.3, Filter file] the search will go on to the index file and return the data.

Commit Log
Writes are first written to the commit log to ensure that a write is saved and can be recovered in case of crash, breakdown or inconsistency. Writings will not be counted as successful until they are written into the commit log. All writes are in sequential order giving a fast write instruction, meaning no reads or seeks when writing a value.

Memtable
Memtable is an in-memory database management system and the data is stored as key-value table. Every (super) column family has its own Memtable and the data is sorted by their keys. When a Memtable is full (default threshold value is 128MB), the data is flushed to the disk and stored as an *SSTable*.

SSTable
Sorted String Table (SSTable) is a data structure where the data is stored on the disk. The data is sorted by row-key and column-name. Depending on the amount of data, there can be multiple SSTables per column-family.

SSTable have headers to improve the performance when making a read. These are:

Index file
Index file stores keys of a value, not the actual data. The file is sorted by the key and offsets to where the data for a specific key can be found in an SSTable. The primary purpose of having index file is to speed up the performance of reading. If the key does not exist in the index file then there is no actual data, but if the key exist the location will be known on the SStable.

Filter File

Cassandra uses a filter file called *Bloom Filter* [15] as a performance booster. A Bloom filter is a fast algorithm for testing if an element is a member of a set. When reading a value, the Bloom Filter is checked before accessing the disk. This filter will check if the row key exists in the SSTable without having to read the index file. The return value from the Bloom Filter check is false-positive or false-negative. False-positive indicates that the row key *might* exist in the database, while false-negative confirms that the searched row key is not in the database.

Since it is possible to get false-positive and impossible to get false-negative, the disk is only checked when the Bloom filter indicates that the value *might* be stored.

4.3.2 Complexity

The complexity for using insert (update), read and delete operation depends on how these instructions are implemented and which data structures are used. In this section we will look at the complexity of the data structure which Memtable uses for saving values for the following operations:

Write

All writes are made into the primary memory and stored as Memtables. The data is sorted by their key and the Memtable is implemented [16] as the data structure ConcurrentSkipListMap [12].

Read

There will be key look ups for searching if the key exists in the tables. Using Bloom Filter and Index File improves the performance and indicates if and where a value might be stored.

Using these data structures for storage gives the following *complexity*:

	Big O notation
Insert / Update	O (Log N)
Read	O (Log N)
Remove	O (Log N)

Figure 9: Average cost for get, put and remove operation, N being number of nodes in the ConcurrentSkipListMap data structure

The reason why making a read is O (Log N) is because of the binary search on the in-memory index.

4.4 Clustering

Cassandra uses a peer-to-peer distribution model; meaning any given node in the cluster is structurally identical to any other node, giving no master and slave relationship. If a node is down the data will still be available for querying depending on the replication placement strategy [Chapter 4.4.2] the system uses.

Cassandra is designed to be distributed over several nodes operating together and appear as a single instance for the end user. Running Cassandra on a single node (server) is not the best choice of database storage system. The structure of data management in Cassandra is the cluster, also called a ring.

4.4.1 Gossip

To keep track of nodes in a cluster, Cassandra uses a gossip protocol [2]. When a node is added or there is a change of state (down / up), the gossip protocol is used to inform other nodes in the cluster. This feature makes it easier to scale up and down than in a master/slave replication which MySQL uses since this notification does not have to go through a master node.

Failure Detection

To determine if a node is available for contact, Cassandra uses a modified version of the algorithm Φ Accrual Failure Detection [17]. The algorithm returns a so called *suspicion value* Φ. This value will determine if it is possible to read or write to that node. In this way Cassandra will know if a node is unavailable and out of reach.

Failure Prevention

When a node is flagged down writes will be done to another node in the same range. Once the node is up again the writes that were supposed to be to the node will be redirected. This is called *Hinted Handoff [2]* and is used to guarantee that a node stores the correct data.

4.4.2 Replication placement strategy

Replication strategies determine where the replicas are located in a cluster. We can almost describe the strategies as rules that determine which node is responsible for which key range and which nodes will be responsible for specific queries and where all the replicas for that node are located within the datacenter or within other datacenters.

The predefined strategies are [2]:

Simple Strategy

This strategy is renamed from Rack-Unaware Strategy. The strategy only places replicas in one datacenter, but is unaware in which rack the node is located.

Old Network Topology Strategy

This strategy is also known as Rack Aware strategy. The strategy is aware of the racks that the node is placed in within a datacenter.
This strategy can even place nodes in other datacenter often as replicas for the first datacenter. A simple example using this strategy is two datacenters and three nodes to place. This strategy will place two of the nodes in the first datacenter on available racks and one node as a replica in the second datacenter on available rack.

Network Topology Strategy

This strategy was implemented in the 0.7 version and is almost the same as the Old Network Topology. The difference is that in this strategy you can select in which datacenter you want your replicas and nodes.

4.4.3 Partitioner

The main purpose is to give the option to specify how to sort the row keys on multiple nodes. There are a few ways to partition row keys in Cassandra, and these are explained in this chapter.

Random Partitioner

This partitioner is default in Cassandra. It hashes the row key into a 128-1 bit MD5 hash and saves the token into a Big Integer. This BigInteger-token will then be used to determine the ring node that the keys will be placed within. Using this partitioner, the advantage is when managing the ring and balancing the load of information within the cluster. The disadvantage is the difficulty of making efficient range queries across multiple keys.

Order-preservent partitioners
In this type of partition the keys are on a UTF-8 token (string tokens). When using this partition you need to define the range of keys that the node will accept as shown in the example below.

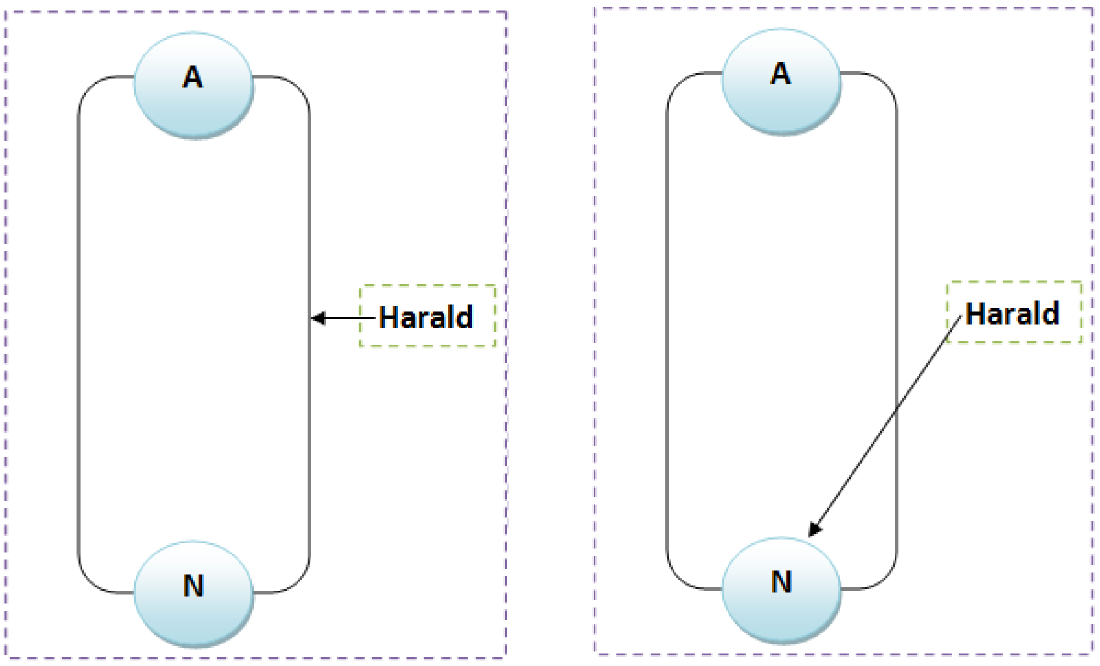

Figure 11: An example of Order-preservent partition

In figure 11 we see a key (Harald) which is going to be inserted to the ring within a node. The first thing the partitioner will do is to look at the defined node ranges that we have. In this example we have a node that has the range up to and including A and one that has up to and including N. So now we check the first letter in the key string, that key will be compared to the nodes ranges, this will tell us that the key would be inserted into the node with the range N.

The disadvantage when using this partitioner is that the cluster has to be balanced manually. As the data and keys increase nodes with the letters Z and Q will rarely be used.This will form a problem in the cluster, having nodes that are overloaded and nodes that are almost empty.

Other Partitioners

Collating Order-Preserving Partitioner
This partition method orders keys according to the United States English locale (EN_US). This partitioner is rarely used and is not an extension of the Order-preservent partitioner.

Byte-Ordered Partitioner
In this partitioner the keys are kept in raw bytes instead of converting them into strings as the Collating order-preserving partitioner and the Order-Preserving partioner. The byte order partitioner is almost only used for performance optimizing.

4.5 Consistency Level

Consistency level is a setting that allows deciding how many *replicas* a cluster has to respond to, for a read or has to acknowledge for a write operation before the queries are considered successful. When setting the consistency level, it is crucial that it is based on the replication factor. As the consistency level is set higher, the demand of more nodes responding for a query grows. This gives the assurance that the present value is the same on each replica. If the values from each node are different, it will return the newest updated and reply that to the client. This is done by comparing timestamps in the columns. After it made its reply, Cassandra will perform a so called *"read repair"* in the background. The read repair will take the newest updated value and write over the old values with the new. In this way the data will always be consistent.

Consistency level is specified *per* query, by the client and is not keyspace-wide like replication factor.

4.5.1 Write operation

When making a write operation the following consistency levels can be specified in the client:

ZERO

Consistency level *ZERO* does not give guarantees of a successful write operation. Write operation will return immediately to the client before the value is recorded.

ANY

This consistency level writes the value to at least one node, allowing hints (*Hinted Handoff*) to count as a write.

ONE

The write is at least written to the commit log and memtable of one node before returning to the client.

QUORUM

Majority ((*replication factor/2) + 1*) of replicas should receive the write to be counted as successful.

ALL

All nodes that are specified by replication factor must receive the write before returning to client. No hints allowed, even if a single node is unreachable the write operation will fail.

4.5.2 Read operation
When making a read operation the following consistency levels can be specified in the client:

ONE
The first node that responds to the query immediately returns the value held by it. A read repair operation is created in the background and checks for out dated values against the returned value.

QUORUM
Query all nodes and when the majority ((*replication factor/2) + 1*) of replicas respond, return the latest value. Perform a *read repair* in the background to keep up the consistency.

ALL
Query all nodes and wait for all nodes to respond and return the latest value. If needed performs a read repair.

4.5.3 Choosing consistency level
Consistency level is defined with the query by the client and should be chosen depending on the value of replication factor defined in the keyspace. According to the CAP theorem, Cassandra chooses AP (availability, partition tolerance) before C (consistency). This is not entirely the truth. Cassandra is *'eventual-consistent'* meaning there is a choice on how to achieve strong consistency in cost of availably and partition tolerance. This feature is given by letting the user to choose consistency level for queries.

If a system requires strong consistency it can be achieved by this inequality:

$R + W > N$,	R: Number of records to be read.
	W: Number of records to write.
	N: Replication factor.

Pone that there are three nodes in a Cassandra cluster and the replication factor is set to two. For achieving high consistent data the consistency level most be *QUORUM* for read and write operations.

Calculating amount of replicas for read and write (QUORUM):

$$\boldsymbol{R} = \frac{2}{2} + 1 = 2 \quad \boldsymbol{W} = \frac{2}{2} + 1 = 2\text{ , }\mathbf{N = 2}$$

➔ **R + W > N = 2 + 2 > 2** gives strong consistency.

5 Development

In this chapter we will describe how to use the Apache Cassandra server, which clients are available for developing and how we wrote the Cassandra API. The tests are mainly made on Windows XP, Windows 7 and Ubuntu 11.04.

5.1 Installing Apache Cassandra

Apache Cassandra is available for download at address: http://cassandra.apache.org/download. Download the gzipped file named *apache-cassandra-x-x.x-bin.tar.gz* where x.x,x stands for the current Cassandra version number. The download size is about 9MB.

Apache Cassandra can be used in operative systems such as Windows, Linux and Mac OS.

The steps for starting the Cassandra server for the first time in a Windows 7 machine:

1. Download the binary file.
2. Extract it to your home directory.
3. Set up the environment variable.
    a. Click on "Start" →Right click on "Computer" →"Properties" →"Advanced System Settings" → "Environment Variables" → "Create a new system variable". In value, type in the path of Apache Cassandra directory and in the variable name filed, write "CASSANDRA_HOME". Make sure you have set environment variables for "JAVA_HOME".
4. Start command window
    a. Type *"%Cassandra_home_directory%/bin/Cassandra*", the server is up and running and a log will be printed.

If the installation is successful, the server will be "*listening for thrift clients…*"

When the Cassandra server will be up running for the first time with the standard configurations it will be a single node cluster named *"Test Cluster"* and this will be listening on the port 9160. The clusters configuration can be changed in *"%Cassandra_home_directory%/conf/Cassandra.yaml.*

Description	Port
Thrift protocol – client traffic	9160
Gossip protocol – cluster traffic	7000
Cluster monitoring via JMX	8080

Figure 10: Ports that the Cassandra server uses

5.2 Clients – Ways to access Cassandra

There are several clients available for accessing a Cassandra server. All of them are more or less based on using the thrift interface. Cassandra's command client *CLI* is entirely written in the Thrift API. CLI is useful for defining keyspaces, column families and columns. The CLI client is not suitable when defining consistency level for a read or write because it is unstable and is generally buggy.

5.2.1 Thrift

Thrift is a framework and a set of a code-generator tools for scalable cross-language service development. Thrift was originally developed at Facebook and was open sourced in April 2007. Thrift works as a code generation library for other programming language clients. The goal with this is to support efficient remote procedure calls (RPC) in other programming languages as easy as possible.

5.2.2 Avro

Avro is a data serialization system and a remote procedure call (RPC) system [20], and is an alternative to Thrift. There are many features that are provided in Avro similar to those of Thrift such as data serialization and RPC mechanisms. The difference from Thrift is that Avro is a dynamic data serialization library that does not require static code generation when using RPC for applications. Another different aspect is that the serialization is compact and efficient in Avro because the definitions are written as schemas using JSON. This means that when data is used, schemas are always presented along.

High level Clients

There are other high level clients that can be used to communicate with the Cassandra server. These clients are recommended to be used instead of raw Thrift when developing applications. The purpose of Thrift is primarily for client developers. See Appendix C for available high level clients.

5.3 Getting started with Cassandra

To start the client open a new terminal window and navigate to the *<Cassandra-directory>/bin* to run the client type in *Cassandra-CLI*. This will start the client. When the client is up and running it will display a welcome sign and after that sign you will have an interactive shell that you can issue commands in.

5.3.1 Basic terminal client commands

Using the data model in Appendix B as an example for how to insert/delete with the Cassandra.CLI client.

Insert

```
set Movies['Inception']['Genre'] = 'Mystery';
set Movies['Inception']['Year'] = '2010';
set Movies['Inception']['Length'] = '148';
```

Get with primary index search

```
get Movies['Inception'];
=>
(column=Genre,value = Mystery,timestamp = 2011032904612000)
(column=Year,value = 2010,timestamp = 2011032904315000)
(column=Length,value = 148,timestamp = 2011032904119000)
```

Get with secondary index search

```
get Movies where year = '2010';
Row_Key: Inception
=>
(column=Genre,value = Mystery,timestamp = 2011032904612000)
(column=Year,value = 2010,timestamp = 2011032904315000)
(column=Length,value = 148,timestamp = 2011032904119000)
```

Secondary index search is only available in version 0.7 and the latest version.

Deleting a column

```
del Movies['Inception']['year'];
column removed
```

Updating a column value is done simply by making an insert in the column and the value will get updated.

5.4 Data model design

This chapter will describe briefly how to design a database model in MySQL, and how to convert from MySQL to Apache Cassandra's data model.

5.4.1 Designing a MySQL data model

When starting to build a data-driven application with RDBMS you might start with modeling the domain as normalized tables and make references to related data in other tables by using foreign keys. This is called relation modeling and by that it means that we start from the domain and then represent the nouns in the domain into tables. Thereafter we assign our primary and foreign keys by looking into our relationships between the tables. When finding a many-to-many relationship we create a join table that has those two keys represented. Thereafter by using the defined keys we put together different data by writing our queries.

5.4.2 Converting from MySQL to Apache Cassandra

When converting from a relational data model to Cassandra data model these steps should be followed:

- Convert the tables that only contain a primary key to a column family. The table's name becomes the column family name and the primary key as the row key in the column family. The attributes in the table becomes the columns in the column family.

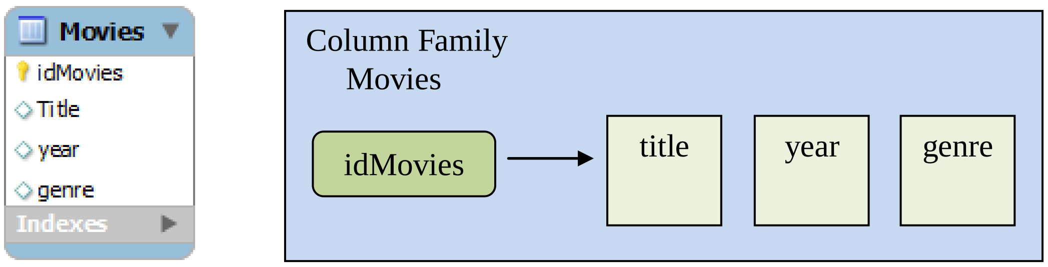

Figure 12: Converting Example of a table to Column Family

- A table that contains a primary key and a foreign key should be converted to a super column family. The table's name becomes the super column family name, the foreign key as the row key and the primary key as the super column name. Attributes of the table will be values (columns) of a super column.

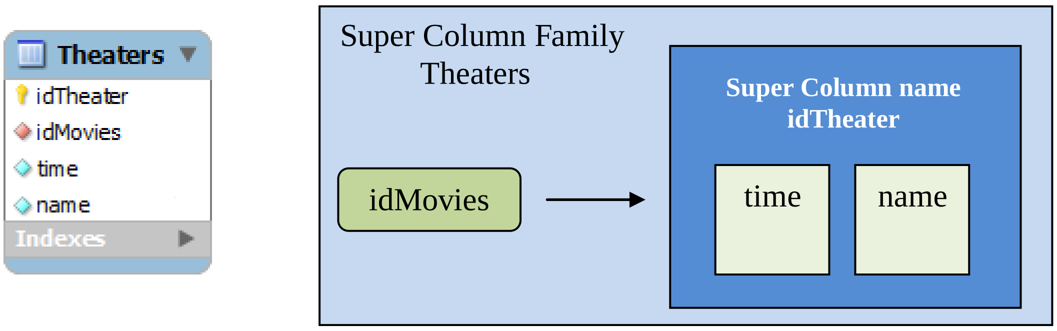

Figure 13: Converting example of a table to Super Column Family

- The last step before having a complete conversion of the MySQL data model is to write down the queries that the system will use. When having the queries figured out, check the queries against the converted data model. If there are any queries that does not match the converted data model, re-construct the data model according to the query.

The most important aspect when converting the MySQL tables to Cassandra's column families is to look at the most used queries. In this way, the database will be optimized and that in turn will give a better performance.

5.5 Web API

The Apache Cassandra database server needs a client for communication with the Adengi webpage. The available high level clients are listed in Appendix C.

Since the webpage was implemented in PHP, it was decided to use the PHP client, *PHPcassa* to write the API.

5.5.1 Adengi Cassandra API

When implementing the API for communication with Adengi database server, we first looked at the data model and then constructed a PHP-class. This PHP-class will be used to create an instance object and the use the function for making different operations on the server.

Example on how to create a connection to the database:

```
public function __construct($host, $keyspace){
          try{
                    $this -> pool = new
                    ConnectionPool($keyspace, array($host));
          }catch(Exception $e){
                    echo "Failed to connect to database. ".$e-
                    >getMessage();
                    return;
          }
          echo "Connection to Cassandra server established";
}
```

In this constructor we used the PHPcassa function *ConnectionPool.* There can be several connections established and the amount can be set directly when calling the function.

```
/*
* Example on how to create an account in Adengi
*/
public function ad_account($userId, $accountame){
          $key = uuid::mint(1);
          $column_family = new ColumnFamily($this ->
          pool,’Column family Name’);
//first key is the name of the super column. The array
//contains columns in the super column
          $arr = array($key->__toString() =>
          array('account_name' => $accountname));
          $column_family -> insert($userId, $arr);
}

public function get_user($username, $password){
          $myusername =strtolower($username);
          $mypassword =$password;
          $index_exp =
          CassandraUtil::create_index_expression('username',
          $username);
          $index_clause =
          CassandraUtil::create_index_clause(array($index_exp));
          $rows = $this ->Column_family_Name->
          get_indexed_slices($index_clause);

          $resultUsername = "";
          $resultPassword = "";

          foreach($rows as $key => $columns){
                    $resultUsername = ($columns['username']);
                    $resultPassword = ($columns['password']);
          }
          if($username == $resultUsername && $password ==
          $resultPassword){
                    return true;
          }else{
                    return false;
          }
}
```

The super column name is of type UUID (Universally Unique Identifier) for a specific account. The account’s information, such as name, date and value is stored first in an array and then inserted in the super column family. Retrieving a user from the database is done by using a secondary index slice. In this way the whole column will be fetched.

6 Test

Different test scenarios were created to analyze how to set up a cluster with multiple nodes and how these nodes would respond in case of a breakdown.

The test scenarios and their results are described in this chapter.

6.1 Clustering with switch

Scope
In this scenario, two Windows 7 machines were set up, connected by a switch.

The purpose of this test was to:

- Test how easy it is to assign nodes their specific *token* values manually.
- Examine the behavior of the nodes when changing the *replication factor* and the *consistency level*.
- Taking down a node and try to write to the cluster.

The token values were calculated by the following equation:

$$Token\ value\ for\ node\ n_i = \frac{2^{127}}{N} \cdot n_i$$

$$N = number\ of\ nodes\ in\ the\ cluster$$

$$i = 0,1,2,3 \ldots N-1$$

Result
Assigning a node a specific token-range using the Random Partitioner strategy [chapter 4.7.1], can be done by editing in the *config.yaml* file. This will change the node's token if the node is running for the first time. Moving a node's token value can be done using *Nodetool*,which comes with the Apache Cassandra package.

Changing the replication factor to two saved redundant copies of the data. Trying to change to a higher number than two would give errors when writing a value, which depended upon inconsistency.

Taking down a node and then trying to write a value could only be done when choosing the consistency level ONE. When the node was up again, the value would be replicated.

6.2 Clustering in different datacenters

Scope
In this scenario, we set up one Windows 7 machine and one Windows XP machine. Each machine was in different geographical location and behind a router.

The test’s purpose was:

- Understanding how to set up Cassandra when using routers.
- Try to connect to the Cassandra node using the client over the Internet
- Examine if the Cassandra nodes could communicate over the Internet

To communicate with the nodes behind router the port for Thrift, Gossip and JMX needed to be open. The ports that Cassandra uses are describe in chapter [5.7, figure 10].

Result
Accessing the Cassandra server over the internet could be done using the command CLI or any other high-level clients. Trying to monitor a node behind a router could not be done since the listen address was always set to the private IP. This could only be changed in the Cassandra source code. Neither could the nodes connect to each other in the cluster.

6.3 Clustering with VPN tunnels

Scope
In this scenario, we set up two Windows 7 machines and one Windows XP machine. Each machine was in different geographical location and behind a router. The Windows XP machine was running as a VPN server, and the other two machines as VPN clients connecting to the server.

The purpose of test was:

- To find a way to try to monitor a node behind a router over the internet without changing the source code of Cassandra.
- Testing to add a third node to the cluster.

Result
Cassandra could still be accessed by command clients and other high-level clients. The only difference was that it was possible to monitor a node behind a router over the internet. However, it was not possible to add a third node to the cluster. This error is not because of the database itself, but of the operating system running on the server.

7 Conclusion

7.1 Result

In the analysis we examined the drawbacks of a relational database and which databases that could replace it. We came to the conclusion that converting the current MySQL data model to Apache Cassandra data model would increase the availability and performance of the system and give an efficient scalable database system. Re-designing the MySQL data model to Apache Cassandra's data model went on successfully by following the steps described in chapter [5.4.2].

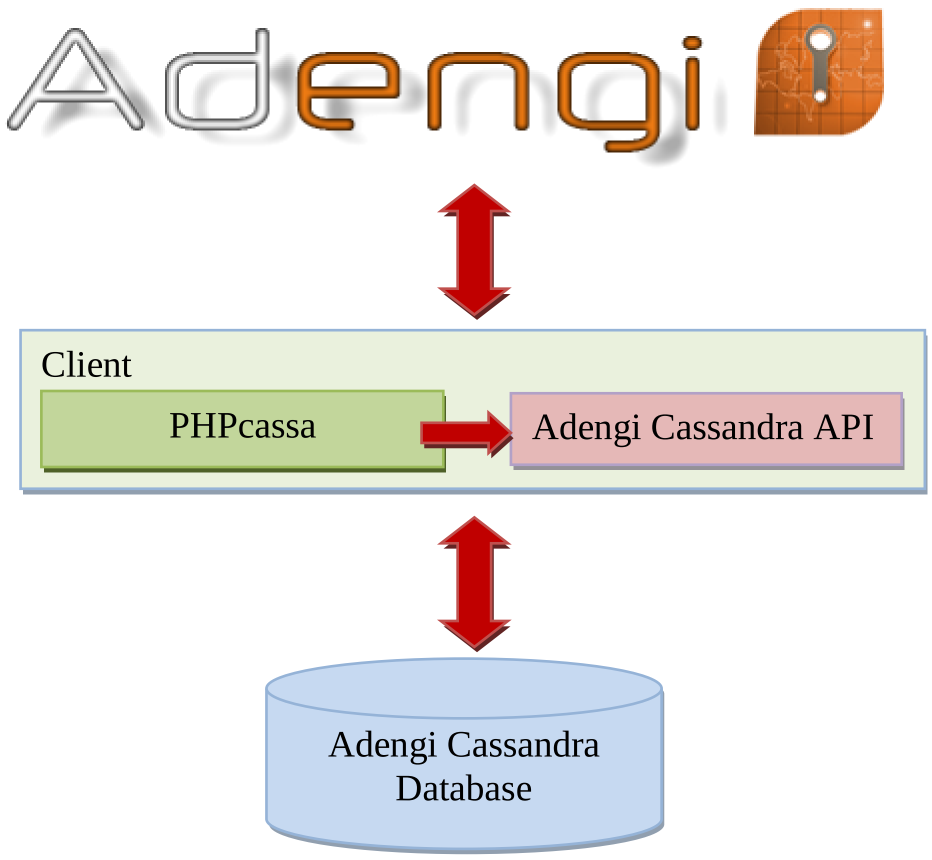

Figure 12: Overview of the Adengi System

The final version of the system consists of the following parts:

Adengi webpage
: The graphical user interface for Adengi users for advertising and controlling their ads.

Client
: The PHPCassa and Adengi Cassandra API client written in PHP for communication with the webpage and Cassandra database.

Adengi Cassandra Database
: The Cassandra database server that is based on the earlier MySQL database with a few adjustments.

7.2 Discussion

7.2.1 Apache Cassandra

Working with Apache Cassandra was a completely different approach than designing a relational database. Instead of focusing on designing a model that was 40 years old, the engineers behind Cassandra looked at the existing problems that were present when designing Cassandra. The big problem was how to design a highly available, scalable and distributed database. This is the main reason for developing a database like Cassandra.

Scalability strategies like sharding and replication is built into Cassandra. Both of these strategies are hard and nearly not possible to implement in a relational database because of the CAP theorem. Since most RDBMS focus on CA (Consistency, Availability) and wanting to add new servers, preferring P (Partition tolerance) would give the loss of C or A. Even if Apache Cassandra is not designed to have C, it can be achieved in expense of response time by applying the feature consistency level to Cassandra.

The core difference between a RDBMS and Cassandra is how to design the data model. In RDBMS the focus is on relations, tables and how to normalize them in order to avoid redundant copies. In Cassandra the main focus is on how the system is built, which queries will be the most used and how the data can be designed to achieve a high availability and efficiency.

One of the big advantages with Cassandra is that it is easy to change the data model or add new column families without normalizing. We had a big advantage of this when designing the model for Adengi. When creating the column families for a specific table we had to change the model when implementing the Adengi Cassandra API, since Cassandra data model is query-based, we had to make some minor changes.

7.2.2 Test

Clustering with switch
This test gave us almost all information about how the replication goes on in Cassandra. This test also made us understand how to calculate token-values and how to set these values right. We had some problems after this test when running our machines independently, this because when clustering we had set the consistency to a higher level than it should be. This consistency level needed both the machines to respond to a query before the returned value counted as a valid return. So when we queried the database it started to give us exceptions messages that the information within the cluster was not consistent.

Clustering in different datacenters
Making Cassandra communicate over internet was not as easy as we thought it would be. Cassandra was implemented in such way that it can only listen to private and not public IP-addresses. Because of this we could not make Cassandra to communicate over the internet. To fix this problem we would have to make some changes in the source code of Cassandra. These changes would have made Cassandra listen to both private and public IP-Addresses. At first we thought that this problem was caused by the ports that would forward the communication from the router to the machines, but as we made research we found out that this was a common issue caused in Cassandra.

Clustering with VPN tunnels
Using the VPN tunnels made the Cassandra nodes connect and communicate with each other. This because of when the client nodes where connected to the VPN server they became members of the server nodes private network. This was solved because when the nodes are within the same private network the problem with having a public IP-address as a listen address disappeared. Using Windows XP operating system in the server with the VPN connection was not good. This gave us a limitation that we did not expect. The limitation was that we could not connect two or more computers to the VPN server at once. Still even if we could not connect more than two nodes together we called this test as a success because this was the first time we could make nodes communicate over the internet.

7.3 Future work

There are many improvements that can be done to the Cassandra database and Adengi. Some of the improvements / ideas are discussed in this chapter.

7.3.1 Cloud Solution

Since most webhosting sites do not support Apache Cassandra, the only way to host Cassandra database servers globally is by using a cloud solution. Amazons cloud solution EC2 (Elastic Compute Cloud) are used by most Apache Cassandra users with good results. The next step for the Adengi system is to use a cloud solution like EC2. This will give them the control of their server without any restrictions and with good monitoring tools besides JMX.

7.3.2 Automatic MySQL to Apache Cassandra data model

Another idea that came while researching for Cassandra was to make a generic automatic MySQL schema converter to Apache Cassandra data model. The main idea would be to use the steps in [Chapter 5.4.2] and implement a script. Using the .sql file from MySQL database and convert the tables into Cassandra’s column families / super column families.

7.3.3 Database for storing statistics

Adding a feature for storing statistics for a certain campaign would make Adengi more user-friendly and useable for advertisers. Cassandra database would fit perfectly for storing that amount of data. The Adengi Cassandra database would be a good place to implement needed column families for storing statistics. The simplest way would be to add the structures according to the queries that would be used for statistics.

7.3.4 Automatic token calculator

There is a load balancing problem that occurs every time adding or withdrawing a node from the cluster. The cause of the problem is the calculation of the nodes token-values. These token-values are dependent on the numbers of the nodes within the cluster. When adding or withdrawing a node, recalculation of the token-value for each node in the cluster is needed. The node calculation is done so the load will be distributed evenly to the nodes. The calculation and the token distribution to the nodes are done manually. Instead of calculating and giving each node its specific token manually, a script that could calculate the tokens for every node in the cluster would facilitate the balancing of the cluster when adding or withdrawing nodes from it.

8 Dictionary

ACID	A set of properties (Atomic, Consistent, Isolated, Durable) that guarantees database transactions.
API	Application Programming Interface
BASE	A set of properties (Basic Availability, Soft-State, Eventual-Consistent) that guarantees database transactions.
Bloom Filter	Is a fast algorithm for testing in an element is a member of a set.
Cluster	A group of linked machines operating together and appearing as a single server to the end user.
False-negative	If the Bloom filter indicates that the key does not exist in the database but in fact it does.
False-positive	If the Bloom Filter indicates that the key exists in the database but in fact it doesn't.
Foreign Key	Is key field in a relational table that machines a candidate key of another table.
JMX	Is an API for management and monitoring of applications, devices, services and the JVM.
JSON	JavaScript Object Notation is derived from JavaScript scripting language for representing simple data structures and associative arrays, called object.
JVM	Java Virtual Machine.
MD5	Is a 128 bit cryptographic hash function.
Normalization	Database normalization is the process of organizing data to minimize redundancy.
NoSQL	Database management systems that differs from classic RDBMS.

Primary Key	Is a unique that identifies a row in a table.
RAM	Random Access Memory.
RDBMS	Relational Database Management System
Replication Factor	This factor determines how many copies of a data that should be stored in a Cassandra cluster.
RPC	Remote Procedure Call allows computer programs to cause a procedure to execute in another computer on a shared network.
Token	A value that are assigned to a node. This value determines the range of the keys to be stored in a specific node.
Tuple	An ordered list of elements.
UTF-8	Transformation Format – 8 bit is a multi byte character encoding for Unicode.
VPN	Virtual Private Network

9 References

[1] *Marcom Professional* http://www.marcomprofessional.com/posts/andrew.grill/mobile-advertising-bucks-downward-trend-reaching-5.7bn-by-2014-says-juniper (March 2011)

[2] Eben Hewitt, *Cassandra The Definitive Guide*, ISBN 978-1-449-390-41-9, 2011, O'Really Media

[3] Avinash Lakshman, Prashant Malik, *Cassandra – A Decentralized Structured Storage System*, http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf (March 2011)

[4] *Apache Incubator project* http://www.incubator.apache.org/ (March 2011)

[5] *Apache Cassandra wiki* http://wiki.apache.org/cassandra/DataModel (March 2011)

[6] *Datastax Cassandra Developer Center* http://www.datastax.com/docs/0.7/configuration/storage_configuration#id1 (March 2011)

[7] *The Neo Database*, page 4 http://dist.neo4j.org/neo-technology-introduction.pdf (April 2011)

[8] Kristina Chodorow, Michael Dirolf, *MongoDB The Definitive Guide*, ISBN: 978-1-449-38156-1

[9] *MongoDB homepage* http://www.mongodb.org (April 2011)

[10] *MongoDB homepage*, API http://api.mongodb.org/java/current/com/mongodb/DBCollection.html

[11] *MySQL performance blog* http://www.mysqlperformanceblog.com/2009/08/06/why-you-dont-want-to-shard/ (April 2011)

[12] *Oracle homepage* http://download.oracle.com/javase/6/docs/api/java/util/concurrent/ConcurrentSkipListMap.html (April 2011)

[13] *Apache Cassandra wiki* http://wiki.apache.org/cassandra/FileFormatDesignDoc (April 2011)

[14] J Ellias OSCON 09 Presentation, page 26 http://assets.en.oreilly.com/1/event/27/Cassandra_%20Open%20Source%20Bigtable%20+%20Dynamo%20Presentation.pdf

[15] *Apache Cassandra Developer blog, Bloomfilter* http://spyced.blogspot.com/2009/01/all-you-ever-wanted-to-know-about.html (April 2011)

[16] *Apache Cassandra Svn page*
https://svn.apache.org/repos/asf/cassandra/trunk/src/java/org/apache/cassandra/db/Memtable.java (April 2011)

[17] Naohiro Hayashibara , Xavier Defago , Rami Yared , Takuya Katayama. (2004).*The Φ Accrual Failure Detector. Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems* (SRDS'04), p.66-78. (April 2011)

[18] *Apache Cassandra wiki*
http://wiki.apache.org/cassandra/ClientOptions. (May 2011)

[20] *Apache Avro*
http://avro.apache.org/docs/current/ (May 2011)

[21] *VSChart homepage*
http://vschart.com/compare/apache-cassandra/vs/neo4j (May 2011)

[22] *VSChart homepage*
http://vschart.com/compare/apache-cassandra/vs/mongodb (May 2011)

[23] *Datastax Cassandra Developer Center*
http://www.datastax.com/dev/blog/why-does-scalability-matter-and-how-does-cassandra-scale (April 2011)

[24] *Brewer's Cap Theorem*
http://www.julianbrowne.com/article/viewer/brewers-cap-theorem (April 2011)

[25] *High Scalability*
http://highscalability.com/unorthodox-approach-database-design-coming-shard (May 2011)

[26] About Database
http://databases.about.com/od/specificproducts/a/normalization.htm (June 2011)

10 Appendix A

Comparing Apache Cassandra, Neo4J and MongoDb [24] [25].

	Cassandra	**Neo4J**	**MongoDB**
Feature			
Query Language	API Calls	API calls, REST, SparQL	JavaScript, API calls, JSON
Database model	Column-oriented	Graph-oriented	Document-oriented
Map and reduce	Yes	No	Yes
Unicode	Yes	Yes	Yes
Full text search	No	Yes	No
Integrity			
Integrity model		ACID	BASE
Atomicity	Yes	Yes	Yes
Consistency	Yes	Yes	Yes
Isolation	No	Yes	Yes
Durability	Yes	Yes	Yes
Transactions	No	Yes	No
Referential integrity	No	No	No
Revision Control	No	No	No
Distribution			
Horizontal scalable	Yes	Yes	Yes
Replication	Yes	Yes	Yes
Sharding	Yes	Yes	Yes
System Requirement			
Operating system	Cross-platform	Cross-platform	Cross-platform
Architecture			
Programming language	Java	Java	C++

11 Appendix B

Schema Example

```
/*This file contains an example Keyspace (Theater) that
can be created using the
cassandra-cli command line interface as follows.

bin/cassandra-cli -host localhost --file conf/schema.txt

The cassandra-cli includes online help that explains the
statements below. You can
accessed the help without connecting to a running
cassandra instance by starting the
client and typing "help;"
*/


create keyspace Theater
    with replication_factor = 1
    and placement_strategy =
'org.apache.cassandra.locator.SimpleStrategy';

use Theater;

create column family Movies with comparator = UTF8Type
    and column_metadata = [
          {column_name: Genre, validation_class: UTF8Type,
          index_type: KEYS},
          {column_name: YEAR, validation_class: Integer,
          index_type: KEYS},
          {column_name: Length, validation_class: Long,
          index_type: KEYS}
          ]
          and comment = 'Column Family Movies containing three
          columns';
```

Data Structure

In this example, the rows are sorted by type “UFT8Type” and columns by “TimeUUIDType”.

Column Family ”Movies”

The Matrix

Name	Value	Timestamp
Genre	Sci-Fi	2011032904512000
Year	1999	2011032904433000
Length	136	2011032904419000

Inserting a new movie in “Movies”, gives the following structure

Column Family ”Movies”

Inception

Name	Value	Timestamp
Genre	Mystery	2011032904612000
Year	2010	2011032904315000
Length	148	2011032904119000

The Matrix

Name	Value	Timestamp
Genre	Sci-Fi	2011032904512000
Year	1999	2011032904433000
Length	136	2011032904419000

Updating a value in a column, e.g. length column in row The Matrix, setting value to “90”.

Name	Value	Timestamp
Length	90	2011032904912000
Genre	Sci-Fi	2011032904512000
Year	1999	2011032904433000

Deleting the value of a column

Name	Value	Timestamp
Length	90	2011032904912000
Genre	Sci-Fi	2011032904512000

Deleting a value removes the entire column. Saving *null* is value takes memory space and is not efficient.

12 Appendix C

High Level Clients API can be found here [21].

Java

Datanucleus JDO
Hector
Kundera
Pelops

Python

Telephus
Pycassa

Grails

Grails-Cassandra

.NET

Aquiles
FluentCassandra

Ruby

Cassandra

PHP

PHPcassa
SimpleCassie