Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datavetenskap
2019 | LIU-IDA/LITH-EX-A--19/026--SE

Evaluation of Couchbase As a Tool to Solve a Scalability Problem with Shared Geographical Objects

Utvärdering av Couchbase som ett verktyg för att lösa ett skalbarhetsproblem med delade geografiska objekt

Fredrik Wallström
George Yildiz

Supervisor : Jonas Wallgren
Examiner : Christer Bäckström

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Sharing a large amount of data between many mobile devices can lead to scalability problems. One of these scalability problems is that the data becomes too large to store on mobile devices and that many updates are sent to each device. In this thesis, Couchbase is evaluated as a tool to solve this problem where the data has a geographical position. The scalability problem is solved by partitioning the data with the help of Couchbase channels and Google's tile-based mapping system. Synchronising and storing only data of interest for each user has been in focus. The result showed that it was effective to use a Couchbase solution together with Google's tile-based mapping system to reduce the amount of data that was required to be stored for each user. It was shown to be more effective to store objects encoded as base64 data instead of their binary data representation for the data set used in this study. The reason for this is because Couchbase stores Binary Large Objects (BLOBs) as separate files and the BLOBs in the data set had much smaller file size than what the disk sector size was. A test to find how the synchronisation time was affected by the number of channels was conducted. It showed that the synchronisation time increased linearly with an increasing number of channels when the objects were stored in separate files. When the objects were encoded as base64 data, the number of channels used had a minor effect on the synchronisation time. The conclusion is that the approach presented in this study has been effective. However, the results are data dependent and therefore it is recommended to rerun similar tests in order to decide the number of channels to use when partitioning the data.

Acknowledgments

We would like to thank IT-bolaget Per & Per for supporting us during this thesis work. A special thanks to **Per Gustås**, **Jonas Bromö**, and **Joakim Andersson** at IT-bolaget Per & Per that has guided us during the study. We would also like to thank our supervisor **Jonas Wallgren** and our examiner **Christer Bäckström** at Linköping University for their valuable feedback. Finally, a thank you to **Yousif Touma** for the feedback that improved the quality of the report.

Contents

List of Figures

List of Tables

Acronyms

BLOB Binary Large Object

RBAC Role Based Access Control

GUI Graphical User Interface

JSON JavaScript Object Notation

GeoJSON Geographic JavaScript Object Notation

RDBMS Relational Database Management System

GIS Geographical Information System

vBucket Virtual Bucket

ISDB Intermittently Synchronised Database

IoT Internet of Things

1 Introduction

Smartphones are widely used in our society and they are used for many different applications. After 2012, there were more than 1 billion smartphone owners in the world [24]. With faster devices in our pockets the demand for better performance follows. Aijaz et al. [4] describe in their article that the growth of data traffic is significant. The authors mention that offloading data over cellular networks by using other techniques will become an important factor in the industry. However, only changing network technique to transfer data is not sufficient. It is important that developers of applications for smartphones take data traffic into consideration. This should include how much data is sent and how often.

Many applications, like social media, are based on interaction between multiple users where they share data. This leads to the challenge of synchronising data between all users. McCormick et al. [28] discuss the importance of synchronisation of data for mobile applications. They argue that it is important because in some applications, data is useless if it is not synchronised correctly and up to date with the server or other mobile users. With many users using an application where they share data, scalability can become an issue. The scalability problem can occur when there are many users accessing the server or when there is too much data to synchronise.

Siau et al. [35] present some limitations for mobile devices, which are affected by how synchronisation is managed. One major limitation presented is the battery life. Sending data to and from a smartphone increases battery consumption. Another limitation is the fact that many users have a data plan from their mobile carrier that limits the amount of traffic users can consume per month. Lee et al. [26] say that it was predicted that by 2014, the average users would use around 7 GB of data per month. These limitations imply that the amount of data that is synchronised should be reduced as much as possible without affecting the functionality.

The limitations of mobile devices and demands of what applications should be able to deliver together with the importance of synchronisation leads to the challenge of efficiently synchronising data. Some of the problems with data synchronisation mentioned by Agarwal et al. [2] are conflict detection, conflict resolving, and propagating updates to other devices. The au-

thors also mention that keeping data consistency between all devices is difficult when many users are involved.

This thesis will evaluate whether Couchbase, see sections 2.5, 2.6, and 2.7, is a suitable tool for an approach to solve a scalability problem. The approach is intended for applications that share large amount of data between users on mobile devices. The data that is shared must have geographical positions in order for the approach to work. Couchbase is a NoSQL[1] document-based database with synchronisation mechanisms that is built to be easy to use, fast to use in terms of accessing data and easy to expand with more servers [13]. The main reason for using Couchbase was because the company that this thesis was done at had a wish to look into it.

The approach is to split the data into multiple data sets through geographical positions. This has been done by storing everything in one database and logically dividing the data by using channels in Couchbase, see section 2.7.2. By having multiple data sets, the users can subscribe to different data sets and receive information of interest. The splitting was done by dividing a map into several regions. This proposes several challenges like handling objects that cross several regions, how many regions to have, and how the borders of each region should be defined.

1.1 Motivation

This thesis is done at IT-bolaget Per & Per, a company in Sweden. They work with mobile solutions for the forestry industry. Its customers are located in Sweden and Norway. Their applications allow users to interact with maps over Sweden and Norway. For example, a user can draw an area on the map and mark it as "this area has damaged trees". The drawn area is represented as a geometry and saved as an object in the database. These objects are synchronised with all other users, connected to the same system, as soon as possible to ensure that all users have the latest data.

The problem that the company faces is mainly scalability which is related to synchronisation in this case. There are tens of thousands of users that use their application to manage their lands and forests. The systems developed are used by organisations that work in different parts of Sweden and Norway. For example, there is one system for an organisation with all its users located in the south part of Sweden. With the company's current solution, the data these users share is stored in a single database and the entire database is replicated to each user's device. This means that each update made by any user connected to the same system will propagate to every other user. Most of the time, a single user is interested in a small region of Sweden. Since there are many users in each system, the database stored in the users' devices is becoming quite large. Every user device receives a lot of data because of the changes that are done each day.

The company has implemented their own synchronisation mechanism in their application and they have had some problems with synchronising the data between devices. They have had to handle many corner cases during maintenance. Evaluating an existing synchronisation solution to see if it is a viable option, is of interest to the company. Although, having less control over the synchronisation mechanism can have its disadvantage. For example, requirements of how to handle conflicts are different for different applications.

[1] NoSQL is a name for non-relational databases that are schema free. More about what NoSQL is can be found in section 2.4

1.2 Aim

The aim of this thesis is to evaluate Couchbase as a tool to solve a scalability problem where large amount of data is shared between many mobile devices. Only problems where the shared data can be partitioned based on geographical positions is considered in this thesis.

1.3 Research questions

1. Is the synchronisation time between the server and a client affected by the number of channels that is used?

   This question is relevant in order to test how Couchbase manages many channels. The number of regions that the map is split into affects the number of channels that is required. Because of this, the result of this question can be of great value when deciding the number of regions to use. The result can show limitations if the synchronisation time is affected when the number of channels is increased while the amount of synchronised data is constant.

2. How much disk size is needed to store all data belonging to a region measured in average, median, and the worst case based on the number of regions used?

   In order to decide how many regions to use, it is interesting to know how much disk size the data in each region requires. Together with the result from question one, it can be easier to make a general decision regarding the number of regions to use by looking at trade-offs between the synchronisation time and the disk size. The worst case is defined as the region that occupies most disk space.

3. What improvements do five real system users get with respect to disk size, based on the number of regions used?

   This question focuses on real users that can subscribe to channels that corresponds to regions that they are interested in. This will show how real users are affected by using Couchbase with partitioned data.

1.4 Delimitations

The functionality described in this thesis is only implemented for an iOS application, because this thesis is done in cooperation with IT-bolaget Per & Per whose main focus is to develop iOS applications.

As mentioned, IT-bolaget Per & Per has several systems for their application. The company has a single database for each system where they store objects. Only one database, from one of the systems, is used in this thesis. This is due to the fact that the other databases consist of the same type of data but they differ in size. Even though the result would be affected by which database that is chosen, the general conclusions drawn from the result are applicable to all data sets.

1.5 Structure of report

The first chapter in this thesis consists of an introduction to what the issues are, in a wider perspective, with large amount of data that should be shared between multiple users. The Introduction also includes a motivation to why this thesis is done. It ends with the aim and the research questions for this thesis. The second chapter consists of theory that is relevant for this thesis. The Theory chapter includes how the map could be divided into regions and a thorough description of Couchbase. The third chapter is Related work which consists of

studies that have already been done and are of interest in this thesis. The fourth chapter is the Method that consists of a thorough description of how the work has been done in this study. The chapter includes a description of how data was partitioned into several data sets, how Couchbase was used, and how tests to answer the research questions were implemented. The fifth chapter is the Results chapter that includes all findings from the conducted tests. The sixth chapter is the Discussion chapter. This chapter discuss the results and choices that have been made in the method. The discussion about the method consists of limitations with the chosen tools and what the alternatives could have been. This chapter also include a discussion about ethical and societal aspects. The seventh chapter is the Conclusion.

2 Theory

This chapter contains relevant theory to understand the tools and methods used in this thesis.

2.1 Map Tiles

Sample et al. [34] have written a thorough book about tile-based mapping. A tile-based map is divided into tiles, for example rectangles. Tile-based maps are used to increase performance of Geographical Information Systems (GIS). Google released Google maps in 2005 which allows users to navigate and zoom in the map without the experience of being slow for the user. Tile-based mapping systems fulfil a set of requirements:

- The map views consist of multiple discrete zoom levels where each zoom level maps to a fixed map scale.
- Multiple tiles are used for a single map view.
- Every tile can be accessed through a discrete addressing scheme.
- Minimal processing is done when clients request tiles since most of the processing is done before.

A logical tile scheme is used in tile-based mapping systems to convert positions on the spherical earth to a two-dimensional surface. This scheme defines methods for generating zoom levels and the discrete addressing of tiles. See figure 2.1 of how the the sphere is mapped to a two-dimensional surface and split into tiles. The tiles in the figure are different from how tiles are defined by Google. Because each tile has a discrete address, a tile can be requested based on a row, column, and zoom level. Without such a system, a request could include continuous real numbers representing an area like [-200.0, 10.0] to [-50.0, 20.0] [34].

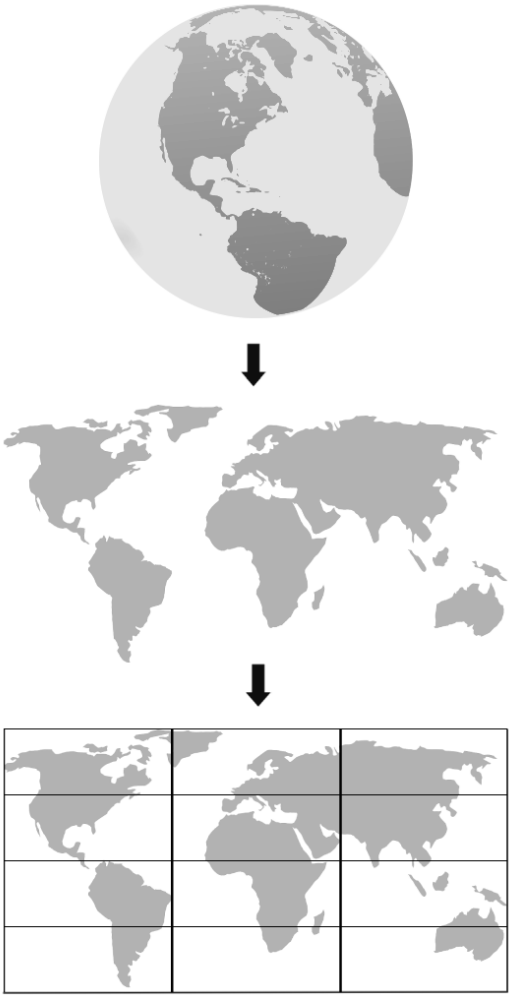

Figure 2.1: Mapping from spherical earth to a two-dimensional surface divided into tiles.

There are different projection base methods that can be used to project the sphere to a grid. One method that is used by both Google and Yahoo is the spherical mercator projection. This method reduces distortion that can occur on shapes and angles at high latitudes. Google define zoom level 0 as a single tile that represents the entire world. For each increasing zoom level, each tile is divided into four sub-tiles. Figure 2.2 shows how tiles are named by Google in zoom level 1. They have their origin in the top left corner [34].

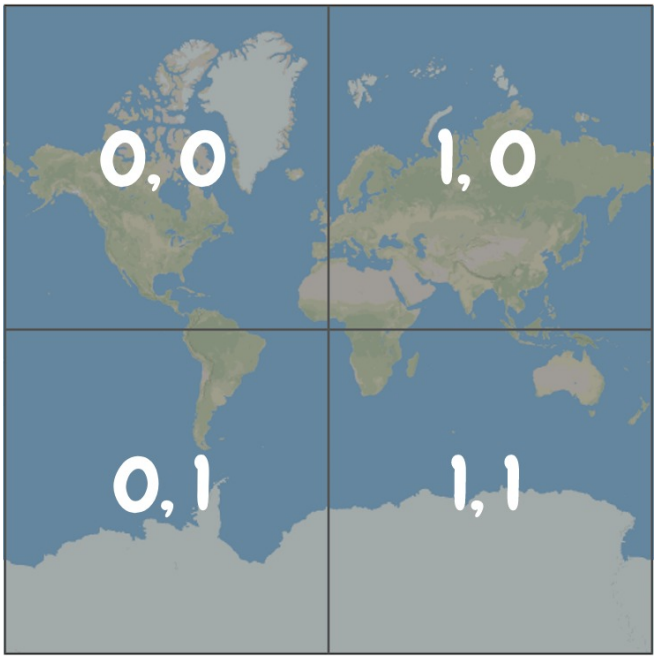

Figure 2.2: Tile addressing by Google on zoom level 1.

2.2 Voronoi diagram

One way to create regions is to use a Voronoi diagram. Voronoi diagram is a plane that is divided into regions based on the distances between a given set of points [21]. These regions are called Voronoi regions. Fortune [21] mentions that Voronoi diagrams can be used to solve closest-site queries. This is a problem where you have several points called sites and one point of interest and you want to decide which site that is closest to the point of interest. Creating a Voronoi diagram over all sites, reduces the problem to instead determine which Voronoi region the point belongs to. He also mentions that there are other fields that Voronoi diagrams can be used in.

A simple way to explain how a Voronoi diagram can be generated is to think of a plane with several points. Each point grows until it collides with another point. When this happens, the point stops growing in that direction and only continues to grow in the other directions. An example of a Voronoi diagram can be seen in figure 2.3.

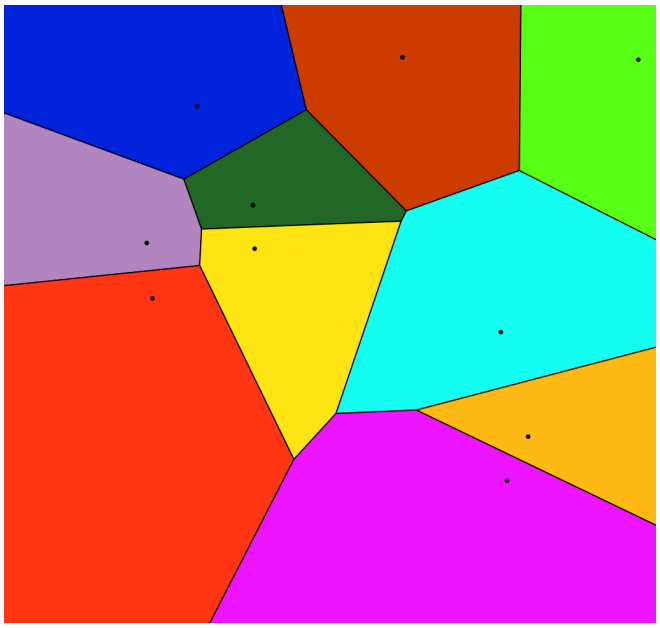

Figure 2.3: Example of a Voronoi diagram.[1]

2.3 R-tree

An R-tree data structure is a tree structure that can be used to create regions. It can be used to represent a group of spatial objects as the minimum rectangle that encloses the objects. This data structure is updated every time elements are added, removed, or modified. R-trees supports spatial data like geometries, and not only single points like Voronoi diagrams. It stores the geometries' bounding boxes without modifying them. A leaf node in the tree corresponds to the object's bounding box. Non-leaf nodes corresponds to the bounding boxes that surrounds every child of that node. An R-tree is automatically balanced when changes in the data set occur. This is done to maintain a certain time complexity when searching for elements in the R-tree [9]. An example of an R-tree can be seen in figure 2.4.

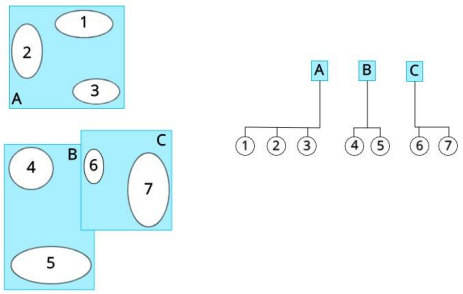

Figure 2.4: Example of an R-tree with two levels.

[1]Figure of Voronoi diagram licensed under, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0/deed.en. No changes made.

2.4 Relational and non-relational databases

Leavitt [25] describes a relational database as a set of tables where each table contains predefined columns. A table consists of at least one column. A row in the table is a record which is data that has been stored. Each record within the same table is unique. Records in a relational database can reference other records in the database and by doing so, relations between tables are created [32]. Potsangbam [32] mentions that non-relational databases, also called NoSQL, differ in many ways from relational databases. One difference is that NoSQL databases are schema free. This means that data can be stored in the database without a predefined schema that defines how the data should be specified. If there are changes needed for a relational database where a table requires modification, there might be a need to take down the server in order to modify the database. This can be a normal task when application evolves and new features are added. With a schemaless database fields can be added as desired. NoSQL databases are designed to handle large amounts of data (terabyte and petabyte) and to scale horizontally, which means that the system adds more server nodes that work together.

Cattell [16] mentions some arguments in favor of using NoSQL databases. One of them is that NoSQL is schema free which was mentioned by Potsangbam. Having a schema free architecture is also allowed in some Relational Database Management Systems (RDBMS) but it is uncommon. Cattell also writes that when using a relational database, expensive operations are made easy for programmers, which is not a good thing because these operations should be avoided. Corresponding operations in a NoSQL database are impossible or expensive for programmers. Bhat et al. [11] evaluated three NoSQL databases; Apache CouchDB, Amazon's SimpleDB, and Google's Bigtable. They found that all three of these were more scalable than traditional RDBMS.

2.5 Couchbase Server

Couchbase Server is a distributed NoSQL database. Ostrovsky et al. [30] say that it is a good NoSQL database solution because of its scalability, availability, and performance. This is also shown in performance tests done by Amghar et al. [5]. Scalability is achieved by Couchbase Server by distributing the data across nodes in clusters [30]. A node in Couchbase Server is defined as an instance of the server and a cluster is a set of nodes [29]. Having distributed data across nodes in clusters makes it possible to spread the workload of lookups and disk IO. This is done by using buckets and virtual buckets (vBuckets), see section 2.5.2 [30]. Availability is achieved by replicating each vBucket across multiple nodes. In case a node fails, another replicated vBucket can be used. High performance is achieved since Couchbase Server will keep data that is accessed often in memory to increase the speed of read and write operations [30].

2.5.1 Key-value stores and document stores in Couchbase Server

Couchbase Server supports two different ways to store data; key-value stores and document stores [30]. Hecht et al. [22] describes key-value stores as similar to a map or a dictionary where data is fetched by a specific key. The data that is stored can only be fetched by keys because the values are not interpreted by the system itself. Because values are isolated and independent there must be application logic in order to manage relationships. Key-value stores are schema free and they are good to use when simple operations on key values suffices. Ostrovsky et al. [30] say that the key-value stores in Couchbase Server can store different data types such as strings, numbers, datetime, booleans, and binary data. The authors do mention that the solution in Couchbase Server even allows the user to do simple operations on values.

Document stores are described by Hecht et al. [22] as documents that contain key-value pairs in JavaScript Object Notation (JSON) format. Every pair of key-value needs to contain a unique key. The document itself also contains a unique key referencing the entire document. The authors continue by saying that document stores make use of the values by allowing the user to query based on these. Document stores are also schema free. According to the authors document stores are suitable when complex data structures need to be managed. Ostrovsky et al. [30] mention that Couchbase Server makes use of JSON documents to store data and has the functionality to index and query based on fields in the documents.

Hecht et al. [22] say that both ways to store data have their strengths. For simple and fast operations they recommend key-value stores and for flexible data models and advanced queries they suggest document stores.

2.5.2 Buckets

Couchbase Server uses buckets to store data. In these buckets the user can store JSON data or binary data. Each bucket is divided into 1024 vBuckets except on MacOS where it is divided into 64 vBuckets. This is a mechanism used by Couchbase Server internally to distribute data across several nodes. The vBuckets are distributed uniformly across the memory and storage space where each vBucket belongs to a node in a cluster. The data in the buckets is distributed evenly into these vBuckets and in this way the workload on Couchbase Server is evenly distributed. Buckets contain document keys that are hashed and used to map documents to vBuckets. Figure 2.5 displays an overview of how buckets, vBuckets, and nodes are related. The figure does not depict replicated vBuckets which are copies of vBuckets used as backups. In order for a client to access a document, a hashing function is used on the key of the document to get the vBucket that contains the document. Then the client can make a lookup in the map between vBuckets and nodes and find the server node where the vBucket resides. This way the client can find the server node that contains the document [39].

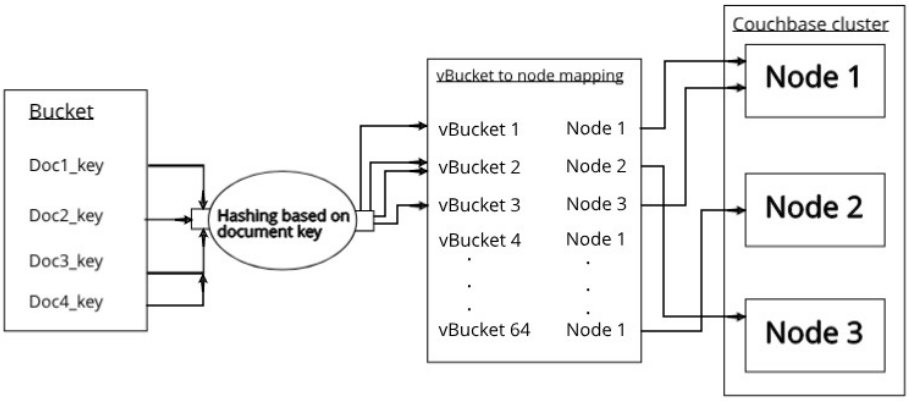

Figure 2.5: Relation overview between bucket, vBuckets, and nodes.

There are three kinds of bucket types to choose between when creating a bucket; Couchbase buckets, Ephemeral buckets and Memcached buckets [14].

Couchbase buckets store data both in memory and on disk. It allows for a functionality to automatically replicate data to several nodes. Each bucket is assigned a memory quota and if this is exceeded, the items will be removed from memory but stay on disk. There are two options to choose between when creating a Couchbase bucket to deal with ejection of data items from memory. One option is *Value-only* which only removes the values. Couchbase suggests this option to achieve better performance. The other option is *Full* which removes all data related to the item. This option is suggested when memory is prioritised over performance.

Ephemeral buckets should be used when persistent storage is not required. It works like a cache and only stores data in memory. To store data on disk is not always needed and the repeated disk access can result in too much overhead. There are two alternatives to choose between to handle ejection for Ephemeral buckets. This is chosen when the bucket is set-up. The first alternative is that data remains in memory and new data cannot be added. The other alternative is that items are ejected so that new items can be added. This means that if an item is removed from memory it cannot be fetched from Couchbase Server because it does not reside on disk.

The last bucket type is a Memcached bucket. It is good to use with other databases, for example relational ones. The Memcached bucket caches data that is frequently accessed. Each bucket has a directly addressable, in-memory key-value cache. This way the number of queries a database needs to perform is reduced. Like Ephemeral buckets, Memcached buckets only stores data in memory. When an item is ejected the item will be removed from memory.

2.6 Couchbase Lite

Couchbase Lite is a mobile database that can be used for several platforms such as iOS, Android, MacOS, Windows, and more. It is designed to work together with Sync Gateway and Couchbase Server. Couchbase Lite includes a database handler, a synchronisation interface, and a database engine [31, 27].

Couchbase Lite also stores documents in JSON format just like Couchbase Server. The documents can contain different data types. Couchbase Lite also allows users to store Binary Large Objects (BLOB). There is a key that is stored in a document that references the BLOB. On a mobile device, BLOBs are stored locally in a separate directory called attachments. Other features enabled by Couchbase Lite is indexing to enhance performance for queries, push and pull replication together with Sync Gateway, and automatic conflict resolving [17].

2.7 Sync Gateway

Sync Gateway works between Couchbase Lite and Couchbase Server. It is Sync Gateway's task to keep documents synchronised between all clients and Couchbase Server. Couchbase Lite that runs on the client-side can only create, delete, and update files locally. Sync Gateway handles the rest. Sync Gateway also has functionality for user access control like authentication and authorisation. Another feature is to route data to specific users based on certain rules. This is done by channels, see section 2.7.2 [38].

2.7.1 User Authentication & Authorisation

According to the Couchbase documentation about authentication [6] there are three different ways to authenticate a user to Sync Gateway. In order to be able to authenticate, a user must be created on Sync Gateway. Having users is necessary for Sync Gateway to function and synchronise documents to the client. Creating a user can be done in two ways:

- **Admin REST API** - To create a user through the Admin REST API [1], a POST request is sent to Sync Gateway containing information about the user. A name and a password must be included. To access the API, the request must be done from a server that has admin privileges. The client cannot use the Admin REST API directly. It is recommended to have another server that runs alongside Sync Gateway that performs validation and creation of a user to Sync Gateway.
- **Configuration file** - A simpler approach, which is more suitable for developing and testing, is to hardcode users in the configuration file before starting Sync Gateway.

The three ways to authenticate a user are by Basic Authentication, Custom Authentication or OpenID Connect. Basic Authentication is the simplest way to authenticate a user. The username and password is provided to a BasicAuthenticator class in Couchbase Lite and the credentials are sent to Sync Gateway in order to receive a cookie that is used during the session. Custom Authentication and OpenID Connect are not of interest in this thesis.

To manage accounts and user settings, each user is stored in a JSON document. This document contains certain properties, one of them is channels. This property include channels that describe what documents a user has been authorised to access [7].

2.7.2 Channels

Couchbase has a feature called channels that makes it more effective to share one database with many users [18]. A channel is like a tag, it is used to identify accessibility to documents. Documents that contain data belongs to a set of channels. When a user is authenticated to Sync Gateway for the first time, a document is created for that user. This document is called user document and contains channels that the user can access in order to only synchronise documents of interest. A user can also have a role where each role is associated with different channels. If a user is assigned a role, the user gains access to the channels associated with that role. One of the two key reasons to use channels is to partition the data set into smaller sets of data when the database grows large. The other reason is to reduce the amount of data that needs to be synchronised for each user. Sync Gateway has a sync function that is written in JavaScript and can be modified as desired. One of its responsibilities is to assign each document a channel every time a document is created or updated. A document can only be read by a user if one of the channels associated with the document is also present in the user's channels.

Channels are defined in each document through the channels property. This is an array of strings where each string represents one channel. There are two main approaches to assign documents to channels. One is through the channels property where desired channels can easily be added. For example, channels can be added on the client side when a document is created. The other approach is through the sync function. It receives a document body and based on its content the document is assigned to channels.

For every channel that the document is assigned to, the name of the channel is added to the document's metadata. The length of the channel name affects the size. There is a size limit

for this metadata at 1 MB or 20 MB per document depending on a flag used in the Sync Gateway's configuration file. The flag is *enable_shared_bucket_access*. If the flag is set to true the limit is 1 MB, otherwise 20 MB. Setting the flag to true enables interoperability between Couchbase Lite and Couchbase Server. Documents can be assigned channels as long as the document's metadata is under the limit.

Beside custom channels, there are two special channels named the public channel, denoted '!', and the star channel, denoted '*'. A document that is assigned the public channel can be accessed by any user because all users have access to this channel by default. A user that is assigned to the star channel will get access to all documents because every document belongs to the star channel by default. It does not mean that the user has access to all channels, it is still one channel.

2.8 File size vs disk size

To measure the amount of data that is stored on a device, it is important to understand the difference between file size and disk size. On a disk, there are sectors that have a certain size. A file that is saved on disk has to be allocated a disk size that is a multiple of the sector size. Modern hard disk drives use Advanced Format, which means that the sector size is 4096 bytes. If a file is smaller than 4096 bytes and stored in the file system, the file will still take up 4096 bytes on the hard drive [36, 20].

2.9 JSON & GeoJSON

JSON is a text format that is used when sending and receiving data between applications [12]. When working with geographic objects, a format called Geographic JavaScript Object Notation (GeoJSON) can be used [15]. GeoJSON is a format to represent geographic objects in JSON format. A GeoJSON object can be a *FeatureCollection* which contains a list of *Features*. *Features* can contain one or several geometry objects, each being a *Feature*. The coordinates for a GeoJSON *Feature* can represent coordinates in a map. GeoJSON supports different geometries like points, polygons, and more. An example of how a pretty printed GeoJSON object can look like, can be seen in listing 2.1. A pretty printed GeoJSON object includes new lines and spaces to increase readability.

Listing 2.1: Pretty printed GeoJSON object that represents a rectangle covering Sweden.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              10.458984375,
              55.178867663281984
            ],
            [
              25.3125,
              55.178867663281984
            ],
            [
              25.3125,
              69.28725695167886
            ],
            [
              10.458984375,
              69.28725695167886
            ],
            [
              10.458984375,
              55.178867663281984
            ]
          ]
        ]
      }
    }
  ]
}
```

3 Related work

This chapter describes research that is related to this thesis.

3.1 Synchronisation with middleware

Wang et al. [40] discuss the problem of data consistency in distributed database systems that are based on a peer-to-peer model. A peer-to-peer model means that all nodes can provide services to update data and query data. The alternative to the peer-to-peer model is a master-slave model where slave nodes have data copies of the master node. To solve the data consistency problem in a peer-to-peer model they propose a protocol that uses a data synchronisation middleware. This middleware is placed between the database system at each node and the network, see figure 3.1. The protocol for the middleware that they present is based on three main stages; local stage, conflict detection stage, and implementation stage. Initially, when a middleware receives a transaction from a user the local stage will analyse the transaction and determine whether it is a read transaction or a write transaction. If it is a read transaction the local node that received the transaction can handle it. However, if it is a write transaction, conflict detection is required. The conflict detection stage will check the transaction and determine if there is a conflict and take actions accordingly. The last stage is responsible for receiving and handling update transactions sent from other nodes and information related to synchronisation. The authors compared this model to a traditional method for guaranteeing serialisability called two-phase locking [10]. Wang et al. found that the amount of communication in the system was reduced and the time to complete a transaction was also improved when conflict detection and resolving was done locally at each node.

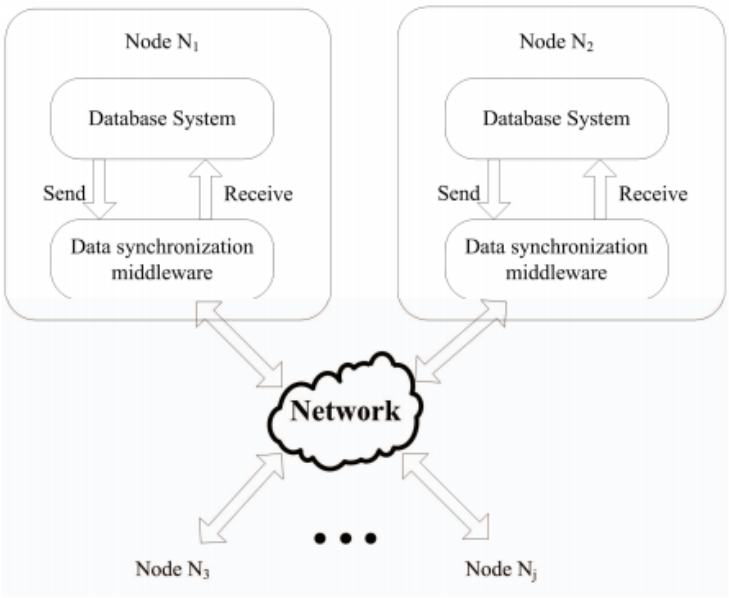

Figure 3.1: System design with middleware for each node [40]. © 2015 IEEE

3.2 Scalable synchronisation

Yee et al. [41] focuses on the problem of synchronisation performance and scalability with mobile databases. Intermittently Synchronised Database (ISDB) systems are database systems where clients intermittently connect to a database and synchronise data. ISDB systems can be used for mobile databases where the synchronisation can be done by using asynchronous message passing. The authors describe that the clients send updates to the server that the server stores in update files. The server will eventually create update files that are downloaded by the clients and used to update their local databases. One important advantage with this message-based synchronisation that the authors mention, is that it is up to the client to decide for how long it is connected to the server. This is important for mobile users because they might be concerned with energy usage. The authors say that message-based synchronisation where update files are generated for each client does not scale well with the number of clients in the system. If there are $\mathcal{N}$ clients and $\mathcal{C}$ updates from each client in average then the server receives $\mathcal{NC}$ updates. The authors argue that the server has to compare each update to every client's subscription, data needs for client, in order to update the client's update file. This process takes $\mathcal{O}(\mathcal{N}^2)$ time. A solution where the server instead generates one update file for all clients is not feasible since the server will send the update to all clients, ignoring the subscription.

Yee et al. suggest improvements to the model by altering the update files. The authors define a datagroup as a set of publications where each publication is a partition of the server's primary database. A client can subscribe to the datagroups that are of interest to the client. Subscribing to a datagroup means that only update files related to that datagroup are downloaded. The authors say that in commercial ISDB systems a Client-centric (cc) datagroup design is often used. This means that each client has its own datagroup. The cost of generating an update file is based on the number of update files, number of update files downloaded by a client, bytes downloaded, and bytes written to disk. According to the authors, the goal is to find a design that has as few datagroups as possible and at the same time as small datagroups as possible. It is hard to achieve both since improving one can make the other one worse. For example, if the server only generates one update file which corresponds to having one datagroup, this

would reduce the cost of bytes written to disk by the server. However, it would increase the total number of bytes downloaded by the clients. The authors present two alternatives to the cc design:

- **One-per publication (op)** - One datagroup for each publication.
- **One-giant datagroup (og)** - One datagroup that contains all publications.

Figure 3.2 shows how datagroups could be designed with the three different alternatives presented by the authors. The designs shown in the figure are based on a situation with three users and four publications. User one wants to subscribe to publications 1 and 3, user two to publications 1, 2 and 4 and user three to publication 4. Each design can fulfil the requirements of each user but it comes with different costs.

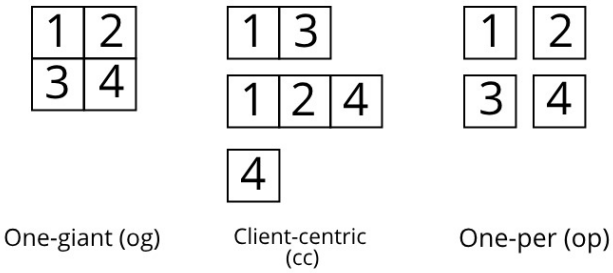

Figure 3.2: Alternative datagroup designs with One-giant, Client-centric and One-per.

The authors run experiments with 400 publications to evaluate the performance by using cc, op, and og designs. For the experiments, the number of publications that each client subscribed to was distributed to be between 5% and 10% of all publications. They also assumed 100 generated updates from each client. The cost of generating update files with respect to disk size was increasing linearly with client population for og and op while it increased quadratically for cc. The authors also measured sync time which they define as "the sum of the average time required to generate and transmit a covering set of update files to a client". When evaluating sync time with low bandwidth, 57.6 Kbps, og performed worst while cc and op had similar cost. For a small number of clients the op result was bad because it had to generate all update files for each datagroup. For better bandwidth, 100 Mbps, og performed best while cc performed worst. They come to the conclusion that op is most suitable when the number of clients increase because cc does not scale well. They continue to argue that in cases where bandwidth is high, og is the best solution. This study was made a couple of years ago and since then the bandwidth has significantly increased. Because of this, og would probably be a good alternative. For smartphones where there are memory limits, an op approach would be more suitable because all data would not have to be downloaded.

3.3 Alternative platform to Couchbase

Agrawal et al. [3] present Simba, which is a platform to facilitate synchronisation for mobile application development. The authors define two types of data storage; table storage that is used for structured data and file systems for larger unstructured data. They discuss that other existing synchronisation approaches have their shortcomings. The shortcomings they

mention are that it is difficult to implement a data model that requires both kinds of data, structured and unstructured, to rely on each other. They say that if there are dependencies between the file system and the table storage, the application will have to handle it. Another shortcoming they mention is that it is burdensome for developers to also have to manage data in separate services, for example in the cloud and locally, that can even have different semantics for synchronisation. The authors say that Simba has three major design goals:

- **Easy application development** - The goal is to have an API that can provide synchronisation and storing for both kinds of data; structured and unstructured.
- **Sync-friendly data layout** - It should be easy to read and query. It should also be easy to identify changes for synchronisation with cloud.
- **Efficient network data transfer** - Minimum network resources should be used for transferring data.

The design of Simba is split into two components; a client application called Simba Client and a server component called Simba Cloud. The authors focus on Simba Client in their study. The local storage on Simba Client is split into a table store and an object store. There is also a component called SimbaSync inside the Simba Client that makes use of both stores to know which changes that should be synced with the server. It also handles conflicts that occur. Simba Client has a data model that merges structured table storage and object storage. This solves the earlier mentioned shortcoming about the difficulties of implementing a data model that requires the two kinds of data. According to the authors, Simba Client helps developers so that they do not have to focus on network management and data transfers. The data storage inside the Simba Client uses SQLite, which is a relational database, for the table storage with an extra data type that is the key for the object storage. The object storage divides objects into chunks and stores them in a key-value store. The reason objects are divided into chunks is to allow for more efficient network synchronisation. The authors say that Simba provides an easy way for developer to construct applications that make use of both table storage and object storage. It is said to provide services to ease development.

3.4 Synchronisation pattern

McCormick et al. [28] discuss different patterns for synchronisation that can be used between servers and mobile clients. They discuss when the different patterns should be used and what strengths and weaknesses they have. One of the patterns the authors present is Partial Storage. The goal with this pattern is to only synchronise necessary data when it is needed. One of the reasons behind this pattern is to optimise storage space usage and another reason is to optimise network bandwidth. They do mention that this pattern is applicable in situations where the entire data set on the server is too large to store on mobile devices. An example they mention is a GIS where different information is displayed based on the user role. In this situation there is no need to receive all information to the device. The solution that they present for this pattern synchronises data dynamically when it is needed. The authors mention that with this pattern there are more network calls because the data is not always stored locally. This specific pattern presented by McCormick et al. is of relevance for our study since our scalability problem is caused by the fact that we use Complete Storage and the entire data set is becoming too large.

3.5 NoSQL and RDBMS comparisons

Băzăr et al. [8] evaluates three popular NoSQL solutions, Couchbase, MongoDB, and Cassandra. The authors also argue why a NoSQL solution would be more beneficial than a

relational database based on the circumstances today. They argue that relational databases were developed for requirements that were viable many years ago but today with more data, more users, and higher demands, a NoSQL solution is preferable. Relational databases scale vertically which means that scaling is solved by a bigger server while NoSQL solutions scales horizontally. They also mention schema free architectures as a benefit because in situations where there is a schema to follow, changes to the database can be more difficult to implement. An interesting aspect that the authors mention is that it can be easier with a NoSQL solution to keep consistency between data when it is updated. This is because a record in a NoSQL solution usually uses documents as data models that contain all information. This way, when an update occurs only a single document can be updated compared to relational databases where multiple tables might be affected. When the authors compared the three NoSQL solutions against each other, they all fulfilled the requirements of horizontal scalability and high availability. For performance, the Couchbase solution gave the lowest latency for reads, writes, inserts, and updates. They do mention that each NoSQL solution is different and it can be difficult to choose which one to use.

Amghar et al. [5] compares five different NoSQL databases, Couchbase, MongoDB, Cassandra, Redis, and Neo4j, for Internet of Things (IoT) applications. There are four different categories for NoSQL databases; key-value databases, column-oriented databases, document-oriented databases, and graph databases. Couchbase and MongoDB are document-oriented databases, Cassandra is a column-oriented database, Redis is a key-value database, and Neo4j is a graph database. Although this thesis does not focus on IoT, some of the requirements for these kinds of applications are also requirements for mobile applications. The authors mention that they will focus on seven requirements that IoT applications usually have. Three of these are also interesting for a mobile application:

- **Spatial data handling** - Data objects with relation to geographic space.
- **Scalability** - Should allow for scalable storage solutions.
- **Real time processing** - Data needs to be in real time.

They say that for spatial data handling the document-oriented databases and graph databases were mostly used for storing spatial data. They mention that Cassandra and Couchbase where the two most scalable databases among the five tested ones. At last, for real time processing, they refer to another study that shows that Couchbase has the best performance for most database operations. The authors conclude that Couchbase is most suitable and provide the best capabilities for an IoT application.

3.6 Mobile cloud computing

Farrugia [19] ran tests in order to evaluate whether mobile cloud computing could be used to relax the constraints that a mobile device has. The author ran tests where local data storage, cloud based storage and peer-to-peer storage were used. For local storage, both SQLite (relational database) and Couchbase Lite (NoSQL) were used. For server storage, Couchbase Server was used. In the tests, the author focused on five criteria; CPU load, memory allocation, battery usage, data usage, and time. The findings were that local frameworks, SQLite and Couchbase Lite, gave better results for the measured criteria. However, the author discusses that a cloud based solution has advantages that were not focused on, such as reliability, robustness, and security. Farrugia recommends to use Couchbase Lite only for simple applications where data is not shared among multiple users.

4 Method

This chapter describes what has been done in order to fulfil the aim and answer the research questions that are stated in section 1.3. To answer the research questions, three tests were performed, one for each question. The goal for the first test was to evaluate how Couchbase will perform given different number of channels. The second test calculated disk sizes for every region on each zoom level. The third test's aim was to see how real users were affected by a Couchbase solution for different zoom levels. Before a thorough description of how these tests were conducted, this chapter first explains the process to set up a Couchbase Server instance, a Sync Gateway, and a Couchbase Lite database to work together. This set up also includes how data was partitioned into channels and how data was migrated from an existing relational database.

4.1 Solving the scalability problem

After some research and discussions with IT-bolaget Per & Per the main idea to solve the scalability problem was to partition the data into several data sets, like McCormick et al. [28] describe in their paper about the Partial Storage pattern. The idea was to have one database for each data set where users could subscribe to databases of interest. Figure 4.1 shows the idea described with two users that subscribe to their desired databases. This thesis used Couchbase to implement the described idea but instead of using several databases, only one database was used. The data was partitioned with the help of channels in Couchbase.

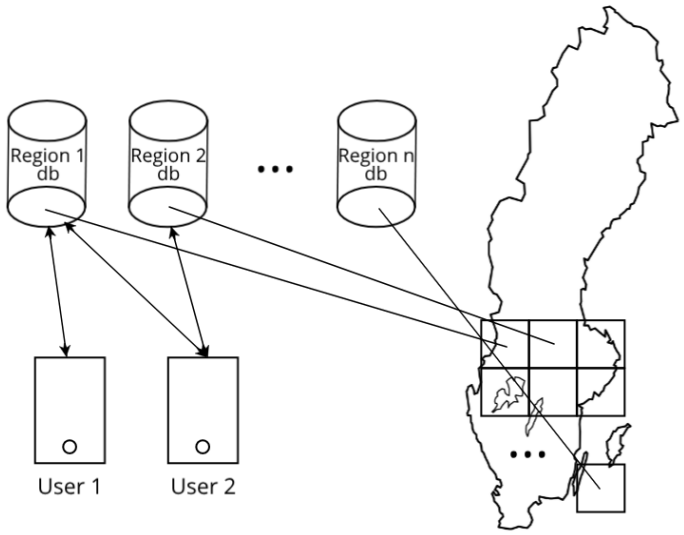

Figure 4.1: Partitioning of data with one database per region and users that can subscribe to desired databases.

4.1.1 Partitioning of data

There are several ways to partition data into subsets. The approach that was used in this thesis was tile-based mapping, where the zoom levels were defined in the same way as Google defines them, see section 2.1. Every tile in the map represents a region. There are several other approaches that could be used to partition the data into regions. Each approach, including tile-based mapping, comes with advantages and drawbacks which are discussed in section 6.2.2. Tile-based mapping was used in this thesis because IT-bolaget Per & Per had tools that helped with the problem of deciding which region a geometry belonged to. Another reason for choosing this approach is because the applications that the company have uses using tile-based mapping for map rendering.

4.2 Couchbase

There are several non-relational databases that could have been used instead of Couchbase, for example MongoDB, Cassandra, Redis, and Neo4j. Băzăr et al. [8] mention that it could be difficult to choose which database to use. The main reason for using Couchbase, as mentioned in the Introduction, was because IT-bolaget Per & Per had a wish to evaluate it. Wang et al. [40] found advantages of having a middleware for each node when synchronising data, see section 3.1. Couchbase has a similar approach with one middleware between the server and all other clients. Even though this middleware is not located locally at each node, it still handles data consistency and conflicts.

An alternative platform that could have been used instead of Couchbase is Simba, which was presented by Agrawal et al. [3], see section 3.3. Simba would be a good alternative because it has built in functionality to handle synchronisation. However, Couchbase was used since it has built in functionality to partition data into data sets and wide online support for the platform. Farrugia [19] recommends to use Couchbase Lite only for simple applications, see section 3.6. However, the tests run by the author did not evaluate any method to solve the problem when there is shareable data amongst many users, which is intended in this thesis.

4.2.1 Couchbase Server configuration

Couchbase Server 6.0.0 community edition[1] was used in this thesis. Couchbase Server was used to store data that should be synchronised between mobile devices. It was running on a MacBook Pro, from 2014, with a 2,5 GHz Intel Core i7 processor, 16 GB RAM, and a 500 GB SSD. The operating system was macOS Mojave version 10.14.3.

After Couchbase Server was installed, a server instance had to be created and configured. This was done by setting up a single node which is an instance of the server. This node operates inside a single cluster. The server instance was running locally and had a data memory quota of 8192 MB. When the server instance was initialised, the next step was to create a bucket. The bucket was assigned 8192 MB memory quota and was of type Couchbase bucket. No replicas was chosen because it was not needed for the tests done in this thesis. We also chose to create a flushable bucket which means that the bucket can be cleaned from data, without the need to delete the whole bucket and create a new one. In order to increase performance, ejection method *Value-only* was chosen. See appendix A for the complete configurations of the server and the bucket.

In order for Sync Gateway to communicate with the server, a special Role Based Access Control (RBAC) user needed to be created on the server. The RBAC user was created via Couchbase Server's Graphical User Interface (GUI). The user was given a name, *sync_gateway*, and a Read-Only Admin role. *Sync_gateway* was also given Application Access to the bucket. The complete RBAC user settings is shown in appendix A.

4.2.2 Sync Gateway configuration

Sync Gateway 2.1.2 community edition[2] was used in this thesis. As described in section 2.7, Sync Gateway works like a bridge between Couchbase Server and Couchbase Lite in order to keep documents synchronised.

Sync Gateway was running on the same hardware configuration as the Couchbase Server instance. Sync Gateway needed to be run with a configuration file. The configuration file was specified as a JSON file with several attributes. In order to run the tests on hardware devices and not on a simulator, the *adminInterface* attribute needed to be set to ":4985". This meant that the interface was reachable from other hosts. To connect Sync Gateway to the server, the *databases* attribute was specified in the configuration file. The *databases* attribute allows Sync Gateway to connect as the created RBAC user on the server. Within the *databases* attribute, the attribute called *enable_shared_bucket_access*, see section 2.7.2, was set to true. All available attributes for the configuration file can be seen in the official documentation of Couchbase[3]. The configuration file that was used in this thesis can be seen in listing 4.1.

[1] `https://www.couchbase.com/downloads`
[2] `https://www.couchbase.com/downloads#couchbase-mobile`
[3] `https://docs.couchbase.com/sync-gateway/2.1/config-properties.html`

Listing 4.1: The Sync Gateway's configuration file used in this thesis.

```
1  {
2      "log": ["*"],
3      "adminInterface": ":4985",
4      "databases": {
5          "sofa": {
6              "server": "http://127.0.0.1:8091",
7              "bucket": "sofa",
8              "username": "sync_gateway",
9              "password": "password",
10             "enable_shared_bucket_access": true,
11             "num_index_replicas": 0,
12             "import_docs": "continuous",
13
14             "sync": `function (doc, oldDoc) {
15
16                 if (doc.channels) {
17                     channel(doc.channels);
18                 }
19             }`
20         }
21     }
22 }
```

The sync function is defined on line 14. The sync function takes two arguments that represents the document that is to be synchronised, *doc* and *oldDoc*. The *doc* argument represents the updated version of *oldDoc*. The only thing that was done in the sync function was to create new channels. This was done by the *channel(doc.channels)* call on line 17 in the code. Sync Gateway will use channels to synchronise data between server and clients. The sync function supports other features, like validating documents, deciding read and write permissions, and adding channels to a user's channel list. These functionalities were not used because a simple sync function that only creates channels based on the document's content sufficed for the purpose of this thesis. The reason to not restrict users to modify certain documents was because the intended use of the company's application was that all users should be able to create, update, and delete a document. This simple sync function is enough to provide each user with data of interest based on which channels the user subscribes to.

4.2.3 Couchbase Lite configuration

In this thesis, Couchbase Lite 2.1.x community edition[4] was used. In order to connect the Couchbase Lite database to Sync Gateway, a replicator had to be configured. The replicator needed to be given a local Couchbase Lite database and an IP address to the host that ran Sync Gateway. Both applications built in this thesis, see section 4.2.5, used the same Couchbase Lite configuration. The replicator was configured to be in *continuous* mode, which means that changes made by the client and the server will continuously be synchronised. The other alternative is to set the replicator in *one shot* mode. *One shot* mode means that the replicator is disconnected after one change and the client must start a new replicator if new changes appear. This thesis used *continuous* mode since this will be IT-bolaget Per & Per's intended use and none of the tests performed required the use of *one shot* mode.

[4] https://www.couchbase.com/downloads

As described in section 2.7.1, the client needs to authenticate to Sync Gateway to synchronise data. In order to authenticate, a user was first created via the Admin REST API. After that, the user could be authenticated in the replicator configuration via the BasicAuthenticator class in Couchbase Lite. Basic Authentication was used because it is the simplest way to authenticate a user and it sufficed for our testing purposes.

4.2.4 Channels

The approach to split the data into several data sets was done with the help of regions and Couchbase channels. For each region, which was defined by a tile, a channel was created. Each object in the database belongs to a channel based on its geometry's position. There are situations where there is a possibility that a geometry overlaps several regions. To handle this, a document which represents one object is assigned all channels that the object's geometry resides in. By using channels all data can be stored in one database and at the same time be partitioned. There is no need to have several databases where each client must connect to multiple databases. The chosen strategy is inspired by the One-per publication design presented by Yee et al., see section 3.2. Since each region has its own channel, it can be seen as a separate data set that users can subscribe to.

As mentioned in the theory about channels, the length of a channel name affects the size of documents' sync metadata. In this thesis the channels are named so that they can easily be mapped to a tile according to Google's addressing scheme. The name is of format zoom/-column/row. For example, 7/70/37 is a tile that is covering a little bit more than Stockholm. The column and row count starts from the top left corner of the earth when it is projected on a two dimensional surface. Where this exact location is depends on the zoom level. From the metadata size perspective it would be more convenient to just number the tiles starting from 0 and counting, but this would make it more difficult for developers to know which tile the channel represents.

The Admin REST API has been used to set the channels for a user when the application launches. A PUT request was made from the client itself. Although a client should not be able to use the Admin REST API, it worked because the server and client ran on the network. By setting the channels this way it was flexible to change the channels that a client subscribed to when running different tests. Documents were assigned channels when they were created. The way the mobile applications was designed for this thesis, documents could not be created by user interaction. Instead, documents were created when migrating the data from a relational database to the Couchbase Server instance.

4.2.5 Swift applications

TestingApp is a mobile application that was created with Swift[5] version 4. *TestingApp* was designed as a tool to answer the research questions. To answer research question one, a synchronisation time test was implemented. A more detailed description of this test can be found in section 4.4.1. Two more tests were implemented in the *TestingApp* to answer research question two and three. These tests are called region statistics test and disk size for users test, see sections 4.4.2 and 4.4.3.

To fulfil the *TestingApp's* purpose, a support application was built. The support application is called *DataHandlerApp*. One of the purposes of *DataHandlerApp* was to migrate data to the server. The migration process is described in section 4.3.1. The other purpose of *DataHandlerApp* was to generate three different groups of files that *TestingApp* was dependent on in order to answer the research questions. The groups of files are:

[5]Swift is a programming language for macOS, iOS, watchOS, tvOS and more [37]

- **Active regions** - Files that contained information about which regions that had data for different zoom levels. These files were used for two purposes, to support *TestingApp* with the synchronisation time test and to support *DataHandlerApp* with generation of the *disk size per region* file.
- **Disk size per region** - Files that contained how much disk space every region requires for different zoom levels. These files were used by *TestingApp* for the calculations of the region statistics test.
- **GeoJSON data** - Files that were used to visualise where the data is located on a map. The visualisation was used for the region statistics test to see how the data was spread across the map. These files contained GeoJSON data that represents regions with data.

4.3 Data

The data that was used to perform the tests was real data generated from actual users from one of the company's system. The data was a snapshot of the database from a client where most users reside in the middle part of Sweden. Because of this, most geometries created by the users are not spread across the entire country.

The company stores the data in an SQLite database. The snapshot of the database consisted of 39854 records that took up 40.7 MB of storage on an iPhone XS 64 GB model. The snapshot consisted of one table with 46 columns. One of the columns contained BLOBs which represent geometries that users draw on the map. The rest of the columns were plain strings, dates, and numbers. All records did not have values for all columns, instead they had *null* values. *Null* values represents that a record does not have a value for that column.

4.3.1 Data migration

As mentioned in section 2.6, BLOBs are stored as separate attachments in Couchbase. There were two options to migrate the data from the SQLite database to the server; with attachments or without attachments. The migration process for both options is almost the same.

To migrate the data, *DataHandlerApp* was used. The first thing that was done in the migration process was to configure a replicator and establish a connection to Sync Gateway. This was done in the same way as described in section 4.2.3. After a replicator was set up, all records were extracted from the SQLite database and handled as objects in Swift. This was done with the help of a framework called FMDB[6]. After this, a model for every record in the SQLite database was created with all corresponding fields. The records in the SQLite database that contained columns with *null* values were replaced with a field in the model containing an empty string. The reason to include all columns from the SQLite database was to maintain the same structure and make the disk size comparison between the two databases feasible. When it comes to the geometry fields, BLOBs were created by using methods provided by Couchbase Lite in order to migrate the data with attachments. To migrate the data without attachments, the BLOBs were encoded to base64 data and interpreted as a string. This string was stored together with the other fields in the model. A document was then created for each model. A property that was added for each document was the channel property. To determine which channels a document should be assigned to, a method already implemented by the company was used. The method returned all tiles, by name, that the geometry resided in. The name of the tiles was of format zoom/column/row. Since each document represented one geometry, the channels for that document were equal to the result from the method. When a geometry overlapped several regions, the corresponding document was assigned

[6] `https://github.com/ccgus/fmdb`

to all channels representing those regions. This was done because of simplicity. It helps to keep the complete geometry in all regions instead of splitting it into several ones based on the border of the regions. Several difficulties arises with splitting geometries; how splitting should be done and how the different parts should be tracked in order to assemble them into one complete geometry again. At last, every document was stored in the local Couchbase Lite database which triggered Sync Gateway to synchronise all documents to the Couchbase Server instance. The migration process was run in an iOS simulator.

A single zoom level had to be chosen before migrating the data, several zoom levels was not used at the same time. Since one zoom level generates a fixed amount of regions based on the data set, the zoom level had to be changed in order to vary the number of regions and therefore the number of channels to be used. Every time the zoom level was changed, the database was deleted and the migration process was rerun.

4.4 Tests

Three tests were conducted in this thesis in order to answer the research questions. The first test was called synchronisation time, which measured how long time it takes to synchronise data for each zoom level. The second test was called region statistics and calculated average disk size per region, the median disk size, and which region that had the most disk size. The third test was called disk size for users. This test evaluated how much disk size real users would need to use on different zoom levels based on the regions they are interested in. This was compared to storing the entire SQLite database.

Each test was run twice, once when the migration process was done with attachments and once without attachments. The reason for running with both options is that a file is created to store each BLOB when using attachments. Since modern hard disk drives uses Advanced Format, see section 2.8, every file would be rounded up to closest 4096 byte multiple. This means that BLOBs could take up more disk space compared to how large they really are. To avoid this, the tests without attachments stored the BLOBs in the document as a base64 data string. There is a drawback from a disk size perspective to convert binary data to a base64 string representation, this drawback is discussed in section 6.1.2

All tests were run on a mobile device. The reason for not running the tests on an iOS simulator was because the computer's hardware is used. This would not be representative for the actual system which runs on mobile devices. By doing this the results would not be affected by hardware that are not available for mobile users. The device used was an iPhone XS with iOS version 12.1.4, 64 GB memory, and 4 GB RAM.

Each of the three tests were required to run on several zoom levels. *TestingApp* does not have the functionality to change zoom level at run time. Instead the zoom level was hardcoded in the application. The reason for this was because for each zoom level, the data in the server also needed to be flushed manually and restored with new data which was done via *DataHandlerApp*. *TestingApp* could then run tests with the hardcoded zoom level that matches the zoom level that was used to fill the server with data.

All three tests were conducted on all zoom levels between 4 and 14, including those. This meant that every test needed to be run eleven times. The choice of not running the tests on zoom levels beneath 4 was made because the region at zoom level 4 was the smallest region that still covered all geometries in the database. Using a lower zoom level would give a larger region but it would give the same result as zoom level 4. The choice of not using higher zoom levels than 14 was made because it resulted in too many regions since each region is split into

4 new regions at most for each increasing zoom level. A consequence of this was that it took to much time to run tests on zoom levels above 14.

As mentioned in section 4.3.1, geometries were stored in all regions it overlapped. If a geometry resides in two regions and a user subscribed to one channel representing one of those regions, the complete object for that geometry would be synchronised. If a user subscribed to both channels the complete object would be synchronised once and not twice. This is a mechanism that comes with Couchbase channels.

4.4.1 Synchronisation time

The synchronisation time test evaluated how much time it took to synchronise all data for the different zoom levels specified in section 4.4. A test user was created that subscribed to every single channel that existed on the current zoom level that was being tested. By doing this the same amount of data was synchronised for each zoom level. This way the result for different zoom levels could be compared. The user did not subscribe to the star channel that was mentioned in section 2.7.2. The reason for this was to find how the synchronisation time was affected by the number of channels a user subscribed to. Subscribing to the star channel means that the user only subscribes to one channel.

To know which channels to subscribe to for a specific zoom level, *DataHandlerApp* was used to generate an *active regions* file for that zoom level. The file contained a list with all regions that had at least one geometry. The process of generating this file consisted of three steps. The first step was to use the FMDB framework to read and extract data from the SQLite database. The second step was to gather all regions that intersected with each geometry. This was done with a method provided by the company. The last step was to store each region in a set to remove duplicates and then save it to a file. For the synchronisation time test, eleven *active regions* files were generated.

When the *active regions* file was generated for a specific zoom level, *TestingApp* was used to set up a user that subscribed to these channels. After the replicator was started, which activates the synchronisation, a timer was started programatically in Swift. When all data had been synchronised to the user the timer was stopped. To know when the synchronisation was completed the replicator's *ChangeListener* was used. It has a status property that checks the activity of the replicator. The *ChangeListener* was called based on the replicator's change events. For one test run on a zoom level, the synchronisation process was automatically repeated five times. The process deleted the local database between the runs in order to repeat the synchronisation. The test was performed twice for each zoom level, one time in the forenoon and one time in the afternoon. This gave ten results for each zoom level. The reason to run several times was made because the synchronisation was done over WiFi and the network load could be different during different times of the day. By using ten results for each zoom level, the effect that variations in the network could have on the result was reduced. Before the synchronisation time test was run, the computer that hosted the Couchbase Server instance and Sync Gateway only had applications necessary to run the tests open. On the mobile device, all background applications were closed and the RAM was cleared before running each test. All ten results for a specific zoom level were measured in seconds and then summarised. The sum for each zoom level was divided with ten in order to get the average time it would take to synchronise all data for a specific zoom level.

4.4.2 Region statistics

The region statistics test was performed to get a better overview of how the number of regions affected disk size. In order to run the test, a dependency file called *disk size per region* was

needed. This file contained the disk size required to store each region individually, given a zoom level, on the mobile device.

The process to generate the *disk size per region* file was made in the *DataHandlerApp*. The process required that a user was set up to subscribe to one of the available channels for the zoom level that was being tested. In order to know which channels that were available, the *active regions* file was required. How this file was generated is described in section 4.4.1. When this was in place, the synchronisation was started and all data that were associated with the chosen channel was synchronised to the disk on the mobile device. The disk size was stored as a value in a dictionary, the key for that value was the synchronised channel. This process was repeated until every channel in the *active regions* file was iterated. The generated dictionary was then saved to a file. Eleven *disk size per region* files were generated, one for each zoom level.

When the dependency files were generated, the region statistics test could be run in the *TestingApp* to calculate the largest, average, and median disk size required for a region. The region that required the largest disk size for each zoom level is referred to as the worst case. The average disk size for a region was calculated according to equation 4.1. To calculate the median, all disk sizes were first sorted. The median value of the regions' disk size was then given by the middle value in the sorted list. If the number of regions was even, the median value was instead calculated by calculating the average value (equation 4.1) of the two values that were closest to the middle. All three metrics were presented in MB and as a percentage of the total disk size. The total disk size was defined as the size needed on disk when there was one region, which corresponds to synchronising all data. The region statistics test had to be rerun for every zoom level.

$$Average\ disk\ size\ per\ region\ = \frac{\sum_{x \in\ set\ of\ regions'\ disk\ sizes} x}{total\ number\ of\ regions}\ [bytes] \tag{4.1}$$

The choice of calculating the three metrics mentioned above was made to get a clear overview of how much disk size a user could expect to need when one region is synchronised. These metrics were used together with the results from the synchronisation time test in order to find bottlenecks when increasing zoom level. This was done by looking at trade-offs between synchronisation time and the disk size required. The result was also used with the purpose to find an optimal zoom level that could be used in production. Rousseeuw et al. [33] describe *outliers* as observations that are different from the majority. The data set that contains the regions' disk sizes can have outliers and to only calculate the average disk size per region could lead to the fact that these outliers are missed. All three metrics together gives a wider perspective of the result compared to the individual metrics alone.

To visualise where the regions were located and thus how the data was spread on different zoom levels, an online tool[7] was used. This tool has functionality to create and edit GeoJSON objects. The *active regions* files were used to generate *GeoJSON data* files which contain GeoJSON formatted data that represents geometries of the regions. GeoJSON is the format that is required by the online tool.

4.4.3 Disk size for users

The disk size for users test was performed to get an overview of how five real users' disk size would be affected by a channel approach with Couchbase. The method for this test is

[7] `http://www.geojson.io`

similar to the region statistics test. The idea for each of these five users was to subscribe to the channels of interest. The disk size for each of the five users on every zoom level was measured. If a user had created a geometry or was the last user to edit a geometry, the user was defined to be interested in the corresponding channel for that geometry. This was the only way to decide which channels a user was interested in. In a real scenario, a user can be interested in geometries that are created and edited by other users.

The five users that were chosen are real users from the SQLite database. Every record in the database contains the columns, *createdBy* and *editedBy*. These columns hold information about which user that created the record's geometry and which user that last edited it. The first user was chosen based on the one that was involved in most records. This user was called most active user and was chosen because this user has the most number of geometries to synchronise. It was of interest to see how a user with much data was affected. For the second user, a sorted list of users was created based on the number of geometries they had created or edited. From this list the middle user was chosen, which worked because the number of users was uneven. This user was called average user and it was chosen with the intention to represent the average user. The third, fourth, and fifth users were randomly chosen and were called random user one, random user two, and random user three.

Compared to the region statistics test, this test focused on users that subscribed to several channels. This is interesting because it was reflecting a true usage for different kinds of users in the application provided by the company. The result was presented as the amount of disk size required to store the data on the mobile device for every zoom level.

5 Results

This chapter presents the results for the three tests that were described in the method. First the results from the synchronisation time test is presented, then the results from the region statistics test is presented. At last, the results of how five real users would be affected by a Couchbase solution with focus on disk size is presented.

5.1 Synchronisation time

The amount of time the synchronisation took with and without attachments is found in table 5.1. The number of channels is decided by the zoom level and where the geometries are located. Only regions that contain a geometry or part of a geometry is considered in this test. The time is presented in average of ten values in order to eliminate variations in the network. All ten values from the tests for each zoom level is presented in appendix B.1 and B.2

When attachments are used, there is a significant time difference between the zoom levels starting from 10 to 14. However, the number of regions containing data also increase significantly between these zoom levels. Figure 5.1 shows all average synchronisation time values with attachments plotted in a graph. The graph shows that the synchronisation time increases fairly linearly when the number of channels increase from around 40 to around 15000 channels. The line in the graph looks exponential but the x-axis has a logarithmic scale. Without attachments the synchronisation time is significantly decreased. The time does not increase linearly as it does with attachments. The only increase in time is shown after zoom level 12. The result for zoom level 4 represents the case where all data is in one database and all data must be fetched for each client. This is where the scalability problems from a disk size perspective is in the company's current implementation. See how disk size is affected by zoom levels in section 5.2.

Zoom level	Number of channels subscribed to	Average synchronisation time with attachments	Average synchronisation time without attachments
4	1	41.80 Seconds	17.12 Seconds
5	2	41.24 Seconds	17.31 Seconds
6	6	42.00 Seconds	16.99 Seconds
7	13	41.83 Seconds	17.41 Seconds
8	41	42.87 Seconds	17.39 Seconds
9	119	44.02 Seconds	17.38 Seconds
10	362	52.23 Seconds	18.05 Seconds
11	1124	95.33 Seconds	17.98 Seconds
12	3134	196.34 Seconds	17.61 Seconds
13	7220	466.71 Seconds	21.08 Seconds
14	14623	985.35 Seconds	31.32 Seconds

Table 5.1: Synchronisation time test result for all zoom levels.

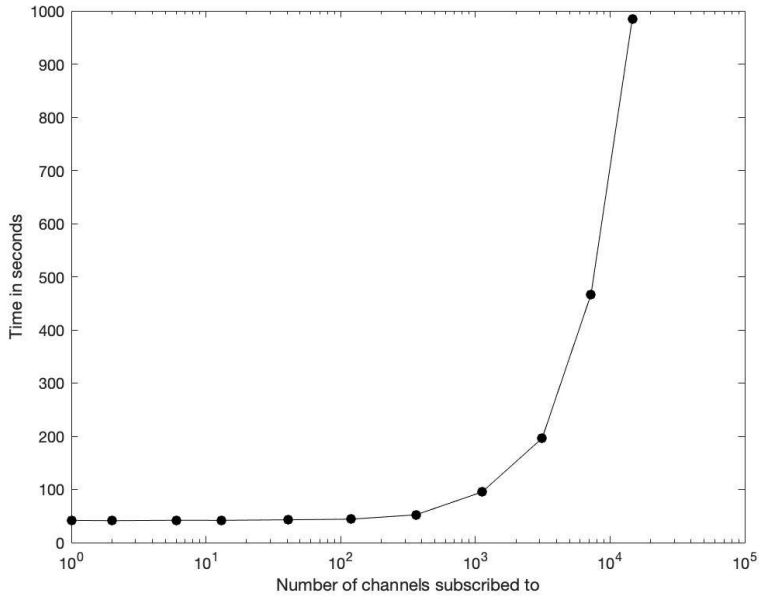

Figure 5.1: Graph representation of the result from the synchronisation time test with attachments.

5.2 Region statistics

The result for the region statistics test with attachments is presented in table 5.2 and the result without attachments is presented in table 5.3. At zoom level 4, the total amount of disk size required for the mobile device was 206.02 MB with attachments and 82.48 MB without attachments. The disk size needed when the data was stored with SQLite and no regions were used was 40.7 MB. Using a zoom level of 4 would mean that more disk size would be required than the old approach used by the company regardless of whether attachments are used. The disk size to store all data has increased with a factor around five and two respectively. If only one region would be synchronised, a zoom level of 6 could be more effective from a disk size perspective compared to the old approach. However, the largest region on this zoom level requires 119.12 MB of disk size with attachments which is still significantly more than 40.7 MB. The largest region without attachments at zoom level 6 is also larger than 40.7 MB. Zoom level 8 is the first level with attachments where the largest region is less than the entire data required with the old approach. In average, zoom level 8 requires around three percent of the total data disk size per region and about 15% (6.26 MB/40.7 MB) of the old total data disk size per region. Without attachments, a zoom level of 7 is the lowest zoom level where the largest regions is less than the entire data set for the old approach.

Even though the number of regions increase considerably on higher zoom levels, the average, median, and worst case disk size does not decrease accordingly. This is because the regions start to become smaller and some geometries start to overlap several regions. For higher zoom levels than 5 both tests shows that the median is always lower than the average. This shows that there are many outliers in the high end of the distribution.

Comparing the results with and without attachments, both approaches decrease the amount of disk size required in similar patterns. This can be seen clearly in the figures 5.2 and 5.3. Worth noting is that, for this data set, it does not matter much if attachments are used or not for the highest zoom levels. The tables and the figures show that the approach without attachments has a better starting point from a disk size perspective but they both converge to similar values. The approach with attachments catches up for the highest zoom levels.

Zoom level	Number of regions containing data	Average disk size per region		Median disk size per region		Worst case	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	206.02 MB	100%	206.02 MB	100%	206.02 MB	100%
5	2	103.89 MB	50.43%	103.89 MB	50.43%	171.46 MB	83.22%
6	6	35.99 MB	17.47%	20.83 MB	10.11%	119.12 MB	57.82%
7	13	17.46 MB	8.47%	7.77 MB	3.77%	62.24 MB	30.21%
8	41	6.26 MB	3.04%	2.55 MB	1.24%	32.53 MB	15.79%
9	119	2.28 MB	1.11%	1.21 MB	0.59%	22.79 MB	11.06%
10	362	0.82 MB	0.40%	0.53 MB	0.26%	8.20 MB	3.98%
11	1124	0.37 MB	0.18%	0.27 MB	0.13%	4.61 MB	2.24%
12	3134	0.24 MB	0.12%	0.20 MB	0.10%	1.77 MB	0.86%
13	7220	0.21 MB	0.10%	0.18 MB	0.09%	1.16 MB	0.56%
14	14623	0.19 MB	0.09%	0.16 MB	0.08%	1.15 MB	0.56%

Table 5.2: Region statistics test result for all zoom levels with attachments.

Zoom level	Number of regions containing data	Average disk size per region		Median disk size per region		Worst case	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	82.48 MB	100%	82.48 MB	100%	82.48 MB	100%
5	2	43.81 MB	53.12%	43.81 MB	53.12%	69.56 MB	84.34%
6	6	15.23 MB	18.46%	9.15 MB	11.10%	49.17 MB	59.61%
7	13	8.22 MB	9.97%	3.66 MB	4.44%	26.70 MB	32.37%
8	41	3.09 MB	3.75%	1.70 MB	2.06%	17.52 MB	21.24%
9	119	1.14 MB	1.38%	0.52 MB	0.63%	15.07 MB	18.27%
10	362	0.41 MB	0.50%	0.29 MB	0.35%	4.60 MB	5.57%
11	1124	0.24 MB	0.29%	0.19 MB	0.23%	2.04 MB	2.48%
12	3134	0.19 MB	0.23%	0.17 MB	0.21%	1.01 MB	1.23%
13	7220	0.18 MB	0.22%	0.16 MB	0.20%	0.87 MB	1.06%
14	14623	0.18 MB	0.22%	0.15 MB	0.18%	0.81 MB	0.98%

Table 5.3: Region statistics test result for all zoom levels without attachments.

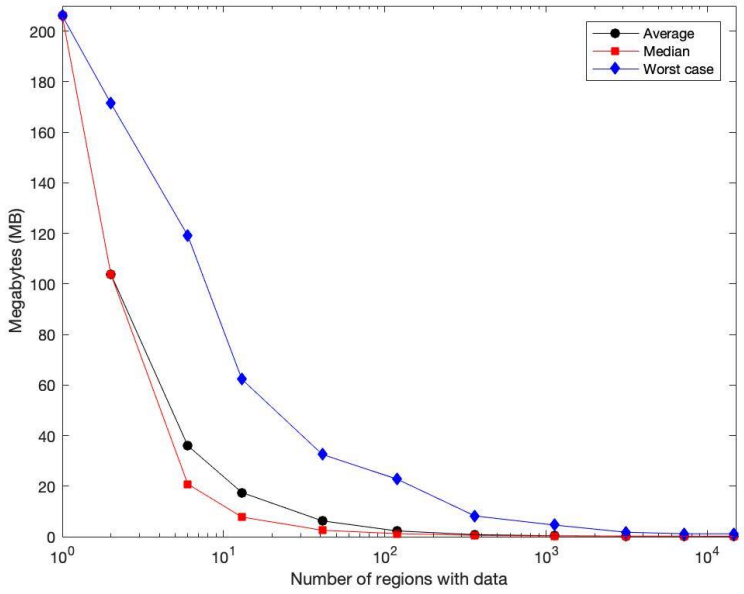

Figure 5.2: Region statistics test result plotted in graph for all zoom levels with attachments.

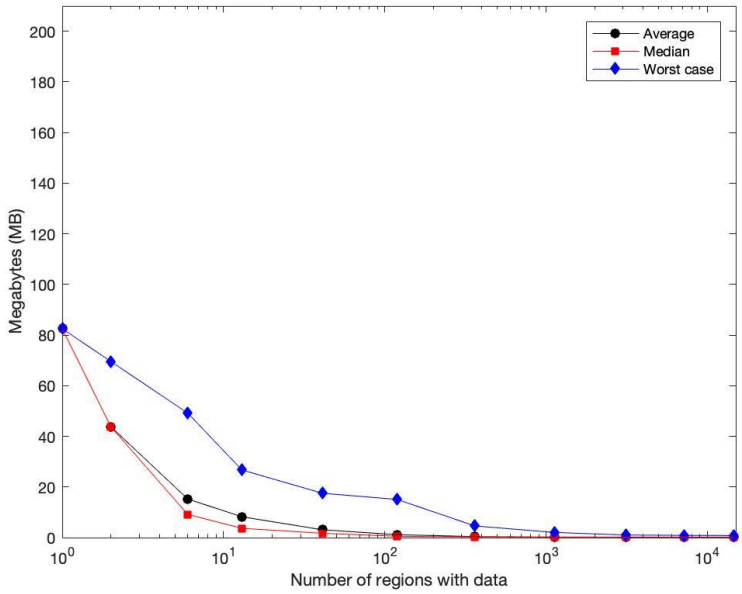

Figure 5.3: Region statistics test result plotted in graph for all zoom levels without attachments.

In figures 5.4 to 5.14 there are polygons drawn in maps that represent the regions containing data. These regions are the same regions that were considered in tables 5.2 and 5.3. By looking at the regions we see that all data is mostly located in the middle part of Sweden. Smaller regions better describe where the data is located, but not the amount of data that is stored in the regions. Comparing the zoom levels, it is clear that for lower zoom levels, unnecessary space on the map is covered by the regions. On higher zoom levels, these spaces are not covered. Having many smaller regions is desired from a disk size perspective. With smaller regions a user can subscribe to channels that covers the area of interest with more precision. However, having many regions affects the synchronisation time negatively.

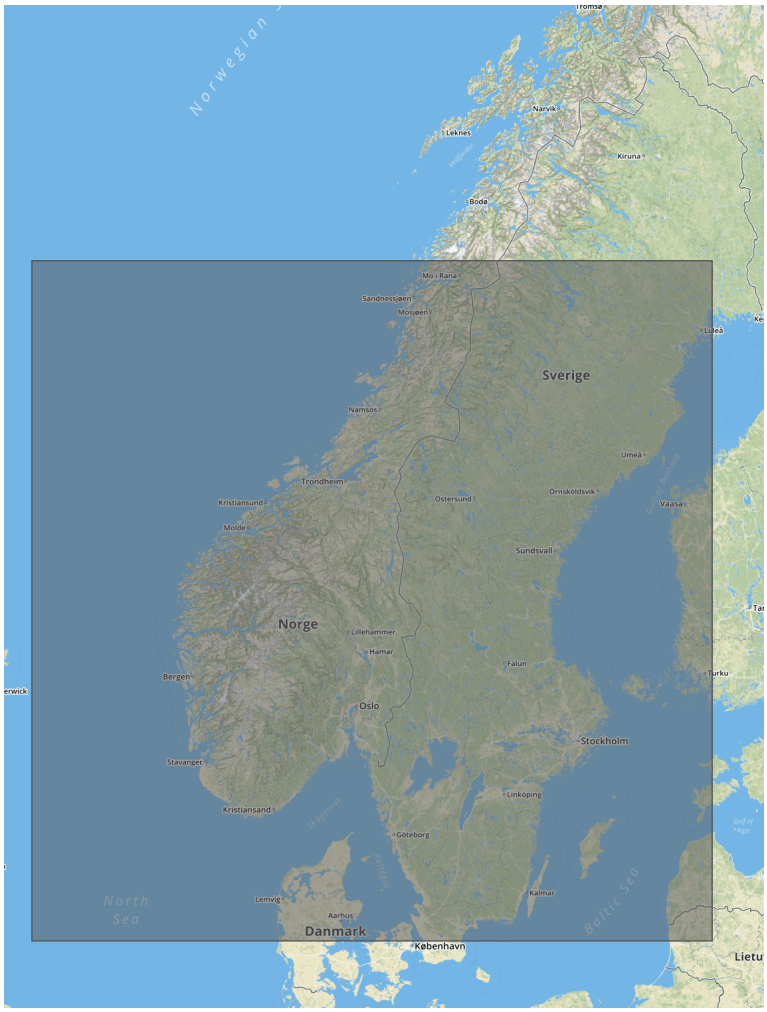

Figure 5.4: Regions for zoom level 4.

Figure 5.5: Regions for zoom level 5.

Figure 5.6: Regions for zoom level 6.

Figure 5.7: Regions for zoom level 7.

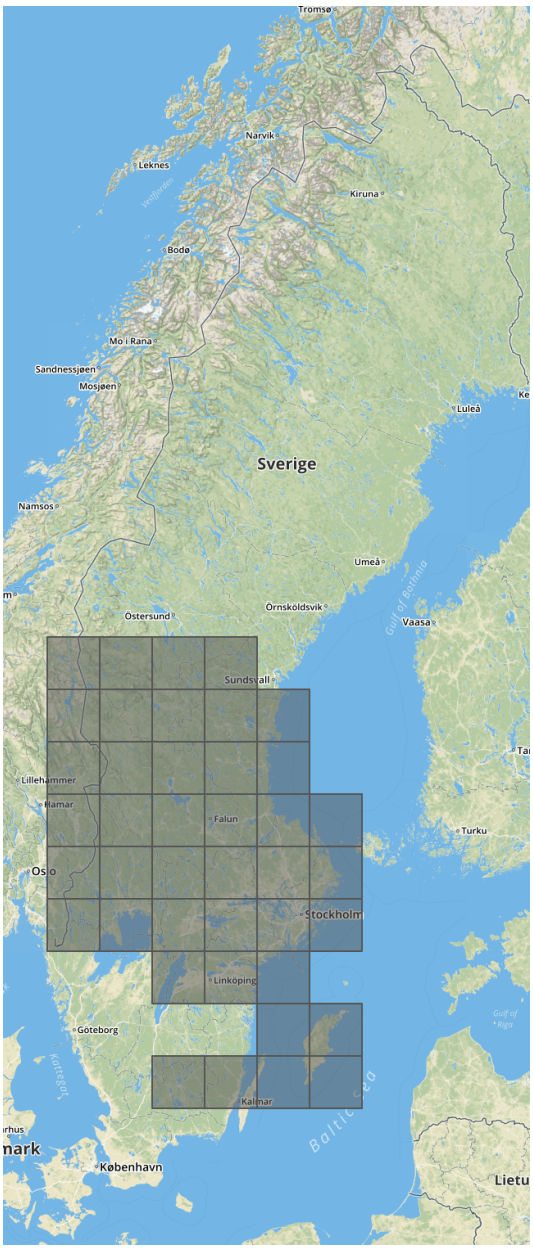

Figure 5.8: Regions for zoom level 8.

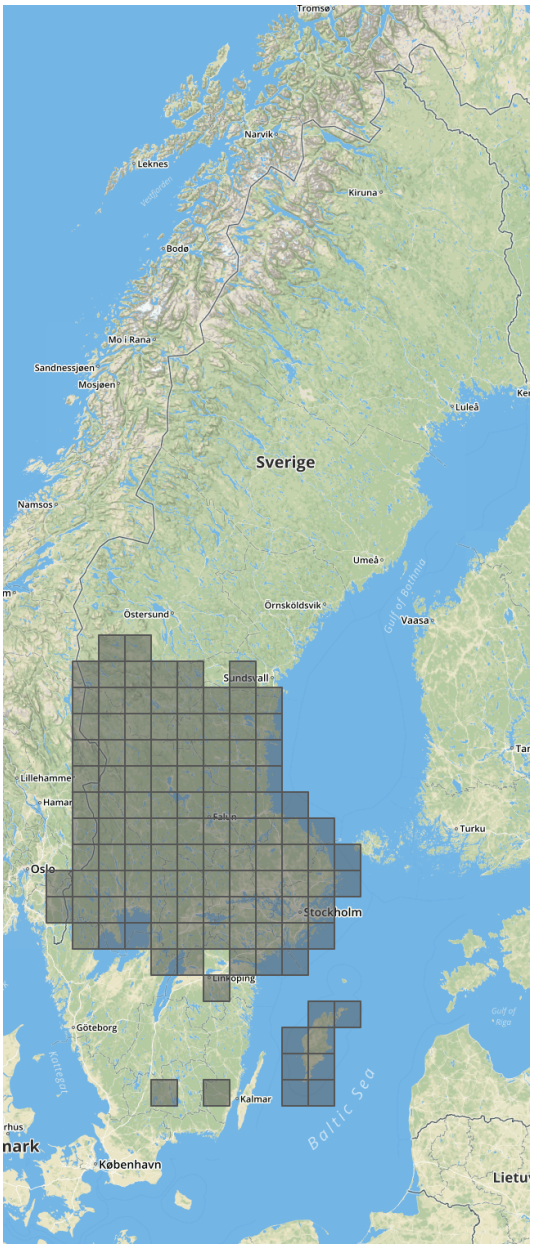

Figure 5.9: Regions for zoom level 9.

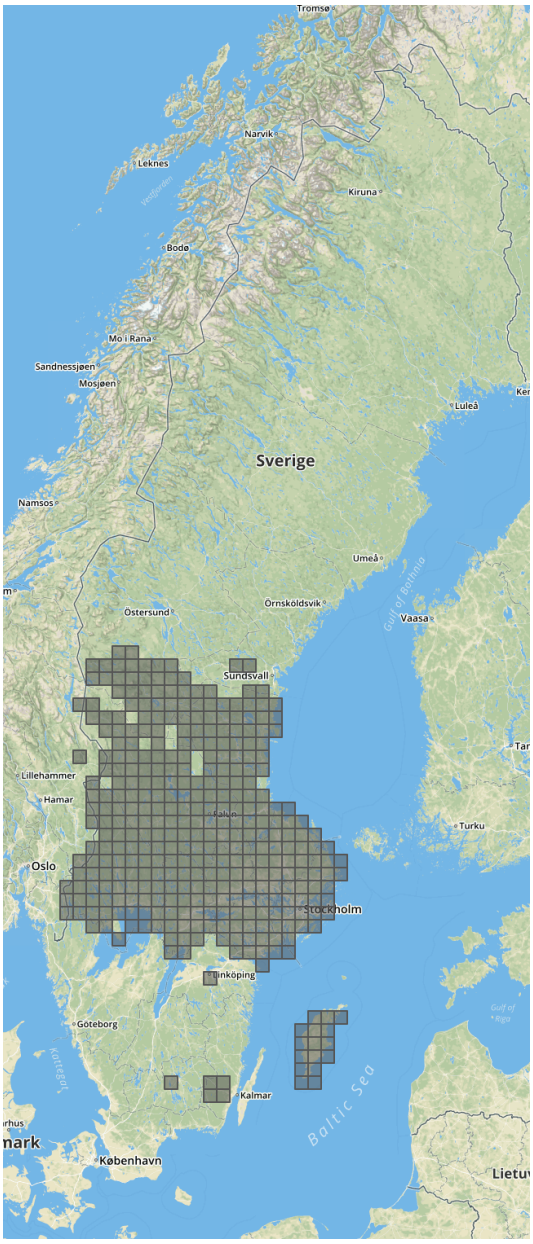

Figure 5.10: Regions for zoom level 10.

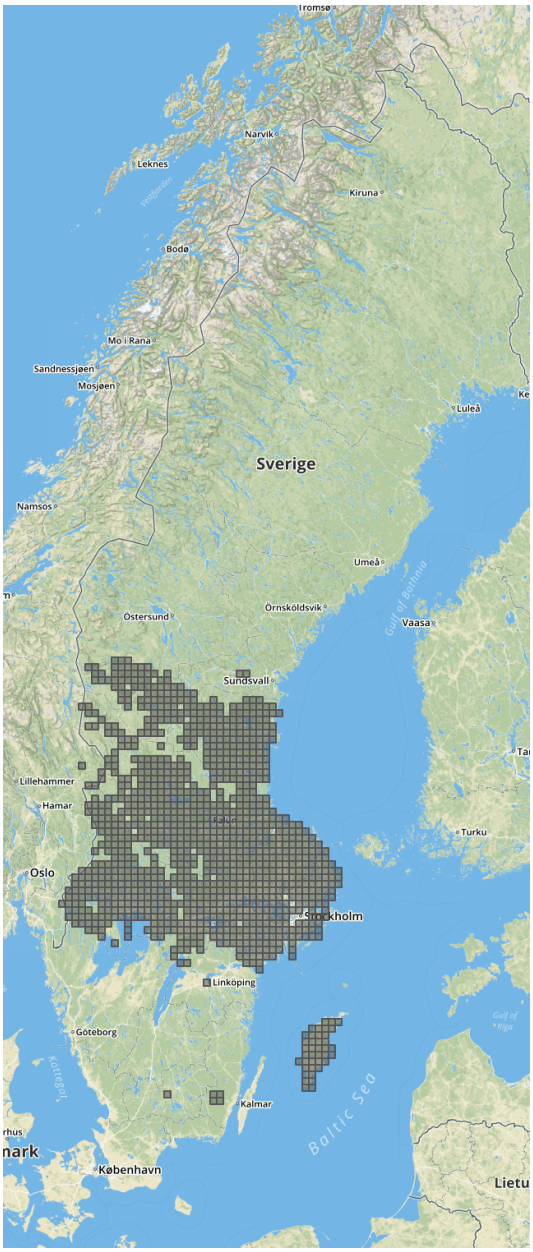

Figure 5.11: Regions for zoom level 11.

Figure 5.12: Regions for zoom level 12.

Figure 5.13: Regions for zoom level 13.

Figure 5.14: Regions for zoom level 14.

5.3 Disk size for users

The disk size requirements for the user defined as the most active user is presented in table 5.4 and in figure 5.15. This user is likely to be one of the users that requires most disk size on different zoom levels. From a disk size perspective there is no improvement between zoom level 4 and 5 and a very small improvement for zoom level 6, 7, and 8 regardless of whether attachments have been used . At zoom level 11, the user has reduced the required disk size by more than 70%. However, compared to the old approach with the SQLite database, zoom level 11 still requires more disk size if attachments are used. It is first at zoom level 12 that the user requires less disk size than the old approach which required 40.7 MB. If attachments are not used a zoom level of 11 is enough to get a lower disk size requirement compared to the old approach. Looking at the bar diagram in figure 5.15 it is clear that increasing the zoom level gives best improvements after zoom level 8. However, the difference of disk size requirement when using attachments and not using attachments decreases when the zoom level increases, just as the region statistics test showed. This pattern holds for all five users.

Zoom level	**Number of channels subscribed to**	**Disk size required with attachments**		**Disk size required without attachments**	
		Megabyte	**Percentage of total data disk size**	**Megabyte**	**Percentage of total data disk size**
4	1	206.02 MB	100%	82.48 MB	100%
5	2	206.02 MB	100%	82.48 MB	100%
6	5	205.05 MB	99.53%	82.37 MB	99.87%
7	11	204.96 MB	99.49%	82.10 MB	99.54%
8	29	201.59 MB	97.85%	81.50 MB	98.81%
9	51	155.70 MB	75.58%	63.64 MB	77.16%
10	74	93.35 MB	45.31%	41.43 MB	50.23%
11	95	47.33 MB	22.97%	21.27 MB	25.79%
12	142	29.99 MB	13.10%	14.84 MB	17.99%
13	234	18.28 MB	8.87%	10.69 MB	12.96%
14	450	15.78 MB	7.66%	7.67 MB	9.30%

Table 5.4: Disk size requirements for the most active user.

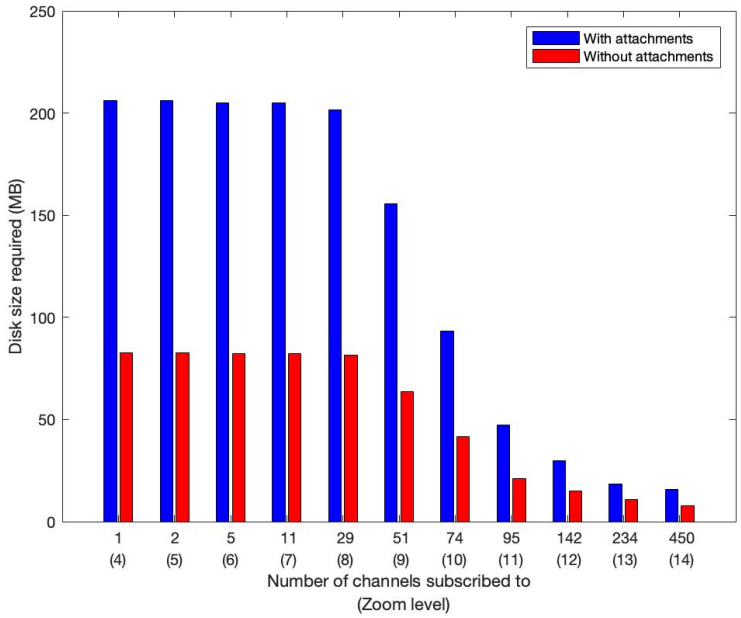

Figure 5.15: Bar diagram for disk size requirements for the most active user.

The average user's result is presented in table 5.5 and in figure 5.16. This user gets a better disk size requirement than the old approach at zoom level 11 and higher with attachments and zoom level 10 and higher without attachments. Even though the difference is just one zoom level, the disk size required without attachments is less than half at zoom level 10 than the required disk size with attachments at zoom level 11. An interesting thing shown for the average user is the decrease in disk size between zoom level 11 and 12 where there is only one more channel that is subscribed to. Subscribing to almost the same amount of channels when the zoom level is increased means that the regions were significantly larger than necessary. For example, if a user subscribes to ten channels on zoom level 8 and ten channels on zoom level 9, this means that three quarters of every region was unnecessary on zoom level 8. Higher zoom levels allow users to synchronise smaller regions that are more likely to contain less geometries that are not of interest.

Zoom level	Number of channels subscribed to	Disk size required with attachments		Disk size required without attachments	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	206.02 MB	100%	82.48 MB	100%
5	2	206.02 MB	100%	82.48 MB	100%
6	4	197.36 MB	95.80%	80.23 MB	97.27%
7	9	197.32 MB	95.78%	77.98 MB	94.54%
8	17	163.75 MB	79.48%	67.72 MB	82.10%
9	29	109.65 MB	53.22%	44.34 MB	53.76%
10	40	50.34 MB	24.43%	8.63 MB	10.46%
11	44	18.94 MB	9.19%	7.81 MB	9.47%
12	45	8.35 MB	4.05%	3.08 MB	3.73%
13	55	3.67 MB	1.78%	1.50 MB	1.82%
14	63	2.23 MB	1.13%	1.06 MB	1.29%

Table 5.5: Disk size requirements for the average user.

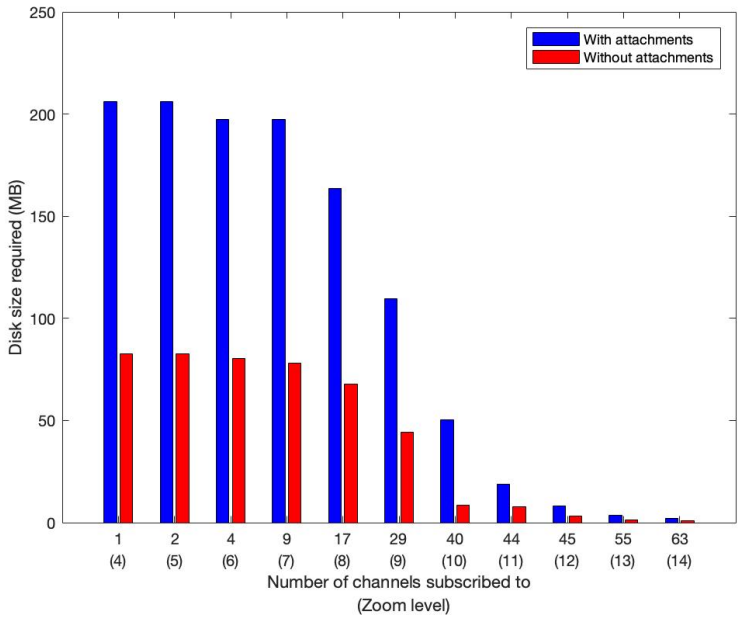

Figure 5.16: Bar diagram for disk size requirements for the average user.

The random users are presented in tables 5.6, 5.7, and 5.8. The respective bar diagrams for the users are presented in figures 5.17, 5.18 and 5.19. The lowest zoom level where random user one requires less disk size compared to the old approach is on zoom level 9 with attachments and zoom level 8 without attachments. Like the most active user and the average user it differs one zoom level. No other user tested benefits on zoom level 8 or lower. Comparing random user one with the other users, this user subscribes to few channels. This is either because the user is related to few geometries or that all geometries are densely located. The

former reason is more likely because the amount of disk size required is low for higher zoom levels. Random user two requires less disk size than the old approach when the zoom level is at least 11 with attachments and zoom level 9 without attachments. It is the only user where there are two zoom levels in difference. Compared to the rest of the users, random user two still has quite large difference between the approaches with attachments and without for the highest zoom levels. The values have not converged to similar values at zoom level 14 as it has for the other users. The average user and random user two are quite similar when looking at disk size required and the number of channels they subscribe to between zoom level 4 and 10. For increasing zoom levels, the number of channels subscribed to increase faster for random user two. Random user two represents those users that are less active then the most active user but more active then the average user. Random user three is the only user that required all data for the three first zoom levels. At the same time the user requires little disk size on zoom level 14. This could be because some geometries of interest is widely spread and the user is related to few geometries. Users in similar situation as random user three would not benefit much on lower zoom levels.

The result shows that all users benefit more when higher zoom levels are used. However, for the users that have more data, higher zoom levels are more beneficial when looking at the disk size requirements. When increasing zoom level from 11 to 14, active users often save more disk size measured in megabytes than less active users. Even though less active users can have a significant decrease factor in disk size, a factor of two can be more worth than a factor of ten. For example, it is better to go from 100 MB to 50 MB than to go from 1 MB to 0.1 MB. The active users do not benefit from higher zoom levels when looking at the number of channels they have to subscribe to. The number of channels increases fast while the disk size requirements start to stagnate with the exception for users like random user two where the stagnation start at higher zoom levels than 14. Not using attachments is beneficial for all users because the least required zoom level to get an improvement compared to the old approach is lowered. However, as the region statistics test also showed, for the highest zoom levels the difference in values between the approach of using attachments and not using it decreases.

Zoom level	Number of channels subscribed to	Disk size required with attachments		Disk size required without attachments	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	206.02 MB	100%	82.48 MB	100%
5	1	172.62 MB	83.79%	69.48 MB	84.24%
6	2	126.78 MB	61.54%	50.11 MB	60.75%
7	3	103.58MB	50.28%	44.33 MB	53.75%
8	4	58.85 MB	28.57%	26.79 MB	32.48%
9	6	37.62 MB	18.26%	18.37 MB	22.27%
10	6	11.89 MB	5.77%	7.80 MB	9.46%
11	6	6.05 MB	2.94%	2.64 MB	3.20%
12	8	2.02 MB	0.98%	1.02 MB	1.24%
13	9	0.91 MB	0.44%	0.48 MB	0.58%
14	10	0.41 MB	0.20%	0.25 MB	0.30%

Table 5.6: Disk size requirements for random user one.

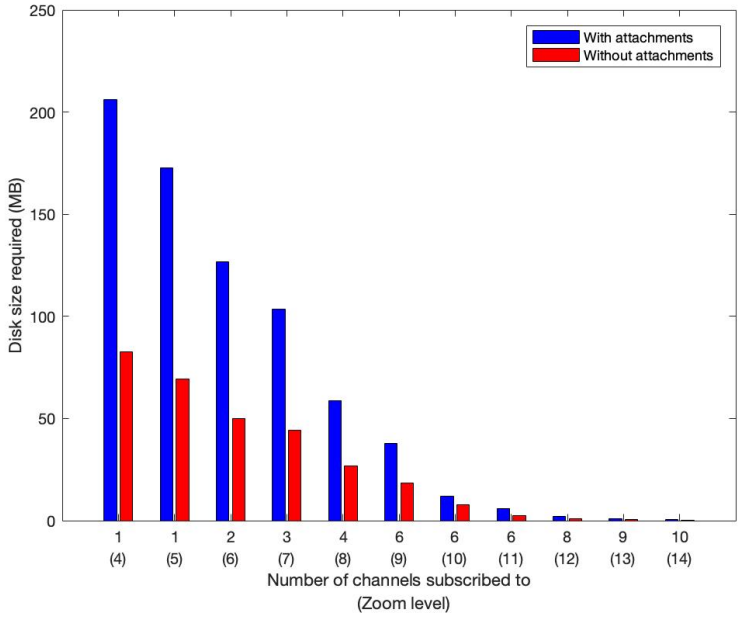

Figure 5.17: Bar diagram for disk size requirements for random user one.

Zoom level	Number of channels subscribed to	Disk size required with attachments		Disk size required without attachments	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	206.02 MB	100%	82.48 MB	100%
5	2	206.02 MB	100%	82.48 MB	100%
6	6	200.73 MB	97.43%	78.22 MB	94.84%
7	9	197.05 MB	95.65%	77.65 MB	94.14%
8	17	153.51 MB	74.51%	61.50 MB	74.56%
9	30	90.28 MB	43.82%	35.40 MB	42.92%
10	46	46.42 MB	22.53%	20.71 MB	25.11%
11	73	25.13 MB	12.20%	10.74 MB	13.02%
12	110	15.14 MB	7.35%	7.94 MB	9.63%
13	151	9.00 MB	4.37%	2.95 MB	3.58%
14	232	6.32 MB	3.07%	2.13 MB	2.58%

Table 5.7: Disk size requirements for random user two.

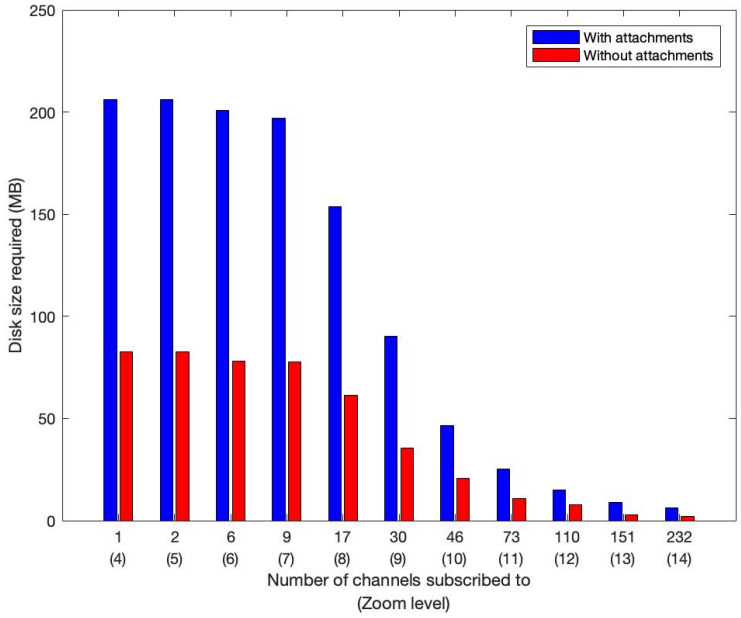

Figure 5.18: Bar diagram for disk size requirements for random user two.

Zoom level	Number of channels subscribed to	Disk size required with attachments		Disk size required without attachments	
		Megabyte	Percentage of total data disk size	Megabyte	Percentage of total data disk size
4	1	206.02 MB	100%	82.48 MB	100%
5	2	206.02 MB	100%	82.48 MB	100%
6	6	206.02 MB	100%	82.48 MB	100%
7	10	195.27 MB	94.78%	78.91 MB	95.67%
8	15	107.22 MB	52.04%	43.07 MB	52.22%
9	19	57.05 MB	27.69%	22.84 MB	27.69%
10	21	24.77 MB	12.02%	10.93 MB	13.25%
11	21	8.02 MB	3.89%	2.69 MB	3.26%
12	21	2.49 MB	1.21%	0.93 MB	3.26%
13	22	1.25 MB	0.61%	0.50 MB	0.61%
14	24	0.93 MB	0.45%	0.41 MB	0.50%

Table 5.8: Disk size requirements for random user three.

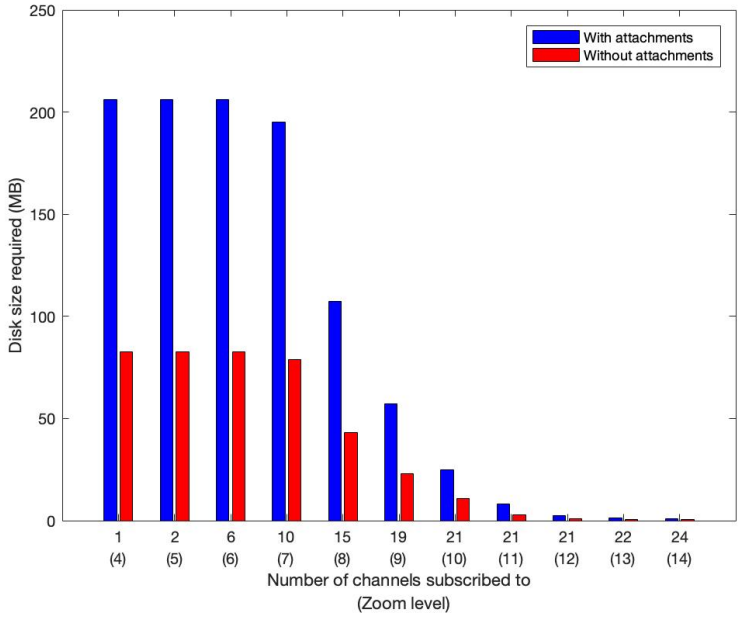

Figure 5.19: Bar diagram for disk size requirements for random user three.

6 Discussion

This chapter will tie the results together and compare them with the company's old approach to synchronise data between many mobile devices. Other techniques to solve the scalability problem with Couchbase are discussed and compared to the one used in this thesis. Validity and reliability of this study will be discussed.

6.1 Results

In this section the results from chapter 5 is discussed. The section is divided into two parts where the first part is a discussion about the tests and the last part is a discussion about the choice of whether attachments should be used or not.

6.1.1 Test results

One important goal with this thesis has been to reduce the amount of data that every user requires to store on their mobile devices. With the company's old solution, a client needs to store 40.7 MB data in total. An important factor that should be remembered when having BLOBs is the way they are stored in Couchbase. Because they are stored in separate files, attachments, and the file sizes were small, the total disk size increased from 40.7 MB to 206.02 MB. This is a significant drawback. When the BLOBs were stored as a string representation inside the documents the total disk size was 82.48 MB which is still more than the old approach. Because of this, partitioning data is a necessity to get the same or better result as the old approach when using a Couchbase solution. If disk size is the only factor of interest, the data set should be divided into as many regions as possible, according to the region statistics test. As Siau et al. [35] mentioned, data transferring increases the battery consumption and affects the data plan, which can be limited, from the mobile carrier. The amount of data that is transferred is reduced by having many small regions. However, it has been shown in the synchronisation time test that having many small regions affects the synchronisation time negatively, more so when attachments are used. A high synchronisation time is not desirable since the system should be practical. It is important to consider this trade-off when deciding which zoom level to use.

The synchronisation time test did show some unexpected results. The linear increase in time when attachments are used is good but whether the synchronisation time increase linearly after zoom level 14 or starts to grow exponentially cannot be known without testing higher zoom levels. The linear increase in time is not present when attachments are not used. This was unexpected since the same number of channels was used when testing with and without attachments. The number of channels a user subscribes to affects the synchronisation in different scale based on whether attachments are used or not. By looking at the result with attachments, a theory for the linear time increase was thought to be because the server and Sync Gateway had to iterate the user's channels and check if each document belonged to one of the channels. With higher zoom levels, a user's channel list and the document's channels list becomes larger since the regions are smaller and therefore more channels are subscribed to. However, even though the lists became larger, the test without attachments showed that this was not the reason for the time increase. Using attachments seems to increase the synchronisation time but it is the same number of attachments that is fetched for all zoom levels. The reason for the increase in time for the test with attachments cannot be explained by us. It is still an interesting finding. It is clear that it is a disadvantage from a time perspective to use attachments with many channels and this is a trade-off each developer has to think about. It is not always better to choose an approach without attachments, see section 6.1.2.

It is worth mentioning that it does not have to be a linear increase in time between zoom level 4 to 14 for the test with attachments. This is dependent on the number of channels that are available. Depending on how many objects there are in the data set and where the objects' geographical positions are, the number of channels for a zoom level can differ from what we have seen in the results in this thesis. Running the test on each zoom level twice to generate ten results was shown to be useful. For example, on zoom level 14 there was a time difference of almost 15% between two of the values for the test with attachments, see table B.11 in appendix. The difference is believed to be caused mainly by the current load on CPU and the network traffic.

The region statistics test showed that in average, using zoom levels with regions to partition data was effective. This was especially true for lower zoom levels where the improvement was large for increasing zoom levels. However, keeping an eye on the median and worst case is important to understand that the average is not representative. This was also shown in the disk size for users test. By looking at the average disk size requirement in the region statistics test, a zoom level greater than 11 does not give much improvement no matter whether attachments were used or not. This fact did not conform with the result from the disk size for users test. That test showed that zoom levels greater than 11 were necessary when using attachments in order to achieve a lower disk size requirement for all users compared to the old approach. The reason the tests do not conform is because some users subscribe to channels that correspond to regions that require more disk size than the average regions. Having both tests gives a better overview of what zoom level might be suitable. The results are data dependent and would vary with other data sets. To find a suitable zoom level, it is recommended to run at least similar tests as the region statistics test and the disk size for users test to get a good overview of which zoom level to use.

6.1.2 With or without attachments

The results in this thesis shows that it is favourable to not use attachments. The reason for the improvements in the region statistics test and the disk size for users test are that there are many files that are much smaller than 4096 bytes and they take up more disk space than the file size. Using a string representation inside the documents instead of storing the objects as attachments might not always give a better result. If the geometries are very small in file size, like in this thesis, it is recommended to not use attachments. If instead other objects

are used where the files are of larger size, using attachments can be desired. In this thesis base64 was used as the string representation. According to Isenburg et al. [23], using base64 encoding instead of a binary representation of an object increase the size with about 33%. It is important to consider this because if files are large, using a string representation would probably mean larger disk size requirements.

6.2 Method

In this section the method in chapter 4 is discussed. This section discusses different choices that have been made in the method. First, several options to partition the data are discussed. Then comes discussion about what would have happened with the result if higher zoom levels had been used and if not *null* values had been taken into account. Finally, a small section about source criticism is presented.

6.2.1 Partitioning of data

As mentioned in section 4.1, the first idea to solve the scalability problem was to partition the data into several data sets and have one database for each set. With this idea, users can subscribe to databases of interest. However, the need to handle several databases makes this idea more demanding. Every time a user would connect to the application, the application would have to open and manage multiple connections to databases. With Couchbase channels, this is simplified since the only connection that is needed is the one to Sync Gateway.

With Couchbase channels, the data can be stored in a single database on the server. Having a single database could be inappropriate if large amount of data is stored or if the workload on the server is high. With several databases, as the first idea proposes, the data and the workload is split between multiple databases. However, existing database solutions often have functionality to split the workload on the server by distributing data into several server instances. Most databases also have support to store large amount of data.

The approach of partitioning data is based on the same idea as the One-per publication (op) design presented by Yee et al. [41]. They did suggest a One-giant datagroup (og) design when the bandwidth was good, which it is in this case. However, an og approach would not have been suitable for this case because the disk size usage on the mobile will not decrease with this approach. The Client-centric (cc) design would not be beneficial either because there would be overhead of updating all databases for each client. The same data would be stored in several databases and this would require the server to do more work.

6.2.2 Splitting of regions

There are many available methods of how regions could be defined. This section will discuss some of them, what advantages and drawbacks they come with and also how they could have affected the result.

Google's tile-based mapping system has been used in this thesis to divide the map into several regions. The motivation for using this mapping system was that the company is using the same mapping system for their application to render interactive maps. Using this mapping system to define regions would make it easier to decide which channels to subscribe to. The reason for this is because when a user is panning around in the map, the information of which tile that the user is looking at can be extracted. Since each region maps to a single tile and there is a channel for every region, channels can automatically be subscribed to based on how the application is used by the user. However, using Google's mapping system comes with a limitation. Regions are predefined for every zoom level which removes the opportunity to decide how many regions to use.

R-tree would be an alternative to Google's tile-based mapping system. With an R-tree approach, regions of different sizes would be defined. An R-tree approach would be the most dynamical mapping solution since the regions will continuously change when geometries are added, removed, or modified. One advantage with R-trees is that it removes the problem of deciding the number of regions that is required. This is handled by R-tree because it is automatically balanced when elements are added, deleted, and modified. The number of regions would be decided by the number of nodes with at least one leaf. Having a dynamical mapping solution would be demanding because every time a change occurs, the R-tree would be rebalanced. This behaviour means that the defining of regions would continuously change and the documents on the server would need to be updated.

A mapping solution that is not as dynamic as the R-tree is a Voronoi diagram. With a Voronoi diagram approach, the definition of the regions would not be continuously changed, as in the R-tree approach. A Voronoi approach would define regions of different sizes and shapes based on the initial points chosen to generate the Voronoi diagram. An advantage over Google's mapping system is that the number of regions to use can be decided. The number of regions is equal to the number of points that is used when building the Voronoi diagram. One way to build a diagram would be to use geometries from the data set. Since the diagram is built on points and not geometries, the center point of the bounding box for the geometry could be used. If the geometries to build the diagram are randomly chosen, it is likely that many of these are chosen from locations with many geometries. This would generate a Voronoi diagram with smaller regions at locations with many geometries and larger regions at locations with fewer geometries. Having regions defined like this would distribute the amount of data more evenly across the regions. The worst case region in the region statistics test would probably be closer to the average region.

With Voronoi diagrams, several problems arises. One problem is that it only supports points and not geometries. This means that it is hard to decide which region a geometry belongs to. One way could be to use the geometry's center point. For geometries that reside in more than one region this would lead to the problem that a user subscribing to a channel might not see the correct geometries at the border of the corresponding region. This behaviour of the system might not be desirable. Another problem with this mapping solution is to decide which points that should be used when building the Voronoi diagram. This would have significant effects on the result since the diagram is used to define regions. Google's mapping system approach makes it convenient to decide how channels should be automatically subscribed to when a user is panning around in the map. With Voronoi based mapping, this will become an issue. There is no simple way to automatically know which regions that the application is showing to the user. An ineffective way of solving this issue would be to iterate over all regions and check if one of the regions reside in the area that is currently displayed by the application. The problem that R-tree has with continuously updating the data on the server is partially a problem with the Voronoi approach as well. Because the Voronoi diagram would be built from a snapshot of the data, the diagram could become non-representative of the data set when a lot of updates has occurred. To keep the diagram up to date it should periodically be recreated, which leads to the need to update documents on the server. However, compared to R-tree mapping, this does not need to done for every change in the data set.

Another approach that could have been used to define regions is an approach referred to as Google's tile-based mapping system with adjustments. This approach would use Google's tile based mapping system but with different zoom levels for different regions. This allows to define the regions so that the data is more evenly distributed. Like the Voronoi based mapping, one drawback is that when data is changed the regions might not be optimal and would thus require to be recreated. The idea of this approach is to divide some of the regions

generated with Google's mapping system into higher zoom levels. This would result in a design that consists of tiles of different sizes, see figure 6.1.

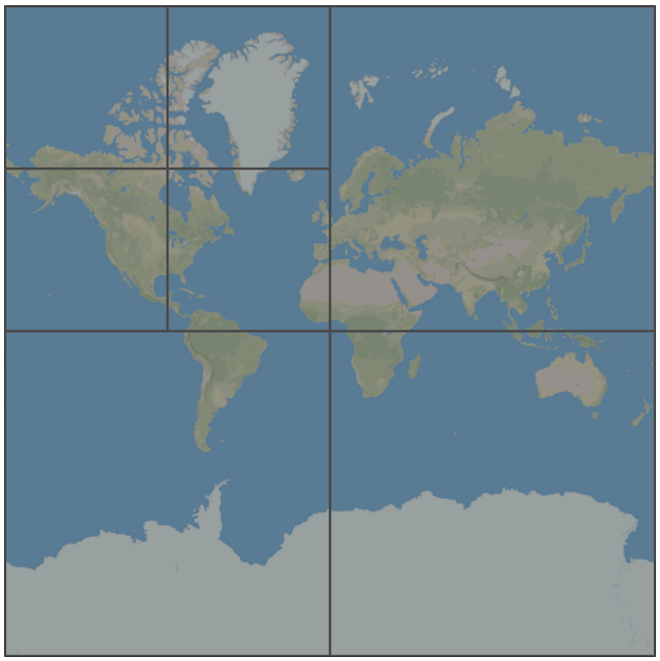

Figure 6.1: Google's tile-based mapping system with adjustments.

Regions could also have been defined by real world borders for cities or provinces. This approach requires to technically define the borders and functionality to decide which region a geometry belongs to. This makes it more complex and probably more time consuming compared to the other approaches. As the Voronoi and the R-tree approach, this approach will probably also struggle with the problem to automatically decide which channels to subscribe to when the user is panning around in the map. In some cases when users only operate in certain cities or provinces, we believe that this approach is suitable.

6.2.3 Improvements and further testing

The schema free architecture of a NoSQL database is mentioned as an advantage by several papers as it allows for flexibility [32, 16, 11, 8]. However, we do not make use of the schema free architecture in this work, instead *null* values from the SQLite database was included as an empty string in the Couchbase document. As mentioned in section 4.3.1, the reason for this is because we want to keep the same structure of the records in order to make the disk size comparison between the old SQLite solution and the new Couchbase solution. In practice, these *null* values should not have been included since a NoSQL database is schema free and there are no advantages of keeping them as empty strings. The *null* values are affecting all three tests' results. Having these *null* values increase the disk size for each documents. The time it took to synchronise all data in the synchronisation time test will probably be affected as well since the amount of data that needs to be synchronised is larger. We believe that without the *null* values, the amount of disk size required in the region statistics test and the disk size for users test would be decreased. How much it would affect the results is hard to predict.

Another choice that was made was that all three tests should be run on zoom level between 4 and 14, including those. The reason for not running on zoom levels higher then 14 was

because it took to much time to complete the synchronisation time test. It would be interesting to run tests above zoom level 14 to see how the disk size would be affected. According to the figures 5.2 and 5.3, the disk sizes converge to specific values. Our theory is that after a certain zoom level the average and median disk size would slightly increase. We believe this would happen because when the regions become small enough many geometries starts to overlap multiple regions. The total disk size would increase faster than the total number of regions since the disk size for each geometry is counted for each region. When it comes to the disk size for users test we believe that higher zoom levels will have minor affect. The disk size will not decrease much with the used data set since the result already shows that the improvement is small between zoom level 13 and 14. We think that it is not worth to increase the zoom level after a certain point. This zoom level does not have to be 14, it can be lower or higher.

6.2.4 Source criticism

To find relevant scientific research and books for this thesis, Google scholar[1], IEEEXplore[2], and ACM[3] was used. Much of the information regarding Couchbase is from the official documentation of Couchbase[4]. The choice to use this documentation was made because it is the primary source for the technical features. We have been aware that the documentation can include subjective information and have had this in mind when choosing what information to use.

6.3 Validity and reliability

One validity concern in this thesis is regarding the method for the synchronisation time test. Sync Gateway was set to log all messages during synchronisation which can affect the time it takes to synchronise. This was not seen as a problem since the same settings were used for all zoom levels with or without attachments. However, the number of logged messages might be affected based on whether attachments were used or not. Another concern is that validity relies on the authors ability to interpret and analyse the results. No previous research was found that could be used to interpret the results in this thesis. For example, the behaviour observed in the synchronisation time test could not be explained by us.

Whether the same results would be achieved if the study was repeated with the same method is heavily based on the data set that is used. If the same data set is used, similar result would be found. However, the exact disk size required would differ by some bytes. The reason for this is because each test run shows different number of bytes required. This is thought to be caused by internal properties of Couchbase. Even though some bytes differ, the differences are too insignificant to affect the findings and conclusions in this thesis. The result would also be affected if hardware with another disk sector size was used. If another data set was used, it is likely that the result would differ but that the conclusions would hold. For example, a larger data set would generate much larger values for the data disk size requirements but the pattern that was observed for increasing zoom levels would still be similar. At last, it is important to acknowledge that all results are interpreted by us and that there is a possibility that people with another background would interpret the result differently.

[1] https://scholar.google.se
[2] https://ieeexplore.ieee.org
[3] https://dl.acm.org
[4] https://docs.couchbase.com/home/index.html

6.4 The work in a wider context

Aijaz et al. [4] conclude that it is expected that mobile data offloading will become important to reduce data traffic on mobile networks. Data offloading is defined as the usage of complementary networks instead of mobile networks, like WiFi. This thesis does not study data offloading but it presents an approach that reduces data traffic, which is the intended goal of data offloading. The authors discuss several business and commercial benefits for data offloading which can be used in this thesis to discuss how this work is related to ethical or societal aspects. They say that due to lesser congestion over mobile network, resources can be saved since the need for upgrading access points will reduce. The authors further mention that fewer access points can be used when offloading data which will save energy. Even though the presented approach in this thesis does not have the same impact to reduce data traffic as data offloading, it is a good way to reduce it through software development.

7 Conclusion

The main aim of this thesis has been to evaluate Couchbase as a tool to solve a scalability problem where large amount of data was shared between many mobile devices. The data had to have the requirement that it could be partitioned based on geographical positions. This thesis also includes a suggestion to an approach to partition the data and discussions for other approaches.

The synchronisation time test showed that the number of channels that a user subscribed to affected the synchronisation time more when attachments were used compared to when they were not. This leads to a trade-off developers have to consider when choosing zoom levels if attachments are used. Both the region statistics test and the disk size for users test showed that the approach implemented in this thesis was effective, more so without attachments. A higher zoom level leads to a better disk size requirement, but it converges to a fixed value which means that it is not necessary to pick the highest possible zoom level.

Couchbase has been shown to be a viable option to solve the scalability problem that has been under focus. Channels have been a solid feature to keep the advantages of having all data gathered in one place and at the same time partition the data into several data sets. Using a tile based approach to create regions is suggested because of the simple ways to determine borders and intersections with other objects. To use a non-relational database has its advantages but it has been shown that using a Couchbase solution without partitioning the data will increase the disk size requirements when migrating from an SQLite database.

It is not feasible to determine a zoom level that is best suited for all applications since this relies on the data set used. For the data set used in this thesis a zoom level of 14 with no attachments is suggested. The synchronisation time test deterioration for increasing zoom levels was insignificant when attachments were not used and because of that, using the highest zoom level tested would be recommended. For other data sets, as discussed, it is recommended to rerun the tests to find suitable zoom levels and determine whether attachments are necessary or not. Since the approach evaluated in this thesis would lead to each user synchronising less data, the network traffic and the battery consumption is likely to decrease for the application. In order to determine whether to use Couchbase in a production environment, a thorough study regarding the synchronisation features has to be done. Since different

companies require different synchronisation features, knowing that Couchbase can be used to reduce the amount of data that is synchronised might not be sufficient.

7.1 Future work

For future work it would be interesting to evaluate Couchbase's synchronisation features in depth. Features like automatic conflict detection and resolution are not evaluated in this thesis but they are important. For example, being able to customise how conflicts are resolved is a requirement for the company mentioned in this thesis. In the Discussion chapter, different methods of partitioning data were discussed. Google's tile-based mapping system with adjustment would be interesting to implement and test because of the advantages that it comes with. There might be more good alternatives to study and test for future work. Even though the aim of this thesis was to evaluate Couchbase, the scalability problem can be solved with other back-end tools. One alternative would be to also evaluate Simba which was mentioned in this thesis.

One of the reasons to solve the scalability problem was to decrease the amount of traffic and battery consumption. Because of limited time these have not been measured. It requires a lot of work to migrate to a Couchbase solution and therefore it can be worth measuring battery consumption and data traffic before deciding to use the presented approach.

Bibliography

[1] *Admin REST API | Couchbase Docs*. https://docs.couchbase.com/sync-gateway/2.1/admin-rest-api.html. (Accessed on 04/10/2019).

[2] S. Agarwal, D. Starobinski, and A. Trachtenberg. "On the scalability of data synchronization protocols for PDAs and mobile devices". In: *IEEE Network* 16.4 (July 2002), pp. 22–28. ISSN: 0890-8044. DOI: 10.1109/MNET.2002.1020232.

[3] Nitin Agrawal, Akshat Aranya, and Cristian Ungureanu. "Mobile Data Sync in a Blink". In: *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. San Jose, CA: USENIX, 2013. URL: https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/Agrawal.

[4] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. "A survey on mobile data offloading: technical and business perspectives". In: *IEEE Wireless Communications* 20.2 (2013), pp. 104–112.

[5] S. Amghar, S. Cherdal, and S. Mouline. "Which NoSQL database for IoT Applications?" In: *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*. June 2018, pp. 131–137. DOI: 10.1109/MoWNet.2018.8428922.

[6] *Authentication | Couchbase Docs*. https://docs.couchbase.com/sync-gateway/2.1/authentication.html. (Accessed on 02/11/2019).

[7] *Authorizing Users | Couchbase Docs*. https://docs.couchbase.com/sync-gateway/2.1/authorizing-users.html. (Accessed on 02/11/2019).

[8] Cristina Băzăr and Cosmin Sebastian Iosif. "The Transition from RDBMS to NoSQL. A Comparative Analysis of Three Popular Non-Relational Solutions: Cassandra, MongoDB and Couchbase". In: *Database Systems Journal* 5.2 (2014), pp. 49–59.

[9] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. "The R*-tree: an efficient and robust access method for points and rectangles". In: *Acm Sigmod Record*. Vol. 19. 2. Acm. 1990, pp. 322–331.

[10] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. "Concurrency control and recovery in database systems". In: (1987).

[11] Uma Bhat and Shraddha Jadhav. "Moving towards non-relational databases". In: *International Journal of Computer Applications* 1.13 (2010), pp. 40–46.

[12] Tim Bray. *The javascript object notation (json) data interchange format*. 2017.

[13] Martin C Brown. *Getting Started with Couchbase Server: Extreme Scalability at Your Fingertips*. " O'Reilly Media, Inc.", 2012.

[14] *Buckets | Couchbase Docs*. https://docs.couchbase.com/server/6.0/learn/buckets-memory-and-storage/buckets.html. (Accessed on 02/05/2019).

[15] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, and Tim Schaub. *The GeoJSON format*. 2016.

[16] Rick Cattell. "Scalable SQL and NoSQL data stores". In: *Acm Sigmod Record* 39.4 (2011), pp. 12–27.

[17] *Couchbase Lite in Swift | Couchbase Docs*. https://docs.couchbase.com/couchbase-lite/2.1/swift.html#getting-started. (Accessed on 02/07/2019).

[18] *Data Routing | Couchbase Docs*. https://docs.couchbase.com/sync-gateway/2.1/data-routing.html. (Accessed on 02/14/2019).

[19] S. Farrugia. "Mobile Cloud Computing Techniques for Extending Computation and Resources in Mobile Devices". In: *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. Mar. 2016, pp. 1–10. DOI: 10.1109/MobileCloud.2016.26.

[20] Michael E Fitzpatrick. *4K Sector Disk Drives: Transitioning to the Future with Advanced Format Technologies*. https://storage.toshiba.com/docs/services-support-documents/toshiba_4kwhitepaper.pdf. (Accessed on 04/29/2019). 2011.

[21] Steven Fortune. "A sweepline algorithm for Voronoi diagrams". In: *Algorithmica* 2.1-4 (1987). Note: contains general information about Voronoi diagrams, p. 153.

[22] Robin Hecht and Stefan Jablonski. "NoSQL evaluation: A use case oriented survey". In: *2011 International Conference on Cloud and Service Computing*. IEEE. 2011, pp. 336–341.

[23] Martin Isenburg and Jack Snoeyink. "Binary compression rates for ASCII formats". In: *Proceedings of the eighth international conference on 3D Web technology*. ACM. 2003, 173–ff.

[24] Yeolib Kim, Daniel A. Briley, and Melissa G. Ocepek. "Differential innovation of smartphone and application use by sociodemographics and personality". In: *Computers in Human Behavior* 44 (2015), pp. 141–147. ISSN: 0747-5632. DOI: https://doi.org/10.1016/j.chb.2014.11.059. URL: http://www.sciencedirect.com/science/article/pii/S0747563214006694.

[25] N. Leavitt. "Will NoSQL Databases Live Up to Their Promise?" In: *Computer* 43.2 (Feb. 2010), pp. 12–14. ISSN: 0018-9162. DOI: 10.1109/MC.2010.58.

[26] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. "Mobile Data Offloading: How Much Can WiFi Deliver?" In: *IEEE/ACM Trans. Netw.* 21.2 (Apr. 2013), pp. 536–550. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2218122. URL: http://dx.doi.org.e.bibl.liu.se/10.1109/TNET.2012.2218122.

[27] *Lite | Couchbase*. https://www.couchbase.com/products/lite. (Accessed on 02/07/2019).

[28] Zach McCormick and Douglas C Schmidt. "Data synchronization patterns in mobile application design". In: *Proceedings of the 19th Conference on Pattern Languages of Programs*. The Hillside Group. 2012, p. 12.

[29] *Nodes | Couchbase Docs*. https://docs.couchbase.com/server/6.0/learn/clusters-and-availability/nodes.html. (Accessed on 01/31/2019).

[30] David Ostrovsky, Yaniv Rodenski, and Mohammed Haji. *Pro couchbase server*. Apress, 2015, pp. 4–7.

[31] Miklós Pál and Gábor Láner. "Mobile Data Synchronization Methods". In: *Hungarian Journal of Industry and Chemistry* 44.2 (2016), pp. 93–98.

[32] Henry Potsangbam. *Learning Couchbase*. Note: also contains a description of NoSQL. Packt Publishing Ltd, 2015, pp. 2–4.

[33] Peter J Rousseeuw and Mia Hubert. "Robust statistics for outlier detection". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (2011), pp. 73–79.

[34] John T. Sample and Elias Ioup. *Tile-Based Geospatial Information Systems: Principles and Practices*. Springer Publishing Company, Incorporated, 2014, pp. 2–6. ISBN: 1489999728, 9781489999726.

[35] Keng Siau and Zixing Shen. "Mobile communications and mobile services". In: *International Journal of Mobile Communications* 1.1-2 (2003), pp. 3–14.

[36] R Smith. *Linux on 4KB-sector disks: Practical advice*. 2010.

[37] *Swift - Apple Developer*. `https://developer.apple.com/swift/`. (Accessed on 05/09/2019).

[38] *Sync Gateway | Couchbase Docs*. `https://docs.couchbase.com/sync-gateway/1.4/`. (Accessed on 02/07/2019).

[39] *vBuckets | Couchbase Docs*. `https://docs.couchbase.com/server/6.0/learn/buckets-memory-and-storage/vbuckets.html`. (Accessed on 01/31/2019).

[40] J. Wang and D. Zhang. "Research and Design of Distributed Database Synchronization System Based on Middleware". In: *2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*. June 2015, pp. 685–688. DOI: `10.1109/ICICTA.2015.174`.

[41] Wai Gen Yee and Ophir Frieder. "Scalable Synchronization of Intermittently Connected Database Clients". In: *Proceedings of the 6th International Conference on Mobile Data Management*. MDM '05. Ayia Napa, Cyprus: ACM, 2005, pp. 299–303. ISBN: 1-59593-041-8. DOI: `10.1145/1071246.1071295`. URL: `http://doi.acm.org/10.1145/1071246.1071295`.

A Couchbase Server settings

This appendix includes figures showing settings and configurations that were used for Couchbase Server instance.

A.1 Couchbase Server configuration settings

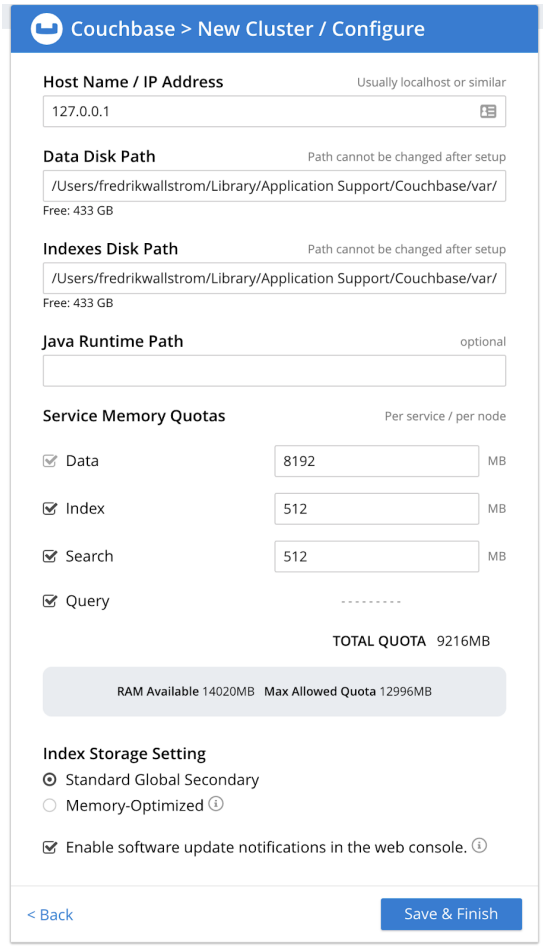

Figure A.1: Complete settings for the server instance.

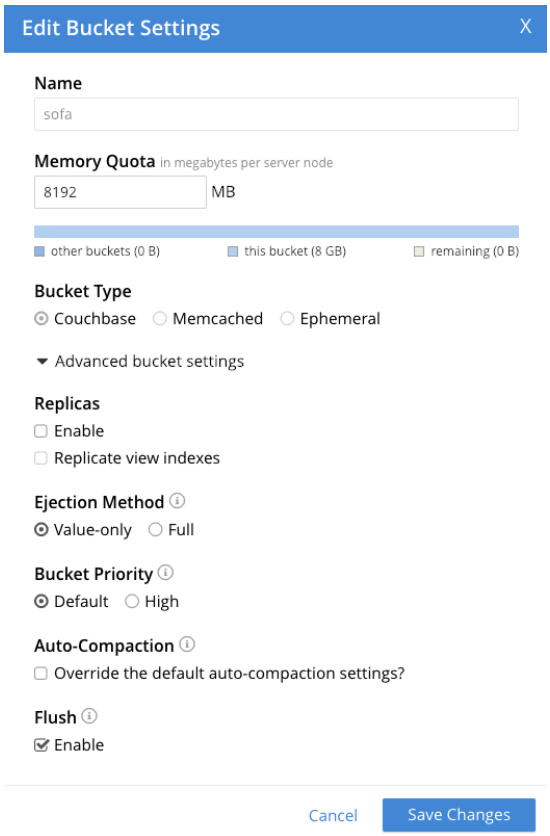

Figure A.2: Complete settings for the bucket.

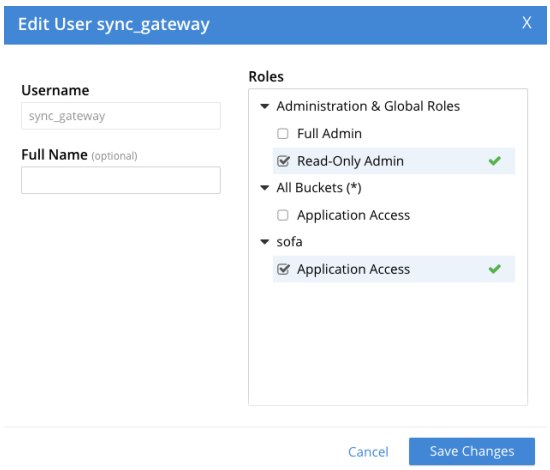

Figure A.3: Complete settings for the RBAC user.

B Synchronisation time

This appendix includes results related to the synchronisation time tests.

B.1 Complete result of synchronisation time test with attachments

Zoom level 4 (1 region)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
42.47 Seconds	41.80 Seconds
42.22 Seconds	41.92 Seconds
42.06 Seconds	39.39 Seconds
41.67 Seconds	42.95 Seconds
41.09 Seconds	42.43 Seconds

Table B.1: Synchronisation time test result for zoom level 4 with attachments.

Zoom level 5 (2 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
39.20 Seconds	42.13 Seconds
41.37 Seconds	40.96 Seconds
41.95 Seconds	41.18 Seconds
42.71 Seconds	43.30 Seconds
40.46 Seconds	39.12 Seconds

Table B.2: Synchronisation time test result for zoom level 5 with attachments.

Zoom level 6 (6 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
40.90 Seconds	41.95 Seconds
42.31 Seconds	42.24 Seconds
40.08 Seconds	40.25 Seconds
42.29 Seconds	43.13 Seconds
43.49 Seconds	43.39 Seconds

Table B.3: Synchronisation time test result for zoom level 6 with attachments.

Zoom level 7 (13 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
43.12 Seconds	39.83 Seconds
40.21 Seconds	44.01 Seconds
41.52 Seconds	43.45 Seconds
42.05 Seconds	40.31 Seconds
42.04 Seconds	41.76 Seconds

Table B.4: Synchronisation time test result for zoom level 7 with attachments.

Zoom level 8 (41 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
41.35 Seconds	39.53 Seconds
43.66 Seconds	41.27 Seconds
43.66 Seconds	44.82 Seconds
46.40 Seconds	43.17 Seconds
42.18 Seconds	43.44 Seconds

Table B.5: Synchronisation time test result for zoom level 8 with attachments.

Zoom level 9 (119 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
43.51 Seconds	45.62 Seconds
43.83 Seconds	46.16 Seconds
44.80 Seconds	43.06 Seconds
43.99 Seconds	43.38 Seconds
41.22 Seconds	44.62 Seconds

Table B.6: Synchronisation time test result for zoom level 9 with attachments.

Zoom level 10 (362 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
50.73 Seconds	57.59 Seconds
48.66 Seconds	54.90 Seconds
48.65 Seconds	56.53 Seconds
46.93 Seconds	54.04 Seconds
48.53 Seconds	55.78 Seconds

Table B.7: Synchronisation time test result for zoom level 10 with attachments.

Zoom level 11 (1124 regions)	
Forenoon - 27/2/2019	Afternoon - 28/2/2019
82.79 Seconds	95.13 Seconds
90.69 Seconds	97.93 Seconds
93.49 Seconds	101.95 Seconds
91.81 Seconds	106.49 Seconds
91.07 Seconds	101.95 Seconds

Table B.8: Synchronisation time test result for zoom level 11 with attachments.

Zoom level 12 (3134 regions)	
Forenoon - 01/3/2019	Afternoon - 27/2/2019
186.48 Seconds	190.72 Seconds
197.28 Seconds	196.33 Seconds
202.25 Seconds	196.38 Seconds
201.70 Seconds	197.80 Seconds
209.64 Seconds	184.84 Seconds

Table B.9: Synchronisation time test result for zoom level 12 with attachments.

Zoom level 13 (7220 regions)	
Forenoon - 28/2/2019	Afternoon - 27/2/2019
530.45 Seconds	448.67 Seconds
470.93 Seconds	444.74 Seconds
462.58 Seconds	448.80 Seconds
455.77 Seconds	460.14 Seconds
465.97 Seconds	479.01 Seconds

Table B.10: Synchronisation time test result for zoom level 13 with attachments.

Zoom level 14 (14623 regions)	
Forenoon - 28/2/2019	Afternoon - 27/2/2019
953.11 Seconds	1063.97 Seconds
955.15 Seconds	1063.51 Seconds
894.94 Seconds	1048.84 Seconds
926.87 Seconds	1038.09 Seconds
887.42 Seconds	1021.62 Seconds

Table B.11: Synchronisation time test result for zoom level 14 with attachments.

B.2 Complete result of synchronisation time test without attachments

Zoom level 4 (1 region)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
16.65 Seconds	16.56 Seconds
16.66 Seconds	19.11 Seconds
16.19 Seconds	17.77 Seconds
18.06 Seconds	16.05 Seconds
16.41 Seconds	17.80 Seconds

Table B.12: Synchronisation time test result for zoom level 4 without attachments.

Zoom level 5 (2 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
18.34 Seconds	18.90 Seconds
16.34 Seconds	16.25 Seconds
18.10 Seconds	17.72 Seconds
17.48 Seconds	16.38 Seconds
17.73 Seconds	17.88 Seconds

Table B.13: Synchronisation time test result for zoom level 5 without attachments.

Zoom level 6 (6 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
17.41 Seconds	16.62 Seconds
16.02 Seconds	17.83 Seconds
17.42 Seconds	17.77 Seconds
16.19 Seconds	16.26 Seconds
17.99 Seconds	16.39 Seconds

Table B.14: Synchronisation time test result for zoom level 6 without attachments.

Zoom level 7 (13 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
18.88 Seconds	16.11 Seconds
16.61 Seconds	16.29 Seconds
16.53 Seconds	21.62 Seconds
18.16 Seconds	15.96 Seconds
17.80 Seconds	16.15 Seconds

Table B.15: Synchronisation time test result for zoom level 7 without attachments.

Zoom level 8 (41 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
19.79 Seconds	19.76 Seconds
16.43 Seconds	16.07 Seconds
16.08 Seconds	16.25 Seconds
17.85 Seconds	18.69 Seconds
16.58 Seconds	16.44 Seconds

Table B.16: Synchronisation time test result for zoom level 8 without attachments.

Zoom level 9 (119 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
19.44 Seconds	16.36 Seconds
16.02 Seconds	18.12 Seconds
19.26 Seconds	16.22 Seconds
18.87 Seconds	16.21 Seconds
16.85 Seconds	16.41 Seconds

Table B.17: Synchronisation time test result for zoom level 9 without attachments.

Zoom level 10 (362 regions)	
Forenoon - 25/3/2019	Afternoon - 25/3/2019
17.32 Seconds	19.61 Seconds
18.94 Seconds	16.76 Seconds
16.99 Seconds	17.33 Seconds
16.88 Seconds	16.83 Seconds
20.31 Seconds	19.54 Seconds

Table B.18: Synchronisation time test result for zoom level 10 without attachments.

Zoom level 11 (1124 regions)	
Forenoon - 27/3/2019	Afternoon - 25/3/2019
15.36 Seconds	20.29 Seconds
20.53 Seconds	16.83 Seconds
17.14 Seconds	16.83 Seconds
17.87 Seconds	19.27 Seconds
18.21 Seconds	17.44 Seconds

Table B.19: Synchronisation time test result for zoom level 11 without attachments.

Zoom level 12 (3134 regions)	
Forenoon - 27/3/2019	Afternoon - 25/3/2019
18.59 Seconds	16.44 Seconds
16.05 Seconds	18.13 Seconds
16.19 Seconds	18.71 Seconds
19.21 Seconds	17.44 Seconds
16.88 Seconds	18.50 Seconds

Table B.20: Synchronisation time test result for zoom level 12 without attachments.

Zoom level 13 (7220 regions)	
Forenoon - 27/3/2019	Afternoon - 25/3/2019
28.08 Seconds	19.64 Seconds
19.50 Seconds	18.55 Seconds
20.01 Seconds	17.18 Seconds
22.55 Seconds	23.58 Seconds
21.81 Seconds	19.89 Seconds

Table B.21: Synchronisation time test result for zoom level 13 without attachments.

Zoom level 14 (14623 regions)	
Forenoon - 27/3/2019	Afternoon - 25/3/2019
29.00 Seconds	35.15 Seconds
26.85 Seconds	36.90 Seconds
28.36 Seconds	32.62 Seconds
28.03 Seconds	35.86 Seconds
26.94 Seconds	33.48 Seconds

Table B.22: Synchronisation time test result for zoom level 14 without attachments.