

Scalable Database Architecture for Human Indoor Mobility Systems

Magnus Alderslyst
Kongshem

Master i informatikk

Innlevert: juni 2016

Hovedveileder: John Krogstie, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Abstract

Throughout recent years, Indoor Positioning Systems (IPSs) have been developed and introduced as commercial systems for a variety of stakeholders. IPSs have many areas of utilization such as monitoring utilization of a building, tracking humans and equipment, wayfinding purposes and estimating the wear of buildings. Cisco MSE is an IPS that collects anonymous indoor positioning data to provide indoor location-based services. This system is currently utilized at Norwegian University of Science and Technology(NTNU).

Previous work has researched how the position data from IPSs can be used to aid facility managers and research optimal visualization methods to interpret, analyze and visualize the data. The research has reached a point where it needs to test these systems on a larger scale with greater amounts of data. This thesis' goal was to create a system to store and provide fast and easy access to historical indoor positioning data for these systems. The thesis also evaluates the position data and transforms it into a more convenient schema.

The project followed the Design Science Research method where an iterative process of several suggestions lead to the creation of three possible artifacts. The evaluation was based on the artifacts read performance, their ability to scale when the data basis increases and if they are fault tolerant. The results of these evaluations lead to selecting AsterixDB which was evaluated to be useful for applications visualizing indoor positioning data. This project encourages to further evaluate other big data management systems for this data set and compare the results with this thesis findings. A proper data cleaning program should also be implemented.

Sammen drag

Gjennom de seneste årene har innendørs posisjoneringssystemer (IPSs) blitt utviklet og introdusert som kommersielle systemer for et bredt spekter av interessenter. Innendørs posisjoneringssystemer har mange potensielle utnyttelsesområder som å overvåke bruken av spesifikke bygninger, sporing av mennesker og utstyr, wayfinding-tjenester og estimering av slitasje på bygninger. Cisco MSE er et innendørs posisjonssystem som kan samle anonyme innendørs posisjonsdata som kan benyttes i innendørs lokasjonsbaserte tjenester. Dette systemet brukes på NTNU i dag.

Tidligere forskning har forsket på hvordan posisjonsdata fra innendørs posisjonssystemer kan brukes til å hjelpe eiendomsforvaltere. I tillegg har det blitt forsket på optimale visualiseringsmetoder for å tolke, analysere og visualisere innendørs posisjonsdata. Forskningen har nå nådd et punkt hvor det er behov for å teste disse systemene i en større skala med større datamengder. Målet med denne forskningen var å lage et system for å lagre, og gi rask og enkel tilgang til historiske innendørs posisjonsdata for disse systemene. Dette prosjektet vurderte også posisjonsdataene og endret de slik at de fikk en mer praktisk struktur.

Denne forskningen ble gjennomført i henhold til *design science research methodology* hvor en iterativ prosess med flere forslag førte til etablering av tre mulige artefakter. Disse artefaktene ble evaluert basert på leseytelsen, hvor bra systemet skalerer når datagrunnlaget øker og deres evne til å tolerere feil. Resultatet av disse evalueringene førte til at AsterixDB ble valgt som det endelige artefaktet fordi det ble evaluert til å være nyttig for applikasjoner som visualiserer innendørs posisjonsdata. Dette prosjektet oppfordrer fremtidig forskning til å evaluere flere *big data management systems* for denne typen data set ved å sammenligne resultatene med dette prosjektets funn. Et program som fjerner støy i dataene bør også implementeres.

Problem definition

MazeMap has provided an indoor and outdoor way finding application for students and staff at NTNU, Gløshaugen. By using the WiFi network at Gløshaugen, MazeMap has been able to collect a large amount of position data. Previous work(Aulie, 2015; Eriksen, 2015) has been conducted to utilize the data and present the data in a web application using different visualization techniques. The data set contains two months of data, but previous work has only utilized a small portion of this data. Therefore the applications created has been limited to a small time period of one day.

This thesis will build upon the previous work and find a way to structure the large data set in a database, so that the applications can utilize the entire data set. This will complete the previously created applications to provide a better solution for representing human indoor mobility patterns. In addition to creating a database, the project has to address the problem of sending large amounts of results to a web application.

Preface

This thesis is the result of my research in Informatics at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) from the fall of 2015 to the spring of 2016. I would like to thank my supervisor John Krogstie for his support throughout this project. I would also like to thank my sub-supervisor Dirk Ahlers for all the technical discussions and assistance during the development. Jan Grønsberg and Arne Dag Fidjestøl also deserves some recognition for their help with the project server, Linux expertise and miscellaneous discussions.

Contents

Abstract	i
Sammendrag	iii
Problem definition	v
Preface	vii
List of Tables	xii
List of Figures	xiii
Acronyms	xv
1 Introduction	1
1.1 Project background and Motivation	1
1.2 Project Description	2
1.3 Research questions and goals	3
1.4 System Requirements	4
1.5 Report outline	5
2 Research design and methodology	7
2.1 Design Science Research	7
2.2 Implementation of design science research	9
3 Theoretical Background	11
3.1 Big Data	11
3.2 Positioning Systems	14
3.2.1 Localization Techniques	14
3.2.1.1 Triangulation	14
3.2.1.2 Trilateration	15

3.2.2	Indoor positioning systems	16
3.2.2.1	WiFi	16
3.2.3	Outdoor positioning systems	17
3.3	MazeMap	17
3.3.1	MazeMap application	17
3.3.2	Cisco Mobility Services Engine	18
3.4	Previous work	19
3.5	How the data can be used	19
3.5.1	Visualization of the data	21
3.6	Use cases	22
4	Data Set	25
4.1	Data types	25
4.1.1	Geographic Latitude and Longitude	25
4.1.2	Unix timestamp	27
4.1.3	JSON	27
4.2	PosData server	28
4.2.1	Noise in data set	29
4.2.1.1	Accuracy	29
4.3	Project server	31
4.4	Data set considered as Big Data	31
5	Technical Background	33
5.1	Possible queries on the data set	33
5.2	Execution time	35
5.2.1	Matrix generation	35
5.3	Architecture	36
5.3.1	NoSQL	36
5.3.1.1	MongoDB	37
5.3.2	SQL	39
5.3.2.1	MySQL	39
5.3.3	NoSql vs SQL	41
5.3.4	AsterixDB	43
5.4	Other	46
5.4.1	Streams & garbage collection	46
5.4.2	B-trees	48

5.4.3	Joda Time	48
6	Technical Evaluation	49
6.1	The data set	49
6.2	Project server	50
6.3	Matrix	50
6.3.1	Possible issues	53
6.4	Performance testing	54
6.5	MongoDB	58
6.5.1	Implementation	58
6.5.2	Testing	58
6.5.3	Other	61
6.6	MySQL	62
6.6.1	Implementation	62
6.6.2	Testing	62
6.6.3	Other	64
6.7	AsterixDB	64
6.7.1	Implementation	64
6.7.2	Testing	64
6.7.3	Other	67
6.8	Comparison	67
6.9	Choosing the final artifact	71
7	Result	73
7.1	Data set	73
7.2	Matrix	74
7.3	AsterixDB	75
8	Discussion and future work	77
8.1	Discussion	77
8.2	Future work	78
8.2.1	Integrating the artifact with the previous created prototypes	78
8.2.2	Data cleaning	78
8.2.3	Investigate other database architectures	78
8.2.4	Workload	79
8.2.5	Look at other data compression solutions	79

CONTENTS xi

Appendices **81**

A AsterixDB web interface **83**

B Data alteration **85**

C Email correspondence with AsterixDB developers **89**

D AsterixDB implementation **93**

 D.1 Test iteration 1 93

 D.2 Test iteration 2 94

E Import more data into the database. **95**

Bibliography **97**

List of Tables

4.1	Decimal precision for latitude/longitude calculated with the earths mean radius of 6371km	26
4.2	Decimal precision for latitude/longitude calculated with a latitude radius of 2852km . . .	26
4.3	Latitude and longitude max/min noise	30
5.1	SQL vs NoSQL	42
6.1	Test results from compressing the result set to a matrix using the proposed matrix solutions Matrix v1 and Matrix v2 using 1 day of data.	52
6.2	Test results from compressing the result set to a matrix using the proposed matrix solutions Matrix v1 and Matrix v2 using 30 days of data.	52
6.3	Batch size values in MongoDB querying one day of data.	60
6.4	Batch size values in MongoDB querying seven days of data.	60
6.5	Batch size values in MongoDB querying thirty days of data.	61

List of Figures

2.1	Design Science Research Process Model (Vaishnavi and Kuechler, 2004)	8
3.1	The IBM characterizes Big Data by volume, velocity and variety. (Zikopoulos et al., 2011)	12
3.2	Triangulation (Hightower and Borriello, 2001)	15
3.3	Trilateration	15
3.4	How Cisco MSE gathers position data (Little and O’Brien, 2013)	18
3.5	Visualization with a heat map (Aulie, 2015)	21
3.6	Tracking movement of devices using the Leaflet PolylineDecorator (Aulie, 2015)	22
4.1	Getpos JSON result	28
4.2	The area where positions are registered in the data set.	30
4.3	The acceptable area of positions at Gløshaugen.	30
5.1	Example query in SQL syntax	34
5.2	Sharding in MongoDB	38
5.3	AsterixDB system overview (Alsubaiee et al., 2014a)	45
5.4	Asterix software stack (Alsubaiee et al., 2014a)	45
5.5	Garbage collection - Marking step	47
5.6	Garbage collection - Normal deletion step	47
5.7	Garbage collection - Deletion with compacting	48
6.1	A summary of the results of test queries in MongoDB.	59
6.2	Import JSON in mongoDB	61
6.3	A summary of the results from test queries in MySQL.	63
6.4	A summary of the results from test iteration 1 in AsterixDB.	65
6.5	A summary of the results from test iteration 2 in AsterixDB.	66
6.6	Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to one day.	68

6.7	Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to seven days.	69
6.8	Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to thirty days.	69
7.1	The structure of the data set after the modifications performed by this project.	74
A.1	Screen shot of the web interface of AsterixDB.	83

Acronyms

CPU	Central Processing Unit
GPS	Global Positioning System
OPS	Outdoor Positioning System
IPS	Indoor Positioning System
RFID	Radio-frequency identification
WPS	Wifi-based positioning system
JSON	Javascript Object Node
AP	Access Point
API	Application programming interface
NTNU	Norwegian University of Science and Technology
Cisco MSE	Cisco Mobility Service Engine
XML	Extensible Markup Language
WLAN	Wireless Local Area Network
DSR	Design Science Research
nm	Nanometers
SAN	Storage area network
RDBMS	Relational Database Management System
ACID	Atomicity, Consistency, Isolation, Durability

Chapter 1

Introduction

This chapter contains an introduction of the thesis. It explains why and how the project was conducted. The project background will be presented and the motivation behind the thesis will be clearly stated. Next, the goal of this thesis will be described. Finally, the outline of the report is presented.

1.1 Project background and Motivation

This project is connected to indoor human mobility pattern, which is a joint research project between the Norwegian University of Science and Technology and Wireless Trondheim/Mazemap in the context of Wireless Trondheim Living Lab. (Andresen et al., 2007) The project seeks to test new products and services using indoor positioning systems, which is useful for several stakeholders to manage and monitor buildings such as universities, shopping malls, airports and hospitals.

The research had arrived at a stage where the large amounts of positioning data needed to be managed and stored so that applications can utilize the entire data set. Ahlers et al. (2015) made a study on different visualization techniques based on the same data set this project is using, except they only used one day of the data set. This is the motivation behind this research project. Since the amount of data available was larger than the data utilized in the original research project, a proper database was needed. Additionally, the server where the data set was located provided a poor method of extracting the data. Since the data set contained noise, a better solution for handling the data set was therefore needed and is what this project seeks to do.

1.2 Project Description

This project was conducted over the course of nine months in the fall of 2015 and spring 2016. The goal for this thesis is to evaluate different database structures and utilize one of them to create a database structure for a large data set of position data. The position data is provided by MazeMap¹. Previous related work and the original research projects had conducted an extensive background study to identify usage areas for human indoor positioning data and found several possibilities for different stakeholders. This project built upon these findings and focused on how the entire available data set could be used in the applications previously created.

Based on the goals and motivation for this project a research process began with investigating and analyzing the data set, and identifying potential use cases for the data. This investigation process led to the development and testing of different database systems with different properties and abilities. With each iteration of development and testing, the results and evaluation was documented. The evaluation was based on the project goals, research questions (see section 1.3) and system requirements (see section 1.4). Previous work also identified issues with the data set and issues related to the large amount of data. (Aulie, 2015; Eriksen, 2015) These issues were also addressed by this project, see chapter 5.

This project was conducted using the design science research methodology.(Vaishnavi and Kuechler, 2004) This research methodology focuses on creating new IT products here also called Artifacts. These artifacts are developed through an iterative process from the beginning of a research project, through research problem and motivation to development and evaluation of the artifact. The artifact represents the end of the project and is used to solve the identified problem.

The result of this project was a database architecture that satisfied the previously described evaluation criteria. The database contains the entire data set and can be interacted with to retrieve the desired location data. The data set was altered to better represent the information for each data entry.

A guide with instructions on how to connect, use and interact with the artifact is attached to this thesis. A detailed description of the artifact is given in chapter 7.

¹<http://mazemap.com>

1.3 Research questions and goals

This section will explain the research questions and goals for this project, and how this project seeks to answer these questions.

Goal nr 1: Find and implement a scalable database architecture that is able to handle millions of rows of location data.

Goal nr 2: Create a solution that solves the problem of sending large amounts of data to the browser.

Based on these goals, three research questions guided the research of this thesis. The first and second research question concerns the first goal of the project by evaluating different database systems. The second research question concerns the second goal by implementing a solution that handles the data size limit in browsers.

RQ1 Does traditional relational database management systems or NoSQL databases provide a fast and scalable architecture for the data set?

RQ2 Does other database systems outperform RDBMS or NoSQL databases based on this projects data set?

RQ3 How can the data set be transformed to a manageable size for browsers?

To be able to answer research question 1, a RDBMS and a NoSql system has to be implemented. The project then has to evaluate the performance and the abilities of the system to come to a conclusion. The reason both RDBMS and NoSql should be evaluated is because of the structure of the data set, see section 4.2. The project should also seek information in other related articles to compare the findings.

To answer research question 2, the project has to investigate other database systems that exist. The project has to evaluate the properties of these databases and compare them to the requirements. The project will test the systems if necessary to be able to conclude on the suitability of the system.

Research question 3 is an open ended question that allows this project to explore different solutions. The question was formulated as an open ended question because the question should not restrict what type of solution the project comes up with. The question was meant to encourage the project to seek non-traditional methods and investigate several options.

In this project, the goal was to investigate different database systems and architectures that can be used to store the data set. This required acquiring knowledge of many different database technologies to understand their properties and what type of problems they are designed to solve. This projects

goal is not to implement a full stack application with a suitable database architecture and a front-end visualization of the data, but to investigate and implement a database containing the data set.

1.4 System Requirements

The requirements for the system are split into four parts, read performance, scalability, availability, and data set requirements. The database systems tested in this project will be evaluated after these requirements.

Read performance

In any database the performance is important. Generally one would measure both write, delete and read performance, but in this project the read performance is more essential because the data set is historical data and the goal of the project mainly concerns querying the data. The requirements for the system does not involve altering or manipulating the data set, hence write, update or delete operations are not essential.

Scalability

The current data set has a fixed size collected in a specific time period. Collecting more data and inserting it into the system is likely to happen in the future. therefore the system should be able to scale with an increase in the data set size. This means that the database system should be able to handle larger amounts of data without major impact on the read performance. Scalability includes more than just speed. It should be possible to distribute the database across a number of nodes or possibly multiple data center, and it should be able to store hundreds of millions of records in the database.

Storage and availability

The resulting artifact should also provide a reliable storage system with high fault tolerance. If an error occurs and data is lost the system data would be incomplete and contain errors. Potential errors can be disk crashes on the server or a system service error. To avoid these potential problems the system should provide a backup function or data replication to provide redundancy, high availability and the ability to restore lost data.

Data set requirements

The data set used in this project needs to be altered based on the intended usage discovered by previous work. The data set should reflect more detailed information about the time the position was registered and provide a more transparant representation of what building the position is registered in.

1.5 Report outline

The outline of the rest of this research report is as follows:

Chapter 2 - Research design and methodology describes the research design and methodology utilized in this research.

Chapter 3 - Theoretical Background describes the Big Data concept and provides an overview of the localization techniques used to collect the data set for this project. Previous work and use cases will also be described.

Chapter 4 - Data set gives a detailed description of the data set, what information it contains and issues related to the data set. It will also describe the server used in this project.

Chapter 5 - Technical Background describes the possible tools and databases that is relevant for this project. It will also describe how the challenges related to the data set can be handled.

Chapter 6 - Technical Evaluation will give a detailed evaluation of the solution used to handle the challenges related to the data set. It will also give a detailed evaluation of the databases this project implemented and tested.

Chapter 7 - Result presents the solution this project chose as its artifact. It will also describe the solution this project chose to compress the query results.

Chapter 8 - Conclusion and future work concludes the thesis, discusses the challenges related to this project and potential future work.

Chapter 2

Research design and methodology

This section will describe the research design and methodology used in this thesis. This includes a description of the methodology and how it was performed during the project.

2.1 Design Science Research

This project selected a Design Science Research(DSR) approach. The reason for this selection is that the project aims to create a solution based on the identified requirements. This section will present an overview of the design science research methodology and discuss why this was selected as the research methodology.

A project performing a design science research approach is mainly a problem-solving paradigm (Hevner and Chatterjee, 2010). The project focuses on developing new information technology products that can be used to solve an identified problem. The Design Science Research approach consists of five main steps; Awareness of Problem, Suggestion, Development, Evaluation and Conclusion.(See figure 2.1) (Vaishnavi and Kuechler, 2004)

Awareness of Problem

This step describes the beginning of the process where the problem a project is to solve has to be identified. This step may come from several sources such as new developments in industry or in reference to a discipline. The output of this phase is a formal or informal proposal for a new research effort.

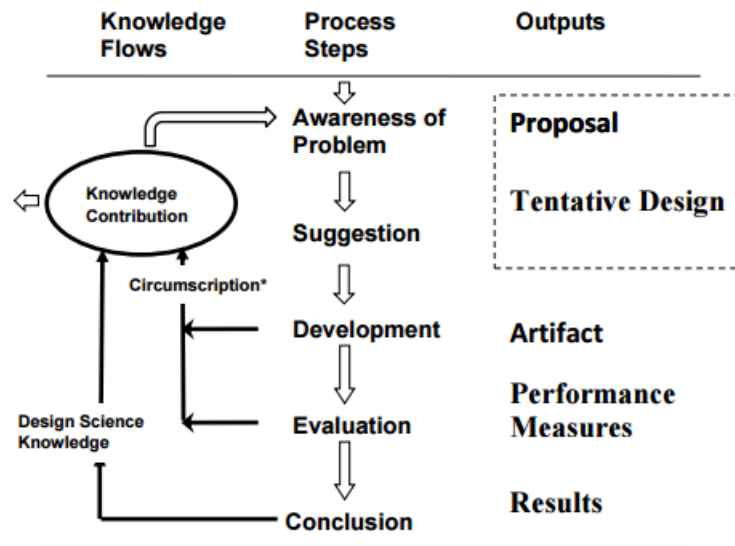


Figure 2.1: Design Science Research Process Model (Vaishnavi and Kuechler, 2004)

Suggestion

This step follows the proposal created in the previous step. Suggestions are a creative step where new functions or designs are suggested based on the existing elements. This step is often criticized because it prohibits repeatability which is often necessary in a creative phase where humans are involved. The output of this phase is a tentative design or prototype to support the proposal.

Development

In this step, the tentative design is further developed and implemented. The technique used for the implementation will differ depending on the artifact that is supposed to be created. The output of this step is an artifact.

Evaluation

Once the artifact is created, an evaluation is necessary to perform in order to evaluate if the artifact satisfies the criteria that were made in the first step. Any deviations from the expectations must be noted and explained. These deviations contribute to the knowledge of the issue at hand and might lead to a new round of suggestions. This results in a research cycle where the hypotheses are modified again to fit the new knowledge and a new development phase may be initiated. The output of this step are performance measures.

Conclusion

This step ends the research cycle when the results of the artifact are acceptable in terms of solving the problem.

2.2 Implementation of design science research

For each step in the design science research process described in section 2.1, the processes related for this project are explained below in order to provide an outline of the execution of the project.

Awareness of Problem

In this project, the problem was to identify potential suitable database systems that solved the requirements for the system. The awareness of the problem originated when the original research project described in section 1.1 had reached a point where the entire data set needed to be utilized in their applications for visualization of indoor human mobility patterns. The Proposal was therefore to create a database architecture that stores that data in an efficient and fault tolerant manner and allowing applications to retrieve the data based on the required parameters.

In order to perform this step, it was important for the researcher of this project to get an overview of the existing system and of the data set to identify the relevant research questions and to make sure the work this project was to perform would be of significance to the original research project. In order to understand the data set and their properties, this project had to read and understand previous work (see section 3.4) to acquire knowledge about localization techniques and the technologies used to collect the data. Additionally, this knowledge was obtained by reading relevant literature, talk with the stakeholders such as the original researchers and guidance from the supervisor. These activities was performed interchangeably because one piece of information lead to another question which needed to be answered. When a satisfiable understanding and overview was obtained, the information was documented in chapter 3.

When the background for the project was documented and understood by the researcher, a formal proposal for this research effort was defined by the research questions and research goals. The research questions and goals are further described in section 1.3. When these were defined, proper suggestions to a solution was proposed.

Suggestion

The functionality described in the previous step was the foundation of what this project had to implement. Based on the research questions and research goals, this project decided to test and implement different database architectures and compare them to each other. This means that for each suggestion, a development and evaluation phase had to be performed.

The first suggestion was to implement and test an SQL database. This was to enable the researcher to evaluate it the project could benefit from the properties that a traditional RDBMS possess. The properties of SQL databases are elaborated in section 5.3.2.

The second suggestion was to implement a NoSQL database. Since NoSQL provides other features than traditional RDBMSs, the project decided that it would be useful to compare their performances on this projects data set. The properties of NoSQL databases are elaborated in section 5.3.1.

The last suggestion was to implement AsterixDB. AsterixDB is an open source Big Data Management System(BDMS). Testing a specific big data system was useful for this project to evaluate if it performs better than the traditional database systems described in section 5.3.2 and section 5.3.1.

Development

The suggested database system, which potentially is the Artifact, was developed and implemented.

Evaluation

Qualitatively focused evaluations were performed on the Artifact in order to determine if the resulting system fulfilled the requirements (see section 1.4). The evaluation produced performance measure which lead to new rounds of suggestions in order to have other potential Artifacts to compare the results to.

Conclusion

In the end, the desired Artifact was selected and the research cycle was ended. The result was a database system which fulfilled the requirements of the system and was evaluated to perform better than the other suggestions in the research cycle. The results were documented in this thesis and communicates the results of the research.

Chapter 3

Theoretical Background

This chapter will explain the theoretical background for this project. First an overview of the term Big Data is explained. Then an overview of the localization techniques used to collect the data set used in this project is given. Previous work will be presented followed by how the data set for this project can be visualized for the end user. Finally the use cases this project had to handle will be presented.

The goal for this project is to identify and create a fast and scalable database architecture (1.3). To be able to do this, one has to identify how the data are supposed to be used. Since the data set consists of position data from the past, the data will only be written once to the architecture. When users want to view the data from different time periods, it will require a lot of read operations.

3.1 Big Data

In the previous decades the amount of data stored around the globe has rapidly increased, which creates a demand for an increase in data power. The development of processors has moved slower relatively to the development of storage capacity. This trend has influenced the research of data storage, data processing and analytics of big data. (Simon, 2013)

The term "Big Data" can be interpreted in many ways, such as large quantities of data or data of significant size. No matter how one defines the term, it is important to acknowledge key Big Data principles. (Zikopoulos et al., 2011)

- Big Data solutions are ideal for analyzing not only raw structured data, but semi-structured and unstructured data from a wide variety of sources

- Big data solutions are ideal when all or most of the data needs to be analyzed versus a sample of the data, or a sampling of data is not nearly as effective as a larger set of data from which to derive analysis.
- Big Data solutions are ideal for iterative and exploratory analysis when business measures on data are not predetermined.
- Big Data is well suited for solving information challenges that does not natively fit within a traditional relational database approach for handling the problem at hand.

As stated in Zikopoulos et al. (2011), recognizing the final point in the list above does not eliminate conventional database technologies such as RDMS. They are equally important and a relevant part of an overall solution.

Big data is often conceptualized by "V's". IBM conceptualizes Big Data with Volume, Variety and Velocity (see figure 3.1).

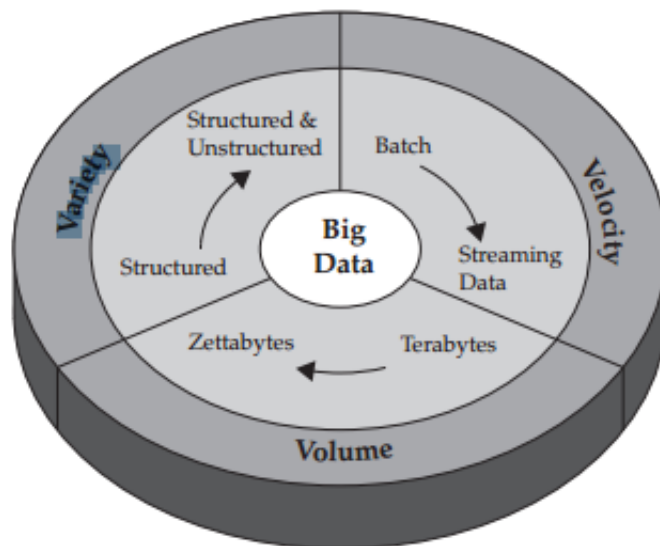


Figure 3.1: The IBM characterizes Big Data by volume, velocity and variety. (Zikopoulos et al., 2011)

Volume

The volume of data stored in the world in the year 2000 was 800.000 petabytes. Zikopoulos et al. (2011) estimates that by 2020 the total volume stored in the world will be 35 zettabytes. This shows that the amount of data in the world is increasing. Another issue Zikopoulos et al. (2011) presents is that while the total volume of data is increasing, the percentage of data an organization is able to

process is decreasing. This gap is referred to as "the blind spot" and represents the data organizations are unable to see and therefore can not utilize when analysing their data. (Zikopoulos et al., 2011)

Variety

The variety term provided by IBM represents all types of data. Because of the vast volume of Big Data, new challenges for the data centers emerge. The different types of sensors and smart devices creates a pool of data with great variation. This is because these devices do not provide structured relational data, but raw, unstructured and semi-structured data from social media, e-mails, documents, sensor data, web pages and so on. This issue is a large task for traditional relation systems because of the various structure. This is the reason why big data with great variations does not fit into traditional database technologies.(Zikopoulos et al., 2011)

Velocity

Since the volume and variety of the data has changed, the velocity has also changed. To understand the term velocity one can consider how fast the data is collected and stored. One can imagine that the volume of the data one is collecting is a consequence of how quickly the data arrives. therefore velocity can be defined as the speed at which the data is flowing. (Zikopoulos et al., 2011)

Additionally to IBM's conceptualization of big data, other sources (Chen et al., 2014; Ware, 2012) justifies adding additional concepts to big data. These are Veracity, Value and Visualization.

Veracity

Veracity refers to the trustworthiness or consistency of the data. Collecting large amounts of data may result in content with false information. An example here are data from Facebook messages or Twitter posts. The risk of collecting data with typos from these sources are present which means the data will contain faults. With the growing rate of social media, the veracity has become increasingly acute and has to be taken into account when using the data.

Value

Possessing big data is useless unless one can make use of the data and turn it into something with value. The idea is to be able to make use of the data without ignoring or missing out of the value of the data. An example is a news station listening to statements coming from a press conference. If a system is unable to interpret that the statement "the bank interest will increase by fifty percent" is significant and important information, the system has failed. (Chen et al., 2014)

Visualization

In order to retrieve the value from the data, it must be visualized in a suitable manner to make the data useful to the user. Using different techniques to transform the data or combining them with other

data points may increase the value of the data.(Ware, 2012) An example is heat maps. Combining the data and transforming it to draw a heat map may give the user more value of the data than reading all the latitude and longitude values from a list.

A comparison and evaluation of this projects data set related to big data is located in section 4.4.

3.2 Positioning Systems

A positioning system is a system that uses different techniques to determine the position of an object or a person. The different techniques that exist will not be elaborated on in this project because it does not affect this project, but the techniques used to collect the data utilized in this project will be explained and discussed.

There are two main categories of positioning systems that exist: indoor positioning system (IPS) and outdoor positioning system (OPS). This project is based on data collected by MazeMap from an indoor positioning system. This section will present both IPS and OPS to explain the difference in area of use and a discussion of why they both exist to complement each other.

3.2.1 Localization Techniques

Every positioning system uses either one or more localization techniques. Fallah et al. (2013) describes four different techniques: Dead reckoning, direct sensing, triangulation and pattern recognition. The following sections will describe triangulation and trilateration since the data collection process for this project utilized these methods and the other techniques are not relevant for this project.

3.2.1.1 Triangulation

Triangulation is a technique used to determine a position based on an angle or bearing measurement. To locate a device in a two dimensional space, two angle measurements and one length measurement is sufficient. In order to locate a device in a three dimensional space, e.g. what floor the device is located on, two angle measurements, one length measurement and one azimuth measurement is required.(Hightower and Borriello, 2001) See figure 3.2.

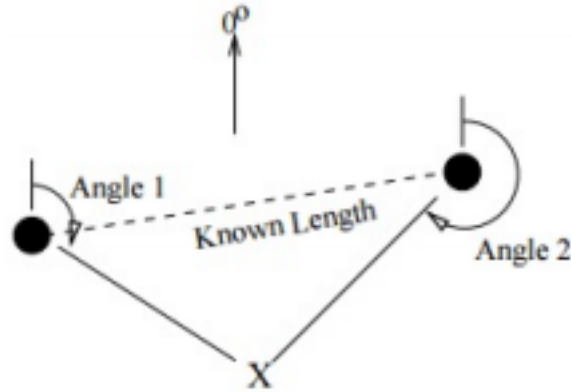
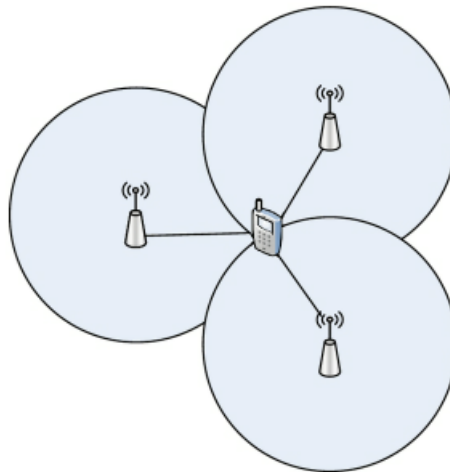


Figure 3.2: Triangulation (Hightower and Borriello, 2001)

3.2.1.2 Trilateration

Trilateration is a technique similar to triangulation. They both produce the same result, but trilateration uses distance to determine a position rather than angles. In order to determine a position in two dimensional space, only three non-collinear points are needed. In order to determine a position in three dimensional space, four non-collinear points are required. There are in general three techniques used to measure the distance to an object, Direct, Time of Flight(ToF) and Attenuation. (Hightower and Borriello, 2001) See figure 3.3.

Figure 3.3: Trilateration¹

¹www.bluefletch.com/blog/the-guide-to-indoor-location-services/, last visited 06.06.16

3.2.2 Indoor positioning systems

An indoor positioning system (IPS) can potentially have many purposes, but is best known to locate objects inside structures such as buildings, warehouses, hospitals, universities and factories. IPS's are used where traditional Global Positioning Systems (GPS) are unable to receive a signal. There are several technologies that can be used as an IPS, such as RFID, bluetooth, infrared and WiFi, but there is no standard for IPS. Scenarios where IPS's are used can be to locate a package inside a warehouse, locating a patient or a specific transportable equipment inside a hospital, locating a fireman inside a burning building, or locating and tracking a dog sweeping a building for explosives or personnel. (Liu et al., 2007)

The data set for this project has used WiFi technology to collect the data. The reason behind this is because it did not require any additional hardware to be installed in order to collect the data. Using bluetooth, infrared or RFID technology would require installation of hardware where the data was intended to be collected. In addition, these technologies has a lower range than WiFi, which would result in more hardware needed to be installed than the hardware needed for WiFi. Since this project uses data collected from WiFi, no further explanation of the alternative technologies will be made since it is not a part of this project.

3.2.2.1 WiFi

WiFi is a local area wireless computer networking technology which provides devices the ability to connect to a network by operating on frequencies of 2.4 and 5 gigahertz using the IEEE 801.11 standard. WiFi is used in many electronic devices such as smartphones, computers, digital cameras, video-game consoles and televisions. The devices connect to a network resource via a wireless network access point. These access points usually have an effective range up to 20 meters, depending on the thickness of walls or other obstacles.

WiFi-based positioning systems (WPS) is used when traditional GPS are inadequate. The principal behind a WPS is to measure the intensity of the received signal and the method of fingerprinting. (Chen and Kobayashi, 2002) IPS's using WiFi can be found at large facilities such as hospitals, office buildings and university campuses. In order for a IPS to successfully utilize WiFi, there has to be installed a large number of access points, making it possible to triangulate (see section 3.2.1.1) or trilaterate (see section 3.2.1.2) the position of a device.

3.2.3 Outdoor positioning systems

Global Positioning System (GPS) is the most used technology in outdoor positioning systems (OPS). GPS's uses trilateration based on satellite positions to calculate the location. It receives periodic signals sent by each satellite to compute latitude, longitude and altitude. Since GPS is free, available anywhere on earth and reliable, it has become the standard for OPS. Using the most advanced technologies in the GPS world like the High Accuracy Nationwide Differential GPS System, also known as HA-NDGPS, an accuracy of 10 centimeters can be achieved. Privacy is also withheld when using a GPS because the signals travel one-way. The main disadvantage of a GPS is the connection limitation inside buildings, since the signal needs a near clear path between the GPS receiver and the satellites.

Other techniques used for OPS are cell-tower positioning and WLAN, but both techniques has a lower precision than GPS because of multi-path reflection problems. Multi-path reflection occurs when reflections from e.g. buildings results in a different travel distance or time. The traveled path may be longer than the actual path in a straight line from the cell-tower to the device. (Fallah et al., 2013)

3.3 MazeMap

3.3.1 MazeMap application

MazeMap is a hybrid indoor and outdoor way finding application that can locate a user's position in a given area with an accuracy of up to 5-10 meters (Biczok et al., 2014). The solution is offered by Wireless Trondheim and was started as an R&D project between Wireless Trondheim and NTNU, and launched for the first time in the fall of 2011, under the name CampusGuide. The application uses a combination of trilateration (see section 3.2.1.2) and GPS to locate the user at selected locations. The service is offered at twenty-seven locations ², including NTNU Gløshaugen in Trondheim, Oslo airport Gardermoen, the company Evry in Stavanger and Trondheim, and St. Olavs Hospital (Trondheim University hospital). In addition, Billund airport in Denmark and a Uppsala Bio-Medical centre in Sweden uses the service.(Biczok et al., 2014)

On Gløshaugen, MazeMap enables its users to find their own location both indoor and outdoor. Users can search for any location on campus, such as lecture halls, reading rooms, toilets, and cafes, and use the application to find the shortest path from their own location to any given location on campus. MazeMap is also able to detect which floor the user is located on, enabling easy navigation inside any building.

²<http://use.mazemap.com/?v=1&campusid=1&left=10.2571&right=11.0852&top=63.7455&bottom=62.9427>, last visited 01.06.16

3.3.2 Cisco Mobility Services Engine

The Cisco Mobility Service Engine (MSE) is the wireless infrastructure created by Cisco to calculate time, position and Media Access Control(MAC) address of devices that has enabled WiFi and are in range of the area covered by their network. (Little and O'Brien, 2013) The system is able to analyze the flow of devices that are located inside buildings that are equipped with WiFi such as universities, hotels and hospitals. A WiFi device is continuously transmitting probe requests to identify the best Access Point (AP) for the client. The requests are used to calculate the Received Signal Strength Indicator(RSSI) of the AP's to ensure that the client is always connected to an AP and connected to the AP with the best range. These requests are gathered by the AP's that are connected to MSE and forwards the MAC address to the MSE. The MSE then uses the RSSI to triangulate the position of that client. See figure 3.4.

MSE also has an API which makes it possible to access the movements of any device if the MAC address is known. This means that any person can use this API to e.g. locate themselves by using their own MAC address on their device.

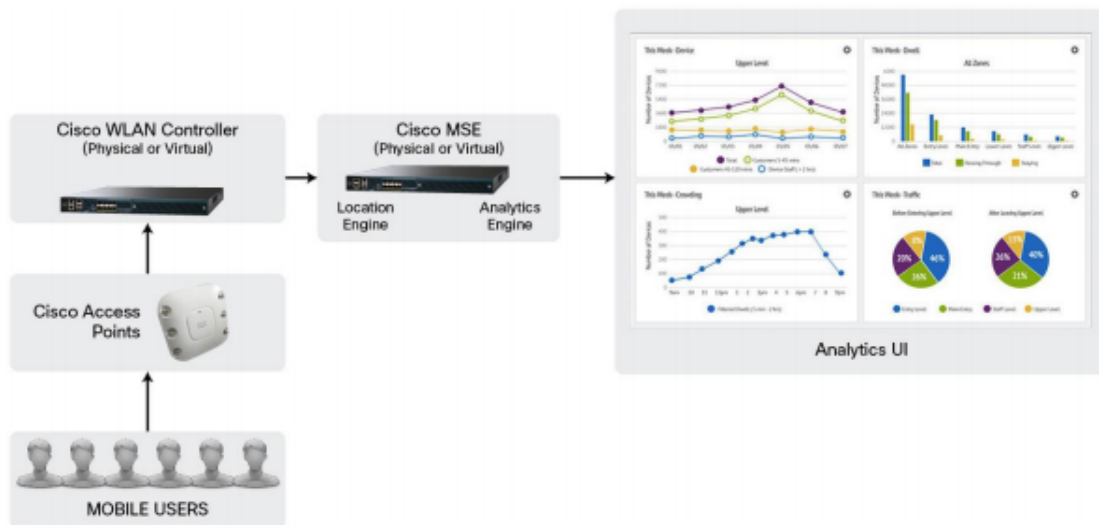


Figure 3.4: How Cisco MSE gathers position data (Little and O'Brien, 2013)

3.4 Previous work

This thesis is built upon previous work related to the same data set as this thesis is based on. Kristoffer Aulie wrote a thesis in 2015 on "Human Mobility Patterns from Indoor Positioning Systems" (Aulie, 2015). The aim of his work was to interpret, analyze and visualize positioning data. The goal was to identify optimal methods to visualize indoor human mobility patterns in order to present the data in a useful format for potential stakeholders. The thesis concluded that the prototype was considered useful for potential stakeholders and that a more advanced tailored tool will assist their work flow.

Other projects related to this thesis is "Visualization of Crowds from Indoor Positioning Data" (Eriksen, 2015), also written in 2015. This thesis goal was to process position data to give an overview of crowds inside buildings, and investigate how this type of data will assist facility managers. A prototype was developed which utilized indoor positioning data.

The common feature for both thesis's is that they concluded that a system for visualizing position data is useful for business stakeholders and facility managers. Both thesis' also stated that they only utilized a small portion of the available data set and that the server they used is not scalable for larger data sets. In order to use their prototypes to their full extent they recommended creating a big data database structure containing data over a larger time period.

Following the thesis of Aulie (2015) and Eriksen (2015), Dirk Ahlers and John Krogstie published a paper for an international conference on big data and analytics for smart cities. The paper presented a system for visual analytics of people's movements on a large university campus. The system served as a proof-of-concept for building-related analysis inside a campus, and their results was a proof-of-concept application visualizing position data with heat maps and building traffic depicted by drawing movement patterns. They conclude that this type of indoor visualization provide valuable information to stakeholders and that patterns of the devices movement can be identified. The thesis was also limited by using one day of the data set.

3.5 How the data can be used

Previous work (Aulie, 2015)(Eriksen, 2015) has determined that companies will benefit from a solution showing position data inside buildings. Below are some of their findings related to how the data set can be used.

Area Utilization and Usage

The data can be used to show the utilization of certain areas or buildings, which will give an unique opportunity to identify how an area is used in practice. Identifying areas that are used are

equally important to identifying areas that are not used. Identifying how an area is not used gives contractors and facility managers knowledge to determine excess areas that can be used for other purposes, thus optimizing the entire building. Knowing how spatial resources are used are considered valuable information for facility managers. Mapping popular and non popular areas can also be used to record the wear of buildings, thus providing the ability to estimate when a building or area needs renovation. Another usage is to help estimate how many people needs to be relocated if an area or building has to be closed for reconstruction or renovation. Similarly the data can be used to estimate if a building can handle the extra personnel if they are relocated there.

These types of information can also be used for other purposes, such as monitoring the changes in a building over time. By storing information of the utilization of building over time, an analysis will aid the facility managers in making decisions. This sort of usage of the data would replace any manual counting used in the past and for the future.

Industry sectors that are known to use position data inside buildings are the healthcare sector. Yao et al. (2012) presented a list of hospitals that utilized RFID to track personnel and equipment. They argued that the advantages of tracking persons and equipment gives major advantages such as patient safety, cost reduction and efficiency improvements. They argued that using IPS in hospitals and healthcare has great potential and that the development in technology with RFID will only contribute to further improve the advantages of such a system.

Another interesting usage area is to evaluate energy consumption. Research shows that in the U.S, 40% of the energy consumption origins from buildings, and 48% of this is from ventilation, heating and air conditioning. (Li et al., 2012) By monitoring the utilization of areas, the temperature can be regulated based on how many people are located in an area. Research shows that energy savings up to 10% to 15% is possible to achieve by regulating HVAC-systems based on how many people are located in an area. (Sohn, 2010) These research results shows that this is a potential usage area for IPS. By combining the data from an IPS and feeding it to an HVAC-system, energy consumption can be reduced. Instead of an HVAC-system assuming an area is filled to its capacity thus heating the room accordingly, the system can heat a room to a temperature suited for the amount of people located in a given area.

Movement

The data can also be used to give an overview of the movement of an individual. By viewing how people move inside buildings one is able to acquire knowledge about the people utilizing the buildings and how the current building or buildings in the future should be designed. This knowledge has many purposes, such as identifying popular entrances, rooms where people go to meet, and exits. A

movement analysis would primarily aid scenarios involving safety such as fire and evacuation plans. Creating evacuation plans for large buildings or at a large campus is difficult to perform, but important. Performing a fire drill while collecting indoor position data would aid the fire department and facility managers to create an evacuation plan where they are able to evacuate as fast as possible. The data can also be used to identify the number of individuals that are located inside the building in order to determine if a building is completely evacuated or if there are people still left inside.

3.5.1 Visualization of the data

Aulie (2015) identified possible ways of displaying the data for the user:

Heatmap

A heatmap is way of visualizing different positions on a map. The heatmap accepts positions as input and then uses different colors to paint the map based on the density of the positions. The result is a map showing where the density of positions are at their highest and lowest. See figure 3.5.

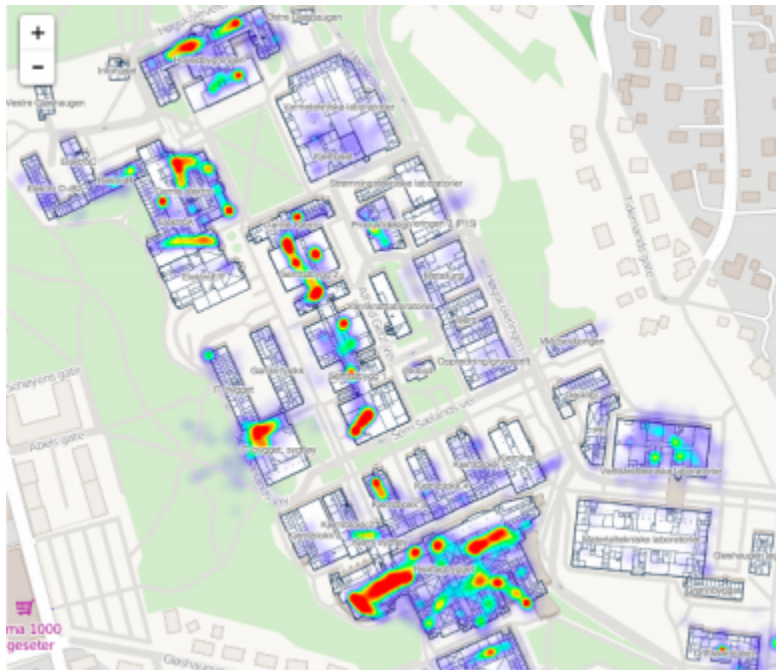


Figure 3.5: Visualization with a heat map (Aulie, 2015)

Leaflet PolylineDecorator

PolylineDecorator is a tool used by previous work (Aulie, 2015) which allows one to draw patterns along paths based on positions. This tool gives the user a useful view to track the movement of individuals between buildings or inside a building. See figure 3.6.

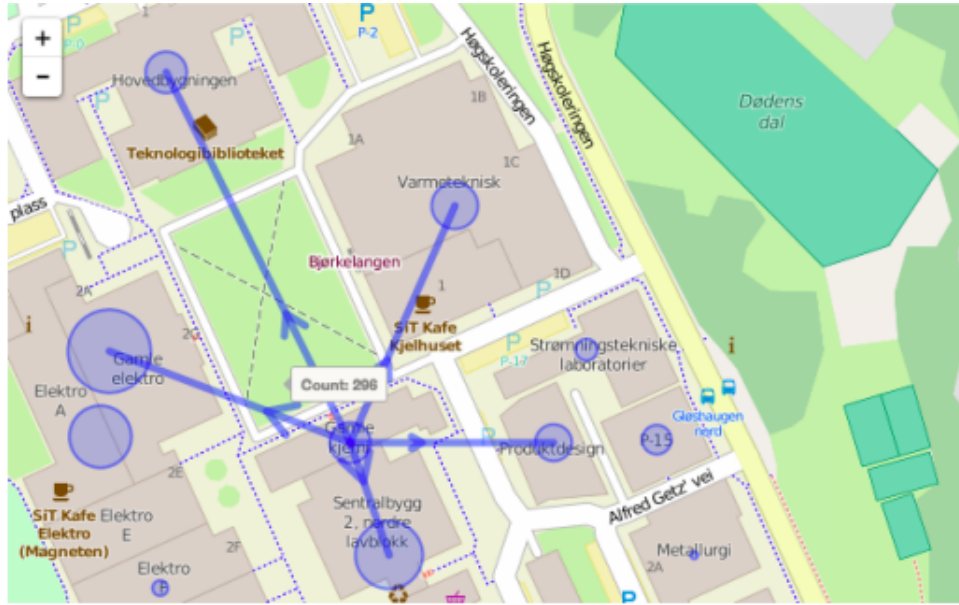


Figure 3.6: Tracking movement of devices using the Leaflet PolylineDecorator (Aulie, 2015)

3.6 Use cases

This section will describe possible use cases related to the goals for this project. This thesis uses the term *location data* frequently. Location data is a latitude and longitude position registered by a device such as a computer or smart phone. Previous work has identified usage areas for this type of data. This section aims to provide some example use cases in an operational environment. This section will describe information about the data set which is described in detail in chapter 4. The use cases are based on the possible queries described in section 5.1.

Use case 1 - The user wants to view location data in a specific time period.

This use case requires the system to provide the possibility to ask for location data between two dates. Each registered position in the data set has a unix timestamp field which provides this information.

Use case 2 - The user wants to view location data based on specific hours of the day in a specific time period.

This use case requires the system to provide information about what hour of the day the location data was registered. This information can be extracted from the timestamp field, but it requires the system to perform calculations to extract the information.

Use case 3 - The user wants to view location data based on specific hours of the day in a specific time period and/or a specific day of the week.

This use case requires the system to be able to determine what day of the week the location data was registered. This information can be extracted from the timestamp field, but it requires the system to perform calculations based on the timestamp.

Use case 4 - The user wants to view location data based on a specific building and/or floor to gain knowledge about the abrasion.

This use case requires the system to be able to analyse what building or floor the location data was registered. The data set has a hierarchy field, which provides information about what campus, building and floor of which the registered position belongs to. In this use case the system has to analyse the hierarchy field and extract the relevant information.

Use case 5 - The user wants to observe how the usage of buildings change over time.

This use case is similar to use case one and four because it requires the same information from the data set, i.e time period and building information. Additionally, the user wants to see changes based on a daily, weekly or monthly summary.

Use case 6 - The user wants to track the movement of a specific device

This use case requires that each position data has a unique ID for each device it has registered. The data set provides this information with the id field, which is unique combined with the date the position was registered; that is the timestamp field.

Use case 7 - The user wants to view location data with a specific accuracy

This use case is useful if the user wants to exclude location data that was registered with a low accuracy, which in some cases may be inaccurate information.

Use case 8 - The user wants to retrieve large results from the database and visualize the results in a web application

This use case involves research question three and is related to the problem of compressing the result to an acceptable size for web applications.

Chapter 4

Data Set

This chapter will describe the data set, its properties, pros and cons of the data set, the quality of the data set and the server this project will use in this project.

The data set used in this project consist of location data gathered by the Cisco MSE, described in section 3.3.2. The data set is stored at a server located at NTNU. The server continuously gathered location data at Gløshaugen from 2. September 2014 to 2. November 2014. The position data consists of 154.378.597 million individual location updates from WiFi devices located within the range of the WiFi at Gløshaugen. The size of the original JSON file is 34,2 GB. Each time the devices searched for available Access Points by transmitting probe requests, their location was stored.

4.1 Data types

4.1.1 Geographic Latitude and Longitude

To be able to understand how positioning data works, there are a some terms one needs to understand.

Coordinates is the combined values of latitude and longitude. The coordinates is therefore a given position on earth. Latitude specifies the north to south position of a point on the surface of the Earth. Its an angle with the range between 0 degrees and 90 degrees. Longitude specifies the east to west position of a point on the surface of the Earth.

Table 4.1 gives an overview of how precise the latitude and longitude values are based on the number of decimals used. The values are calculated using the *Haversine* formula depicted below.

Haversine formula:

$$a = \sin^2\left(\frac{\Delta\Phi}{2}\right) + \cos\Phi_1 * \cos\Phi_2 * \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 * \operatorname{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = R * c$$

where Φ is latitude, Δ is longitude and R is the earth's radius(mean radius = 6.371km).

Decimal	1	2	3	4	5	6	7	8	9	10	11
Precision	11.1km	1.1km	111m	11.1m	1.1m	0.11m	11mm	1.1mm	111microns	11119nm	1112nm

Table 4.1: Decimal precision for latitude/longitude calculated with the earths mean radius of 6371km

An important notice is that these values are not correct everywhere on earth. The length of a degree of longitude is not the same everywhere on earth. It depends on the radius of a circle of latitude. Using the earth's mean radius results in distances only accurate at the equator. To calculate the distance between two points at Gløshaugen at NTNU, the radius of the correct latitude value. The average latitude value at Gløshaugen is approximately 63.415. The radius of that latitude value is 2852km. The radius is calculated with this formula:

Formula used to calculate the radius of a given circle of latitude

$$6372 * \cos(\text{latitude}/180 * \pi)$$

where latitude was set to 63.415. Given the latitude radius, table 4.1 can be recalculated to present more accurate values. Table 4.2 shows the result using the Haversine formula and the correct latitude radius at Gløshaugen NTNU.

Decimal	1	2	3	4	5	6	7	8	9	10	11
Precision	4.97km	497m	49.7m	4.9m	49.7cm	4.9cm	4.9mm	0.497mm	0.049mm	4977nm	497nm

Table 4.2: Decimal precision for latitude/longitude calculated with a latitude radius of 2852km

Table 4.1 and 4.2 shows the difference results calculating the distance between two coordinates on the equator and at Gløshaugen at NTNU. There is a significant difference as for each decimal value, the distance is about twice as long at the equator than at Gløshaugen. Since the earth is not shaped like a perfect sphere but more like a ellipsoidal, the values in table 4.2 has an error of up to 0.3%. Since this possible error margin is so low, this thesis considers it to be insignificant and not to be of an important factor in further calculations.

4.1.2 Unix timestamp

Unix time, also known as Epoch time, is a 32-bit integer describing the number of seconds elapsed since Thursday 1. January 1970 at 00:00:00 (UTC). Unix time is the standard time measurement used in applications world wide.

4.1.3 JSON

JavaScript Object Notation, abbreviated JSON, is a an open standard data format used for transmissions between a server and a client. JSON is supported in the Cisco MSE API(3.3.2) with parameters and a return statement. Even though Extensible Markup Language, abbreviated XML is also supported, only JSON will be utilized in this project.

```

1  [
2    {
3      "hierarchy": "Gløshaugen>Varmeteknisk og Kjelhuset>3. etasje",
4      "timestamp": 1412121600,
5      "longitude": 63.41904008880636,
6      "salt_timestamp": 1412121602,
7      "latitude": 10.405133392801218,
8      "id": "4",
9      "accuracy": 221.8943996627205
10   },
11   {
12     "hierarchy": "Gløshaugen>Gamle kjemi>Kjeller",
13     "timestamp": 1412121600,
14     "longitude": 63.41808624621221,
15     "salt_timestamp": 1412121602,
16     "latitude": 10.403139597256665,
17     "id": "110",
18     "accuracy": 56.083199914753536
19   },

```

Figure 4.1: Getpos JSON result

4.2 PosData server

To access the data set, a python script developed by Aulie (2015) must be used. Executing a command will generate a JSON file containing the data. Figure 4.1 shows a dump of the data set.

Explanation of the fields:(Aulie, 2015)

- **Hierarchy:** Explains the location of the position. The hierarchy is showed by using the ">" operator. The hierarchy is divided into the campus name, building name and then the floor of the position.
- **Timestamp:** The UNIX timestamp for the position data.¹
- **Longitude:** The latitude of the position data(See section 4.1.1). This is an error caused by the PosData server(4.2) where the latitude and longitude coordinates are switched.
- **salt_timestamp:** This is the time when the last generation of the salt value was performed. This value is processed through a salt, e.g hashed with a another value. In this case the value is the MAC addresses of the devices in order to generate anonymous ID's for the devices.

¹www.unixtimestamp.com/

- **Latitude:** The longitude of the position data(see section 4.1.1).This is an error caused by the PosData server(4.2) where the latitude and longitude coordinates are switched.
- **Id:** The ID of the device the position data was recorded from. By viewing all the positions from the same ID within one day in chronological order will result in a movement trace of that specific device. This id is not unique and can not be used as a primary key in a database.
- **Accuracy:** Explains the accuracy of the position data. Higher value indicates a lower accuracy, and lower values indicates a higher accuracy.

This structure allows access to every necessary attribute for location data. The data set does not contain a unique field that can be used as a primary key in a database. This issue must be handled when implementing a database either by generating a unique key for each row of data or use the auto-generated unique ID provided by the databases.

4.2.1 Noise in data set

The data set contains a lot of noise. Some of the positions are hovering outside a building or buried under ground outside a building. A proper data cleaning program should be written to handle these positions.

As described in section 4.2 the latitude and longitude values in the data set is switched. This is also considered as noise in the data set and should be handled by this project.

Other position noise are latitude and longitude positions registered far away from campus. Table 4.3 shows the highest and lowest registered latitude and longitude values in the data set, along with the *acceptable* latitude and longitude coordinates. From table 4.3, we can plot the positions into a map and see how far off the worst positions are. Figure 4.2 shows the corresponding map.

4.2.1.1 Accuracy

Each position from the PosData server contains a latitude and longitude value. These values has between thirteen and fourteen decimal points. That means each point is given a theoretical accuracy of about one angstrom. That is half the thickness of a small atom. Table 4.2 describes how precise each decimal point is. Representing a person's location does not require the precision in nanometers or millimeters. That excludes more than five decimals. Five decimals gives a precision of 0,49 meters, which arguably gives a representative precision. Four decimals gives a precision of 4,9 meters. One can fit a lot of people within a 4,9 x 4,9 meter area. Most dorm rooms are smaller than 4,9 x 4,9 meters, and considering the purpose of the position data, the precision should be more precise than four decimals.

Table 4.3: Latitude and longitude max/min noise

Type	Value
Min lat	61.20482
Max lat	71.68732
Min long	1.85396
Max long	10.41160
Min acceptable lat	63.41372
Max acceptable lat	63.41992
Min acceptable long	10.39873
Max acceptable long	10.41134



Figure 4.2: The area where positions are registered in the data set.

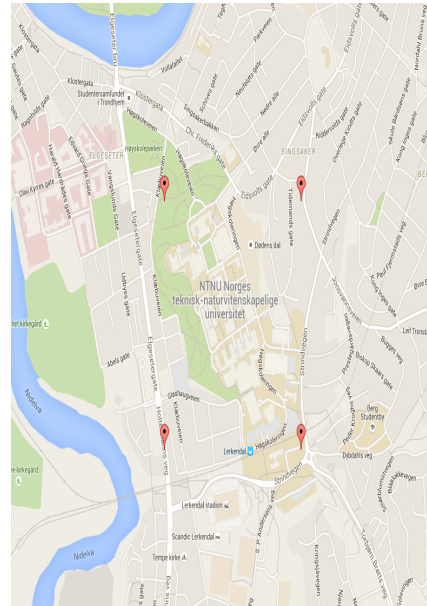


Figure 4.3: The acceptable area of positions at Gløshaugen.

4.3 Project server

The project was not in position to replace the hardware or software the server was set up with other than requesting a reasonable RAM size. This section is meant to provide a brief overview of the hardware and software the server uses.

This project used a hyperv VM server running on a DELL PowerEdge R720 HyperV server with 192GB RAM and 32 cores. The project was assigned a partition of this server for the project which was initialized with 8GB RAM and a dual core 2.00 GHz Intel Xeon processor. The name of the partition is **HV-6023**. The server disposes these two cores without any interference from other partitions, and the server is not used by any other projects or persons.

The server disk is a 2TB virtual drive using iSCSI SAN. SAN is a network that gives access to consolidated block level data storage. SAN usually has its own network of storage devices, usually not accessible though LAN by other devices. SAN only allows block-level operations. However, file systems built on top of SAN can provide file-level access. These are known as shared-disk file systems. The SAN has a 1Gbit/s connection which is shared between many other virtual machines. This gives an expected read performance of 106 MB/s. This may limit the performance compared to a local disk when it comes to transfer rate in MB/s, but the SAN is considered fast for internal operations and has a large amount of read/write cache which is more important for this project than the transfer rate.

NTNU is planning to upgrade the SAN on the server to HyperV during the fall of 2016. If the faculty finds it useful, the server will be upgraded to a virtual disk. The new SAN supports 10Gbit/s connections which will remove bottlenecks in the transfer rate.

4.4 Data set considered as Big Data

As explained in the introduction to this chapter, the position data consists of 154.378.597 million individual data rows. The size of the original JSON file is 34,2 GB. Recall the definition of big data in section 3.1. Since the data set contains large amounts of data and a large amount of individual data points, the data set fulfills the Volume property from the big data definition. Additionally, the data set has no Velocity rate since the data set is historic data collected from 2. September to 2. November 2014. If the data set had been real time data, the data set would have a Velocity property. When it comes to the Variety property, the data set has a fixed structure. The different positions has exactly the same properties and one can therefore state that the data set does not have a Variety property.

As described in section 4.2.1 the latitude and longitude values are switched which indicates that the data set contains errors and is not to be trusted. Table 4.3 and figure 4.2 shows how the data set

contains location points far away from the original location where the data was collected. These two flaws in the data set are considered to break the trustworthiness of the data set. Fixing the switched latitude and longitude values is an easy task, and if performed, will increase the Veracity of the data set.

If one disregard the fact that the latitude and longitude values are switched, this project argues that the data set provides the necessary information for location data, since latitude, longitude, hierarchy and timestamp information is provided. This means that the data set possesses the Value property.

Previous work and the original research project has already researched different ways of visualizing the data set and concluded that there are multiple ways of visualizing the data for users to aid the user to understand the data. Depicting the data in heat maps and using arrows to display movement are examples of visualization that can be used for the data set (see section 3.5.1). This means the data set fulfills the visualization aspect of big data.

Chapter 5

Technical Background

This chapter will present the technical aspects of this project. First possible queries on the data set will be presented. Then, the problems related to the requirements is presented. Lastly potential tools and technologies that can be used in this project is elaborated.

Based on the requirements for this project, the system this project will create needs to handle large amounts of data. This means that the system needs to be able to store hundreds of millions of rows of data and provide an easy, scalable and dynamic way of querying the data. The system should mainly focus on fast read operations rather than write operations because there will be performed more read operations compared to updating or inserting new data.

5.1 Possible queries on the data set

Based on the data set, there are many possible queries that can be performed on the data set. The simplest queries can be between date A and date B, but to acquire data from a specific date with specific attributes, the other attributes must be used. Other possible queries can be:

1. Querying on a specific building

Querying on a specific building was deemed necessary by previous work (Aulie, 2015; Eriksen, 2015) to provide more detailed information to facility managers.

2. Querying a specific floor or several floors.

The data set offers the possibility to view more detailed information than only their connected building. Querying on the floor provides more detailed information to the user.

3. Querying specific week days

Potential users of the system might want to view location data on specific weekdays such as Saturday and Sunday to observe how often the buildings are used in the weekends.

4. Querying specific hours on specific week days.

Querying on a specific hour of the day is useful for a user to e.g. observe where people are moving during lunch hours.

5. Querying on the accuracy.

Since the positions is registered with an accuracy attribute, the user might want to view position data with a specific maximum accuracy value.

6. Querying on a specific id

To be able to track a device, the system has to be able to query on the ID field.

7. A combination of the above

As each of these possible queries are relevant, a user might want to combine them into a more complex query such as querying a specific ID during lunch hours on a friday.

Performing a query on a specific floor requires an analysis of the hierarchy field in the data set. Since the information about the floor is found in a sub set of the field, the field has to be split into different parts by splitting on the ">" character. The same analysis is required when querying on a specific building or campus. Performing this type of analysis for each position for each query is time consuming and can be avoided by having the database performing the operation in advance. Creating additional fields in the data set named *campus*, *building* and *floor* will aid the database to gain faster access to the desired field. The hierarchy field can be removed if this operation is conducted.

Figure 5.1 shows an example query in SQL syntax how the data set might be queried by the user. SQL syntax is used to provide this example query because it is easily readable for both developers and non-programmers. The relevant columns in the data set are the latitude and longitude values, hence the values after the SELECT statement. The parameters after the WHERE clause asks for position data where the timestamp is greater than or equal to X, and less than or equal to Y. Additionally, the query asks for location data where the day of week is four, i.e. Thursday, and the hour of day is between 10:00 and 14:59. The floor is not specified in this example query, which means the result will contain all floors.

```
3 SELECT latitude, longitude FROM table WHERE
4 timestamp>= X AND timestamp<= Y AND dayOfWeek = 4
5 AND hourOfDay>= 10 AND hourOfDay<= 14;
```

Figure 5.1: Example query in SQL syntax

5.2 Execution time

One of the expected issues this project will encounter is the execution time of the database. Previous work, see section 3.4, identified that the large amount of data being pushed to the browser caused issues. The browser will load very slow when the data contains a lot of megabytes. Previous work loaded almost 700 megabytes to the browser, which caused the browser to load slow and in some cases not respond. One of the main problems this project therefore has to handle is to compress the result data to be as small as possible.

When querying large amounts of data and performing calculations on the data, the computation time will be significant. The entire data set utilized for this project has a size of 34.2 gigabytes and contains over 154 million positions. Iterating through tens of millions of rows will result in a significant execution time.

The following sub sections describes possible solutions to decrease the size of results from the queries.

5.2.1 Matrix generation

One way of decreasing the amount of data passed to the browser is to create a matrix containing the position data on the server side. This entails to create a matrix over the location the data set is from and create a cell for each latitude and longitude position. A prerequisite for this solution is selecting the area and storing the maximum/minimum latitude and longitude values. This solution allows the database to iterate through the data set and count each registered position for each cell. The matrix will be a two-dimensional array where each cell represents a latitude and longitude position. Early tests with one day of position data showed that to cover the campus at Gløshaugen, the matrix requires a dimension of 1261 x 620 using five decimals for the latitude and longitude values (see table 4.2). The matrix could potentially be larger if this solution is to be used on different locations, e.g. on an airport. Since the matrix size is smaller than an entire JSON file, pushing the matrix to the browser avoids the probability of a crash. This solution is later referred to in this thesis with the name **Matrix v1**.

Another solution is to give the matrix a fixed dimension that is not based on the maximum and minimum latitude and longitude values. This means that the dimensions of the matrix can be set to a size based on how accurate the user wants to view a heat map. Instead of each cell containing a position with one value of latitude and longitude values, each cell will contain the amount of positions based on number of different lat/long values divided by the width/height of the matrix. An example is with a position area with 400 different longitude values and 600 latitude values, and a fixed matrix

dimension of 200 x 200. This means that each cell will contain positions with two different longitude values, and three different latitude values. This solution is later referred to in this thesis with the name **Matrix v2**.

A third solution is to generate a matrix where the dimension is set dynamically based on the users screen resolution. An example screen resolution is 2560 x 1600. Compared to **Matrix v1**, the matrix dimension is about twice the size. To be able to set the dimension based on the users screen resolution, the web application or backend system has to read the resolution and calculate the coordinate value of the screens edges. Theoretically, generating a matrix with a dimension based on the screen resolution makes sense because generating a matrix with a higher dimension than the screen resolution results in a higher resolution than the user is able to see. The downside is for instance not knowing the zoom level of the view, the matrix might cover large areas outside the desired area, thus creating a large number of unnecessary matrix cells. Zooming in on the matrix will reveal inaccurate position points. To fix this issue, a new query may be run when the user is zooming, but this is time consuming work and leave the user waiting for each query to finish. This solution guaranties a dynamic solution, but has disadvantages related to an unnecessary amount of cells.

An evaluation of these solutions is described in section 6.3.

5.3 Architecture

This section will explain different tools and architectures relevant for this project anchored in the requirements (see section 1.4).

5.3.1 NoSQL

NoSQL is common term for databases with different properties than traditional relational database systems. NoSQL stands for *Not Only SQL* or *Not Relational SQL*. The key features are:

- Horizontal Scaling across many servers
- Replication and distribution of data across servers
- Has a call level interface or protocol
- Weaker concurrency model than SQL databases which has ACID transactions
- Efficiently distributes indexes and uses RAM for data storage
- Can dynamically add new attributes to data records.

The key features of NoSQL is horizontal scaling, partitioning and replication across many servers. These properties allows NoSQL databases to support a large number of read & write operations per second and is known as Online Transaction Processing (OLTP). This project seeks a system with high performance in read operations and does not consider write operations as important. Online Analytical Processing (OLAP) deals with historical data and is used in data warehouses to perform read operations with complex queries. This project therefore favours a typical OLAP system over OLTP.

NoSQL databases do not possess ACID properties as SQL databases does, but instead have what some authors call "BASE" properties: Basically Available, Soft state & Eventually consistent. Cattell (2011) argues that by giving up ACID constraints, the possibility of achieving better performance and scalability is much higher. NoSQL databases is relevant for this project because it supports horizontal scaling and heavy read/write operations. These properties covers most of the requirements for this project.

5.3.1.1 MongoDB

MongoDB is a popular open-source NoSQL database with every property described in section 5.3.1. MongoDB provides high performance and automatic scaling. The records are documents, which is a structure that consists of fields and value pairs. The values of the fields may include other documents, arrays or arrays of documents, which allows us to see similarities between MongoDB and JSON objects.¹ MongoDB stores the data in documents called BSON. This format is a JSON-like format that supports the main data types like integer, float, date, string and boolean. The client encodes document data structure into BSON and sends it over a socket to the MongoDB server. (Arora and Aggarwal, 2013)

Using MongoDB with their documents structure has great advantages because it supports native data types in several different programming languages, dynamic queries with indices like traditional RDBMS and MapReduce which allows complex aggregations across the entire document set.

MongoDB supports sharding, which is another word for horizontal scaling. The concept involves dividing and distributing the data over multiple servers, or *shards*. Each shard contains a piece of the entire data set and therefore reduces the total number of operations needed to access the data. Sharding is used by MongoDB to support large data sets and a high throughput of operations, which is highly relevant for this project. MongoDB uses either range based partitioning or hash based partitioning to shard the data into chunks.² Since sharding divides the database into several smaller sections, each shard processes fewer operations as the cluster grows. This results in a database structure which will

¹<http://docs.mongodb.org/manual/core/introduction/>, last visited 06.06.2016

²<https://docs.mongodb.com/v3.0/core/sharding-introduction/>, last visited 06.06.2016

increase capacity and throughput horizontally. Sharding also reduces the total amount of data that each server needs to store. With this ability, sharding will provide properties for a system which may be of significance for this project.

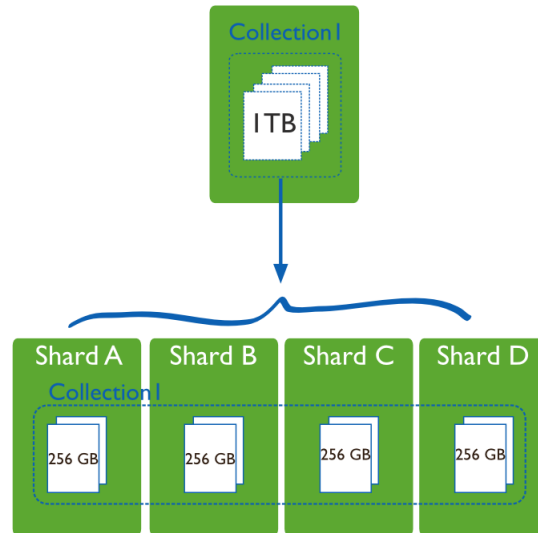


Figure 5.2: Sharding in MongoDB³

Sharding is a technique which is mainly used to improve the write processes of a database, but it is also recommended as optimal for heavy write operations. MongoDB states on their web site that:

“For write-heavy applications, deploy sharding and add one or more shards to a sharded cluster to distribute load among mongod instances.”⁴

MongoDB supports replication to provide redundancy and increase the availability. Storing multiple copies of data on different database servers provides fault tolerance against the loss of a server. A replica set is implemented by creating a group of *mongod* instances that maintain the same data set. One of these instances is the primary node, and the other instances are secondary nodes. The primary node handles all the read and write operations, but the primary instance may in some cases send read operations to the secondary nodes.⁵

There exists numbers of ways to optimizing MongoDB. One of them is to set the batch size. The batch size is a parameter one can set on the query in MongoDB which indicates how many documents will be processed at a time. If the data contains 1.000.000 documents and the batch size is set to 1,

³<https://docs.mongodb.com/v3.0/core/sharding-introduction/>, last visited 06.06.2016

⁴<https://docs.mongodb.org/manual/administration/analyzing-mongodb-performance/> last visited 01.06.16

⁵<https://docs.mongodb.com/manual/core/replication-introduction/> last visited 01.06.16

the number of round trips will be 1.000.000. If the batch size is set to 100, the number of round trips will be 10.000. E.g. with a batch size of 100 eliminate 99% of the network round trips.

Other optimization techniques is to make use of indexes and distribution among a cluster of machines. Creating indexes in MongoDB is similar to how it is used in traditional RDBMSs. If indexes is not utilized in MongoDB, a collection scan, i.e. scanning every document in a collection is done. By using an index, MongoDB can use this index to limit the number of documents needed to be inspected. MongoDB also supports performing *explain* on a query in order to evaluate the query performance. This is helpful to identify the indexes used in a query.

5.3.2 SQL

SQL stands for Structured Query Language and is a programming language designed for relational database management systems, also called RDBMS. SQL has been around for over 40 years and has been the main language for databases. SQL stores the data tables where the rows contain the information for each data entry and the column describes the properties for the rows. Each row has a fixed structure which can not be altered. There are some options to this issue such as making the database offline and altering the entire database. Other properties for SQL is their ACID properties and that it scales vertically. This means that expanding the database is expensive because the server has to expand. A summary can be found in table 5.1.

5.3.2.1 MySQL

The most commonly known SQL databases are MySQL and PostgreSQL. This project will elaborate on MySQL.

To improve the execution time for queries in MySQL, choosing and selecting an index is important. Indexes in MySQL are used to find rows with a specific column value quickly. If an index is not set, MySQL has to begin with the first row and read through the entire table to find relevant rows. If the table is large, the cost in time is large. Most of the indexes in MySQL are stored in B-trees, see section 5.4.2. In many cases, creating an index based on multiple columns is optimal because a query is rarely based on a single column. Based on the identified possible queries to be performed on the data set for this project (see section 5.1), indexes on multiple columns will be necessary. The following list describes the advantages of using indexes in MySQL: ⁶

- Finding rows that matches a WHERE clause quickly

⁶<http://dev.mysql.com/doc/refman/5.7/en/mysql-indexes.html>

- Elimination of rows quickly. If there are multiple indexes to choose from, MySQL uses the index that finds the smallest number of rows.
- If the table has a multiple-column index, any leftmost prefix of the index can be used to look up rows.
- Retrieving rows from other tables when join operations are performed. The query will perform more efficiently if the columns has the same type and size.
- Finding the `min()` and `max()` value for a specific indexed column.
- Sorting or grouping a table if the sorting or grouping is done on the leftmost prefix of a usable index.

In addition to creating indexes, setting the fetch size on a query statement in MySQL is necessary when working with large data sets because it prevents overload of the heap space in the JVM. Setting the fetch size to `Integer.MIN_VALUE` will signal the driver to stream the results row by row. Excluding this statement will result in the entire query to be fetched and will trigger an `OutOfMemory` exception if the data is larger than the JVM heap size.

As presented in section 4, the data set is in JSON format. Importing JSON data into MySQL is more complex than e.g. MongoDB. Out of the box, MySQL does not provide any easy solutions for import. To use the data set in a MySQL database, this project has to write code to read the JSON file and insert the rows manually. Reading large JSON files also requires the use of streams in order to avoid garbage collection, see section 5.4.1.

Related to this project MySQL is expected to have a low performance and not meet the requirements in section 1.4. It does not scale horizontally which is fundamentally restrictive when working with big data. SQL databases was created for relational databases and since this project uses one large table the relations between tables are mute.

MySQL supports replication to provide high availability. The advantages of replication in MySQL is the spreading of work load among multiple slaves to improve the performance. This model can improve the write performance, but with a trade off on read operations because of the increasing number of slaves to read from. ⁷

⁷<http://dev.mysql.com/doc/refman/5.7/en/replication.html> last visited 01.06.16

5.3.3 NoSql vs SQL

This section will provide an overview of the differences between SQL databases and NoSQL databases.

Looking at the technology used for traditional data stores, RDBMSs has been the preferred choice. In recent years, some of the properties RDBMSs has can be discarded such as transactions and durability, in favour of write performance and scalability. A traditional RDBMS is not cost efficient with scalability on write performance or distribution of data stores, because RDBMSs was not built for it. NoSQL replaces RDBMSs in these situations to achieve optimal performance.(Dyrkorn, 2016)

	SQL	NoSQL
Data storage	SQL databases uses a relational model, storing the data in rows and columns. The rows contains all the information about a specific entity while the columns represent all the separate data points.	NoSQL databases have different ways of storing the data. The most utilized are document, graph, columnar and key-value.
Schema's and flexibility	Each data record are fixed to a schema. This means that the columns must be decided before the data entry, and each row must contain data for each column. This rule can be amended, but it involves taking the database offline and altering the entire database.	The schemas are dynamic. Information can be added whenever needed and each data entry does not have to contain data for each property.
Scalability	In SQL databases, scalability is vertical. This means that if the database are to expand, the server needs to expand. This can be expensive to perform. It is possible to scale an RDBMS across several servers, but the process is expensive and time-consuming.	In NoSQL databases, scaling is horizontal. This means that it can be distributed across multiple servers. These servers can be both cloud instances and cheap hardware. This makes the scaling more effective and cheaper than in SQL databases. Modern NoSQL databases distribute data across servers automatically, which is time saving.
ACID	The majority of relational databases are ACID compatible	It varies between the technologies, but many NoSQL database systems sacrifice ACID compatibility for scalability and performance

Table 5.1: SQL vs NoSQL

Table 5.1 shows the major differences between SQL databases and NoSQL databases. Related to scalability, Rick Cattell (Cattell, 2011) argues for both SQL and NoSql. The advantages for SQL are that RDBMSs have had the majority of the market the last thirty year, which means that if new relational system are able to do everything a NoSQL database system can do, the obvious choice would be SQL. Other arguments are that successful relational RDBMSs are built to handle other specific application loads in the past. Examples are read-only or read-mostly data warehousing, OLTP on

multi-disk multi-core CPU's, distributed databases and now horizontally scalable databases. If the data in the database is well formed with a given structure, and the records is presumed not to require additional fields in the future, RDBMS is preferred.

Arguments in favour of NoSQL are that no SQL databases has yet shown the same dynamic scalability as NoSQL databases. If the queries only require a look up of objects based on a single-key, then a key-value store database is easier to understand and more adequate than a relational DBMS. Some applications require a dynamic structure of the data set, e.g adding/removing attributes for each record. Since RDBMSs in most cases require a given schema structure, NoSQL databases is preferred. NoSQL is relatively new to the market compared to RDBMSs, but NoSQL databases are well established in markets where the applications need to handle special cases, e.g. graph-following operations with object-oriented DBMSs.

5.3.4 AsterixDB

AsterixDB is a scalable data management system that can store, index and manage semi-structured data. It has its own query language (AQL) and data model (ADM). ADM is a key-value super-set of JSON because it has type constructors and additional primitive data types compared to JSON. The storage system in AsterixDB is a hash-partitioned LSM based storage layer.(Alsubaiee et al., 2014b) It also has support for external storage in HDFS. AsterixDB provides a rich set of data types such as spatial and temporal, and provides different types of indexing structures such as B+ Trees, spatial indices(R-Tree), and text indices. The execution engine in AsterixDB is Hyracks (Borkar et al., 2011), a data-parallel run time platform for shared-nothing clusters. Jobs submitted to Hyracks comes in the form of a Directed Acyclic Graph(DAG) made up by operators and connectors. The operators are responsible for consuming partitions of inputs and producing output partitions. The connectors redistribute data from the output partitions and provide input partitions for the next operator. Hyracks is used to accept and manage data-parallel computations requested by the other layers in the Asterix software stack(see figure 5.4), or directly by the end users of Hyracks.(Alsubaiee et al., 2014a; Borkar et al., 2011) Clients can submit queries in AsterixDB through the HTTP API. When AsterixDB processes a query it first compiles it into an algebraic form which is optimized by a rule-based optimizer and turned into a corresponding Hyracks job. AsterixDB can be classified as a Big Data Management System and aims to support a wider range of use cases than other Big Data technologies such as Hadoop-based query platforms. Their slogan is: *One size fits a bunch*

There are some similarities between AsterixDB and MongoDB, but the major difference is that AsterixDB also supports a full-power query language with SQL expressiveness. It uses the knowledge

of data partitioning and the availability of indexes to avoid scanning data sets to process queries, such as Hive and Spark does. The core functionality of AsterixDB are: (Alsubaiee et al., 2014a)

- A flexible, semi-structured data model for use cases ranging from “schema first” to “schema never”.
- It’s own expressive and declarative query language for querying semi-structured data, with at least the power of SQL.
- An efficient parallel query run time.
- Support for data management and automatic indexing;
- Support for a wide range of query sizes, with processing cost proportional to the task at hand.
- The run time query execution engine Hyracks, used for partitioned-parallel execution of query plans.
- Support for continuous data ingestion.
- the ability to scale gracefully in order to manage and query large volumes of data by using large clusters.
- Partitioned LSM-based data storage and indexing for efficient ingestion of new data.
- Basic transactional (concurrency and recovery) capabilities akin to those of a NoSQL store.
- Indexing options that include B+ trees, R trees, and inverted keyword index support.
- A rich set of primitive data types, including support for spatial, temporal, and textual data.

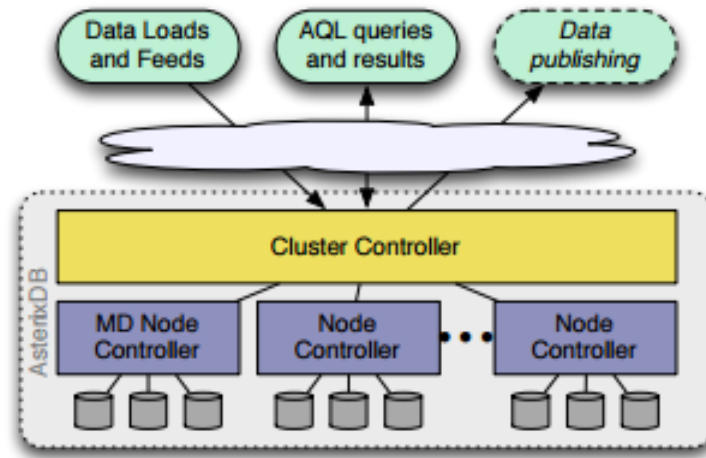


Figure 5.3: AsterixDB system overview (Alsubaiee et al., 2014a)

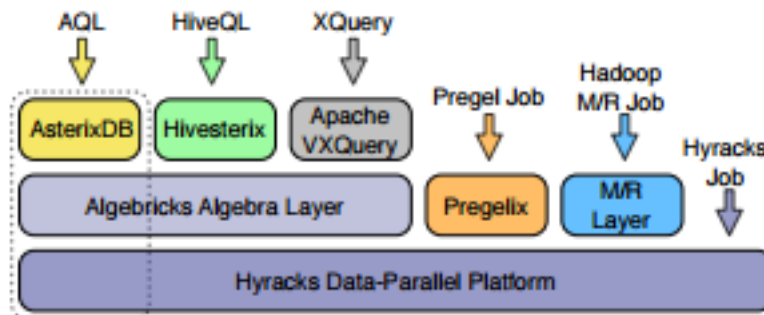


Figure 5.4: Asterix software stack (Alsubaiee et al., 2014a)

When a new disk component is created in AsterixDB, a validity bit is atomically set in a metadata page of the component to ensure that the operation has completed successfully. During a crash recovery a no steal policy, that is only committed operations from in-memory components, are replayed. This means that there is no need for an *undo* phase. Crash recovery in AsterixDB consists of two phases, an analysis phase and a redo phase. The analysis phase reconstructs the committed transactions by reading log records. The redo phase then replays the effects of the committed transactions. This ensures durability of committed transactions and that a redo operations is not performed if the effect of that redo operations is already made. (Alsubaiee et al., 2014b)

Related to the data set for this project, importing the json files as they are is not possible. Since AsterixDB uses ADM file extension, some modifications of the data set is necessary.

Traditional relational databases use conventional index structures such as B+ trees because of their

low read latency. These indexes use in-place writes to perform the updates which results in expensive random writes to disk. AsterixDB supports LSM-trees which avoids the cost of random writes by performing batch updates into a component of the index that resides in the main memory. When the main memory exceeds a specified threshold, the entries are flushed to disk forming a disk component. Since disk components accumulate on disk, they are merged together based on a merge policy that decides what to merge and when to merge. Since LSM-based indexes partitions data into multiple disk components, it is possible to use the partitioning to only access some of the components and successfully filter of the remaining components to reduce the query execution time. (Alsubaiee et al., 2015)

One of the developers of AsterixDB, Heri Ramampiaro, is a professor at Norwegian University of Science and Technology in Trondheim, working at IDI, the institute for data technology and informatics.

5.4 Other

This section will describe additional libraries and techniques relevant for this project.

5.4.1 Streams & garbage collection

When an application is reading or writing data, the data is temporary stored in the memory to handle the entire data set. When the data set is huge and exceeds the memory of the client, the application will fail due to garbage collection. Garbage collection is the process of looking at the heap memory and identifying which objects are used and which are not. If an object is in use, it means that the application holds a reference to that object. An object not in use means that the application no longer has a reference to that object. That means that the memory used to hold that reference is freed and can be reclaimed to reference another object.

In java, the process has two steps. The first step is called marking. That is when the garbage collector identifies which parts of the memory are used. See figure 5.5.

Marking

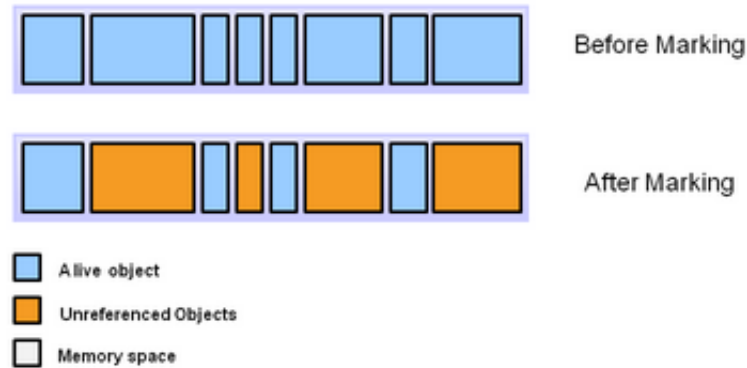


Figure 5.5: Garbage collection - Marking step ⁷

Step 2 removes unreferenced objects which frees space for new objects to be referenced. See figure 5.6.

Normal Deletion

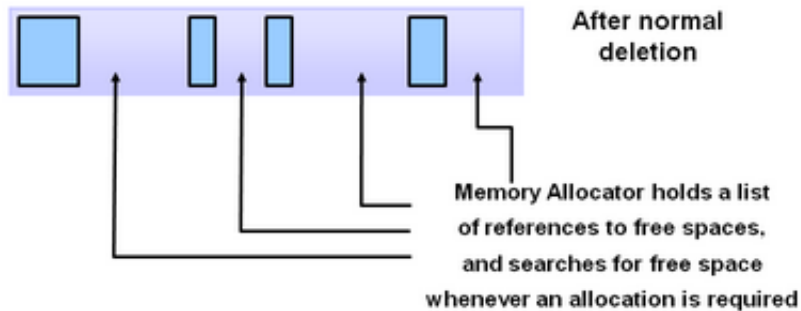


Figure 5.6: Garbage collection - Normal deletion step ⁷

Step 2 also has another operation to improve the performance. When deleting unreferenced objects, one can also compact the remaining referenced objects. By clustering these objects together, the memory allocation becomes easier and faster. See figure 5.7.

⁷<http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

Deletion with Compacting

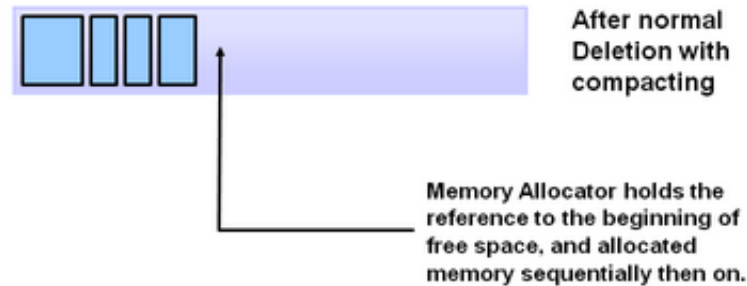


Figure 5.7: Garbage collection - Deletion with compacting ⁷

In order to avoid garbage collection and to read large amounts of data seamlessly, streams are used. This project utilized the Jackson⁸ library to read large amounts of data.

5.4.2 B-trees

B-trees are a data structure popular used for database indexes. The structure is always sorted which enables fast look-up for exact matches and ranges. The difference between a B-tree and a binary tree is that the b-trees can have n children while binary trees can maximum have two children. B-tree is available for most storage engines, including InnoDB, which this project used in MySQL. (Cormen, 2009) ⁹

5.4.3 Joda Time

The standard date and time classes prior to Java SE 8 are poor. Joda-Time has therefore become frequently used date and time library for Java. The design allows for multiple calendar systems, while still providing a simple API. The *default* calendar is the ISO8601 standard which is used by many other standards. Supporting classes include time zone, duration, format and parsing.¹⁰ Since the data in this project consists of timestamps, joda-time makes it easy and flawless to convert string dates, set date formats and getting the unix timestamp from a date object.

⁸<http://wiki.fasterxml.com/JacksonStreamingApi> last visited 03.05.16

⁹<http://dev.mysql.com/doc/refman/5.7/en/glossary.html> last visited 03.05.16

¹⁰<http://www.joda.org/joda-time/> visited on 01.06.16

Chapter 6

Technical Evaluation

This chapter will evaluate the data set and how it should be transformed to better represent the data. Then, a detailed evaluation of the solution to reduce the size of the data will be given. Third, test results of the selected database systems will be presented and how they were optimized to perform better related to this projects data set. In the end, a comparison of the database systems will be done.

6.1 The data set

Recall the possible queries presented in section 5.1. In some cases, companies would want to combine these queries and create a complex query to access the exact data they want. This project is focusing on create an architecture that enables users to perform these queries, and will not focus on enabling the users to perform the most complex combination of queries by splitting the data set into several tables and performing join operations. Query number three and four in the list is more complex than the other, because the query has to calculate the week day and hour based on the unix timestamp. These calculations can be avoided by adding additional fields to the data set.

The hierarchy field contains information about three different properties and should be split into three different attributes for each data entry. Dividing it into campus, building and floor will aid the database when performing the queries because it skips the process of evaluating a long string to extract the necessary information.

Adding the described fields will increase the number of attributes per location point. This means that the size of the data set will increase accordingly. This project added these fields to evaluate how much the data set would increase in size. Transforming the entire data set resulted in a data set size of 38.9 GB.

Previous work (Eriksen, 2015) decided to group the smaller buildings into larger groups because the smaller buildings located next to each other is practically the same building. The only arguments were that generating a heat map of these small buildings was less useful. This grouping format was not implemented in the data set this project was working with. This project argues that this type of grouping should be researched further and that it is a feature that may depend on the area where the location data was collected.

6.2 Project server

Recall section 4.3 describing the server used by this project. When working with large amounts of data the memory of the server represents an important part. To achieve good performance the memory can be upgraded to support results from larger queries, but memory expansion is expensive. To produce realistic test results, this project set the memory of the server to 8GB. This is large enough to support the smaller queries but not so large that the performance from the larger queries becomes unrealistic.

When performing performance tests on a server, background tasks or other tasks may steal resources from the tests one is measuring. To prevent these complications the project only performed one test at a time. This is time consuming working but necessary to produce liable results. Some databases also use caching to improve the execution time if the same query is executed multiple times in a row. To prevent this and achieve a realistic average execution time the project performed the queries multiple times and in order.

6.3 Matrix

This section will evaluate both matrix solutions described in section 5.2.1. A summary of the test results can be found in table 6.1 and 6.2.

Recall the proposed **Matrix v1** solution to reduce the data size explained in section 5.2.1. Using six decimals for each latitude and longitude value, transforming an entire day of location data (1. October 2014) with an original size of 920 megabytes to a two-dimensional array results in a size of 312,8 MB which is 34 % of the original size. Testing the same data set where the matrix only contains latitude and longitude values with five decimals gives a matrix size of 3,2 MB which is 0,34 % of the original size. This shows that reducing the decimals utilized for latitude and longitude to an acceptable value greatly reduces the data size. An area represented with five decimals in the latitude and longitude values represents an area of $0,24meters^2$. Based on this test result, the project decided to use five decimals to represent the latitude and longitude coordinates.

Further evaluating the solution showed that there are additional ways of further reducing the resulting matrix. Testing the proposed solution showed that out of the 781.200 thousand cells in the matrix, 624.425 thousand cells did not contain any position data. This equals 79.9 % of the matrix. By definition these matrices are sparse. Wilkinson (1971) defines a matrix as "sparse" if it has enough zeroes that it pays to take advantage of them. (Wilkinson, 1971) Sending a sparse matrix to the browser is a poor solution because it contains data that is not relevant for the user in this project. Since the user is querying location data, sending non-relevant location data is unnecessary. Decomposing the matrix by constructing Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) matrices represents a possible solution. (Buluç et al., 2009) These operations decomposes the two-dimensional array to three one-dimensional arrays. The CSR compresses the matrix by reading the rows first. The first of the three one-dimensional arrays contains the none-zero values, the second array contains the column indexes for each value and the third array contains the row pointer. CSC compresses the matrix by reading the columns first. The first of the three one-dimensional arrays contains the non-zero values of the matrix, the second array contains the row pointer and the third array contains the a list of indexes where each column starts. Since none of these arrays contain information about the zero values in the sparse matrix, the size of the matrix is greatly reduced.

Compressing the matrix using CSR compression, the matrix size is reduced to 1,6 megabytes compared to the original 3,2 MB. This equals a result where the data is compressed to 50 % of the original size. These results shows that using only five decimals with the latitude and longitude values, and transforming the matrix to CSR matrix gives the best result. Starting with 920 MB and ending with 1,6 MB is a satisfactory result. A summary of the test results can be found in table 6.1

The other proposed solution in section 5.2.1, **Matrix v2**, was to generate a matrix with fixed dimensions, allowing multiple positions to be stored inside each cell. Using this solution makes it essential to select a suitable dimension which will represent the data without a complete loss of accuracy. Selecting a dimension where each cell contains hundreds of meters of latitude and longitude values will work against the purpose of the matrix and provide false information. This project therefore set a limit to how many latitude and longitude values each cell can contain. This limit was set to 5 and is justified by the conclusion of using five decimals in the latitude and longitude values. This means that the maximum number of latitude and longitude values could not exceed 10. This results in a maximum area of $6meters^2$ for each cell. If the limit was set to more than five, the values within each cell would represent a cluster of of positions representing a larger area than $6meters^2$. Since this project concluded that using an accuracy of 4,9 meters, which would represent an area of $24meters^2$, gives a false representation, using 10 as a limit gives the same false representation. The same conclusion can be drawn using an accuracy of 4,9cm, which would represent an area of $24centimeters^2$. Setting

the limit to 1 is unnecessary because this will result in the exact same results as the other proposed matrix solution evaluated above.

The benefits of using the **Matrix v2** solution is that it forces the matrix to be smaller, which also results in a smaller volume of data sent to the browser. The disadvantage is that the matrix becomes less accurate because of the clustering of several latitude and longitude values.

This project tested the matrix transformation using five decimals for the latitude and longitude values. The matrix dimension was set to 250 x 150. This results in a cluster of approximately five latitude values and 4 longitude values which covers an area of 4,8*meters*². Recall the original size of one day of location data on 1. October 2014 of 920 MB. The transformation to a two-dimensional array resulted in a size of 165,1kB. This is 0,018% of the original size. Out of the 37.500 cells in the matrix, 26.154 cells had a value of zero. By the definition of sparse matrices, this matrix is sparse and eligible to be compressed to a CSR matrix.(Wilkinson, 1971) Compressing the matrix using the CSR compression algorithm resulted in a size of 126,1kB. This is 0,013% of the original size and 76,3% of the size of the two-dimensional array. The test results can be found in table 6.1

The project tested both matrix solutions with a larger portion of the data with 30 days of data from 2. September 2014 to 2. October 2014. This portion of the data set had an original size of 20,4GB. The test results are viewed in table 6.2. The number of cells with zero values in the two-dimensional array for **Matrix v1** was 548.449 out of the total of 781.200. The number of cells with zero values in the two-dimensional array for **Matrix v2** was 24.124 out of total of 37.500. By definition, these matrices are sparse.

Matrix solution	Original size	2-dimensional array	CSR
Matrix v1	920MB	3,2 MB	1,6MB
Matrix v2	920MB	165,1kB	126,1kB

Table 6.1: Test results from compressing the result set to a matrix using the proposed matrix solutions **Matrix v1** and **Matrix v2** using 1 day of data.

Matrix solution	Original size	2-dimensional array	CSR
Matrix v1	20,4GB	3,4MB	2,6MB
Matrix v2	20,4GB	181,6kB	162,1kB

Table 6.2: Test results from compressing the result set to a matrix using the proposed matrix solutions **Matrix v1** and **Matrix v2** using 30 days of data.

It is important to understand that a CSR compressed matrix does not double in size when querying twice the amount of data. The matrix will still have the exact same dimensions, which means the increase in size will only be the bits used to increase the number inside each cell and the additional none-zero values. The test results reveals this. The difference in the size of the two-dimensional array when compressing 1 day of data versus compressing 30 days of data, 0,2MB and 16,5kB respectively. The difference in size of the CSR matrices from 1 day of data versus 30 days of data is 1 MB and 36kB. These results shows that generating a matrix and compressing the matrix to a CSR matrix scales when working with large amounts of data.

Both solutions showed satisfactory results and solve the problem of sending large amounts of data from the database to a visualization application.

6.3.1 Possible issues

This section will describe issues related to creating a matrix to handle the size of the result set.

Small queries

When querying a tiny portion of position data the entire matrix for the area still has to be created. This might result in a slower response time for small queries compared with the original solution to push the entire JSON query result to the browser. E.g. if the query only contains 50 positions, the size of the JSON file containing those positions is only 12,7kB.

Losing additional properties for each position

When a matrix is generated, only the latitude and longitude is utilized from each registered position. This will limit the possibility to view additional information for each position, e.g. if the user wants to click on a position and view the specific timestamp registered on that position.

Floor issues

A matrix is only possible to create for one-dimensional and two-dimensional data. The position data consists of latitude, longitude and floor information. A two-dimensional matrix does not have the ability to view the floor information. This is problematic because that information is useful for the users of the system.

A workaround for this issue is to create a matrix for each floor and combine them when the user wants to view every floor. This also gives the users the possibility to view position information separately for each floor. Adding a parameter to each query and individually query each floor to generate a matrix for each floor will work in theory. Since I/O transactions are the bottleneck for this project, computing several different matrices instead of one large matrix does not increase the run

time. Creating CSR matrices for each floor will increase the size of the JSON file sent to the browser by the number of floors the user want to view. The trade off is the increase in size of the result.

Scaling

It might be difficult to make the project scale with this solution. The matrix can scale up to a certain point and change it's size by a small variable from the server, but that solution will only last as long as the matrix does not exceed the maximum size of an array. In java this is `Integer.MAX_VALUE - 8`, or 2.147.483.639 cells. The larger the area the matrix has to cover, the larger the size of the matrix. Covering e.g. the capitol of Norway, which is roughly $454km^2$, the dimensions of the matrix will be about 32574 x 46250 using latitude and longitude values with five decimals. This is equivalent to 1.506.547.500 cells in the matrix. Even though the difference between the dimensions of a matrix of Oslo and the maximum integer value in java is over 500 million, further expansion of a larger area will quickly exceed the max integer limit.

Selecting matrix dimension

Generating a matrix is a solution that requires an input from the system or the user in order to acquire information about the matrix dimension. This means that before the user can query the data, the user has to select the desired area where the matrix is to be generated based on. A solution to this problem is that the user can be prompted to mark the area on a map or input the desired minimum and maximum latitude and longitude coordinates. This requires additional interaction from the user and it can be perceived by the user as unnecessary and complex.

6.4 Performance testing

This section will explain the queries used for testing the performance of the databases and why these queries are representative for measuring the performance. It will also present a list of the fields the indexes was based on.

The following queries was used to test the databases, expressed in SQL pseudo code. The queries represent a relevant set of queries users of this system is likely to perform and was selected based on the use case examples described in section 3.6 and possible queries in section 5.1.

1. **SELECT latitude, longitude FROM table WHERE timestamp >X AND timestamp <Y**

Query number one queries between two dates without any other parameters. This is useful for the users to view every position registered between two dates. This query satisfies use case number one where a user wants to view location data in a specific time period.

2. **SELECT latitude, longitude FROM table WHERE >X AND timestamp <Y AND hourOfDay >9 AND hourOfDay <15**

Query number two represent queries where the user want to retrieve data in a more specific time frame by specifying the hour of the day. An example is a building manager who wants to view where there is a cluster of people during lunch hours. This query satisfies use case number two where the user wants to view location data based on specific hours of the day.

3. **SELECT latitude, longitude FROM table WHERE >X AND timestamp <Y AND hourOfDay >9 AND hourOfDay <15 AND (dayOfWeek = 1 OR dayOfWeek = 3)**

Query number three is similar to query number two, but it adds the dayOfWeek parameter to the query. This query satisfies use case number three where the user wants to view location data based on specific hours of the day within a time period and on a specific day of the week.

4. **SELECT latitude, longitude FROM table WHERE >X AND timestamp <Y AND hourOfDay >9 AND hourOfDay <15 AND building = "IT-Vest"**

The fourth query represents a more specific query than query two and three where the building is specified. This is useful for the same reason as described for query two and three, but limited to a specific building. Having the possibility to limit queries on buildings is useful if the user wants to exclude irrelevant information. The data set contains forty-four different buildings. By excluding forty-three irrelevant buildings the number of positions returned to the user will decrease and the user will experience a faster execution time. This query satisfies use case number four where the user wants to gain knowledge about the abrasion in a specific building.

5. **SELECT latitude, longitude FROM table WHERE timestamp >X AND timestamp <Y AND accuracy <10**

Query number five was created to satisfy use case number seven, where the user wants to view location data based on the accuracy. Previous work (Aulie, 2015; Eriksen, 2015) created prototypes that enabled the user to query data with a specified accuracy related to the average accuracy of the data set.

6. **SELECT latitude, longitude FROM table WHERE timestamp >X AND timestamp <Y AND id = 10**

Query number six is a typical query used to track a device during a day. This query satisfies use case number six where the user wants to track the movement of a specific device. Since one of the areas of usage for the data is to track movement from building to building, querying on a specific *id* gives the user the possibility to track devices.

7. **SELECT latitude, longitude FROM table WHERE timestamp >X AND timestamp**

<Y AND hourOfDay >9 AND hourOfDay <15 AND (dayOfWeek = 1 OR dayOfWeek = 3) AND building = "Realfagbygget" AND floor = "4.etasje"

Query number seven is a combination query that contains the most important fields in the data set. A database system should be able to provide the possibility to retrieve data based on several parameters to support the more specific requirements from the user.

8. **SELECT latitude, longitude FROM table WHERE timestamp >X AND timestamp <Y AND building = "Realfagbygget"**

The last query was created to satisfy use case number five where the user wants to observe how the usage of buildings change over time.

The X and Y values is the timestamp value describing the date range. To determine what X and Y values the queries should use, the project suggested testing the queries with different X and Y values. This forces the project to test the queries in a short time period and over a long period. Additionally, one of the queries will not work as expected if the X and Y values are set to two different days. This is query number six with the *id* parameter. Since the id is only unique within a single day, setting X and Y to 1. October 2014 and Y to 20. October 2014 will return a different device for each day. This works against its intention and will return useless data. This project therefore performed the queries with the following parameters for X and Y in the queries described above. Since each date is a timestamp, the time was set to 00:00 for each date.

The list below describes the dates used in the tests.

1. X was set to 1. October 2014 and Y was set to 2. October 2014, which is one day.

Testing with these parameters is useful to evaluate if the database system has a high read performance even for small portions of the data.

2. X was set to 1. October 2014 and Y was set to 8. October 2014, which is seven days.

These parameters makes the query return one week of data. This is useful to evaluate the performance when the data range is larger than one day. Testing query number six with these parameters is unnecessary as explained above.

3. X was set to 1. October 2014 and Y was set to 1. November 2014, which is thirty days.

These parameters represents approximately half of the data set. This is useful to evaluate if the query results in a full table or collection scan, or if it uses the indexes to exclude irrelevant data. Query number six will not be tested with these parameters as explained above.

For each database implemented and tested in this project, indexes based on the following fields were created:

1. **timestamp** - every test query includes the timestamp field. Previous work (Aulie, 2015; Eriksen, 2015) created prototypes where the start date and end date had to be specified. The requirements states that the intentions with the data set is to view the data over a specific time period, which makes the timestamp an important property for the location data.
2. **hourOfDay** - indicates what hour of the day the position data was recorded. Query number two queries on this field which makes it eligible to be indexed.
3. **dayOfWeek** - indicates what day of the week the position data was recorded. This field is used in query number three which makes it eligible to be indexed.
4. **id** - indicates a specific device. This field is a vital part of query number six and is therefore qualified to be indexed.
5. **accuracy** - indicates the accuracy of a registered location. This field is used in query number five which makes it eligible to be indexed.
6. **building** - indicates what building the position belongs to. This field is used in query number four, seven and eight. An index based on the building field is therefore necessary.
7. **floor** - indicates the floor the position was registered on. The floor field is used in query number seven which makes it eligible to be indexed.
8. **timestamp and id** - Creating this combined index should further improve the execution time when performing query number six.
9. **timestamp and hourOfDay** - Creating a combined index based on these two fields should improve the execution time for query number two.
10. **timestamp and dayOfWeek** - Creating a combined index based on timestamp and dayOfWeek may aid the database to improve the execution time on query number three and query number seven.
11. **timestamp and accuracy** - Creating a combined index based on timestamp and accuracy gives the database the possibility to improve the execution time when performing a query number five.
12. **timestamp and building** - This index was created to support query number eight. These are the fields used in the query and should be used by the database to improve the execution time.
13. **timestamp and hourOfDay and dayOfWeek** - This index was created to be used by query three. Query number seven also contains these fields which gives the database the possibility to use it.

14. **timestamp and hourOfDay and building** - This index was created to be used by query number four. Query number seven also contains these fields.

6.5 MongoDB

This project implemented and tested MongoDB based on the requirements (see section 1.4).

6.5.1 Implementation

MongoDB version 3.2 was implemented on a single node cluster. In order to use and test the system, an application in java was implemented where MongoDB performed queries and generated a matrix as described in section 6.3. As stated in section 5.3.1.1, indexes should be created in MongoDB. This lead this project to create the indexes described in section 6.4.

6.5.2 Testing

Testing the MongoDB system with the queries in section 6.4 was performed in a Java environment. Figure 6.1 shows the test results from the queries described in section 6.4. The results shows that MongoDB produces acceptable results when querying small amounts of data. Four of the eight test queries had an execution time of six seconds or less for one day of data, but have an increase in execution time when performing the same query on seven and thirty days of data. Query number one where the query returns every position within a timestamp thirty days produces poor results.

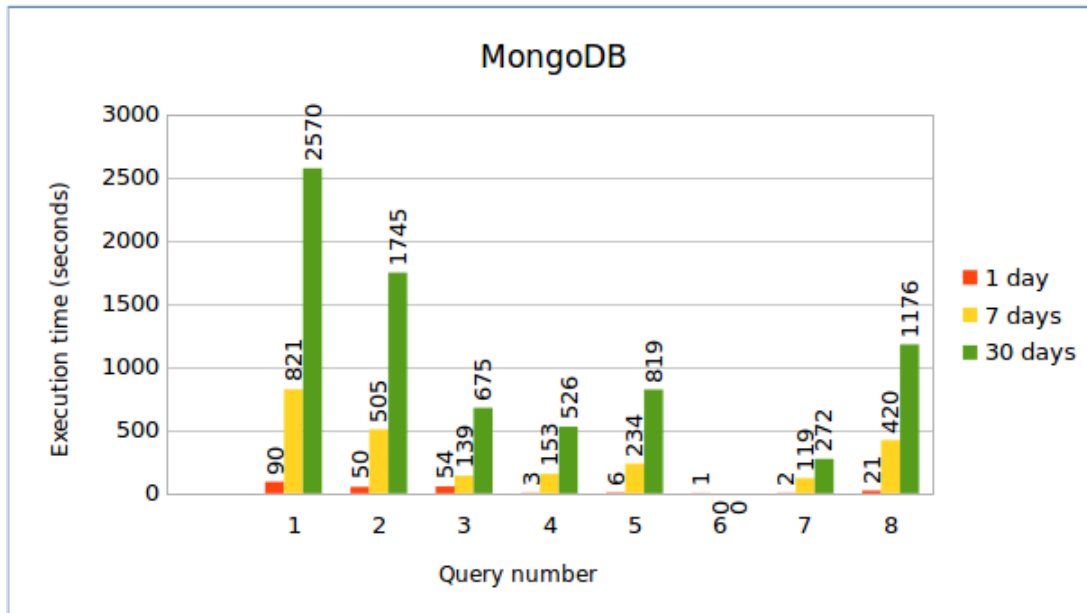


Figure 6.1: A summary of the results of test queries in MongoDB.

The test results in figure 6.1 shows that MongoDB is able to handle large amounts of data in a collection and produces the correct results. MongoDB also provides the possibility to distribute the collection across several servers as discussed in section 5.3.1.1. This project implemented sharding in MongoDB to evaluate the impact sharding had on the read operations. The test was performed using query number one with a timestamp range of one day. The average read operation took 90 seconds with no sharding. The average read operation with sharding took 115 seconds. Performing the same test with a timestamp range of seven days had an execution time of 821 seconds with no sharding. With sharding implemented, it took 1304 seconds. These results showed that there is a trade off between implementing shards to achieve a higher scalability property and the read performance.

As discussed in section 5.3.1.1, MongoDB supports replication. Replication provides a system with high availability as specified in this projects requirements. Implementing replication is considered vital in a production environment, but is not required for testing and development. Because of the time limitation this project had, implementing replication was not prioritized.

Batch size

Setting the batch size in MongoDB explains how many documents to be fetched for each call to the database. If the batch size is set to 10, and there are 100 documents to be returned, 10 round trips will be made. The project tested different batch sizes on the data set and discovered that different batch sizes gave different run time for a query. The tests were performed in three iterations using

query one where the timestamp range was set to one day, seven days and thirty days. Tables 6.3, 6.4 and 6.5 depicts the results. It also ranks each batch size based on the lowest average run time. The default batch size is 101 documents, or enough documents to exceed 1 megabyte for the first batch. The subsequent batch size is 4 megabytes.¹ The default values in table 6.3 represents queries where the batch size were not set.

Batch size value	Execution time	Rank
10	268 s	7
100	101 s	6
1.000	79 s	4
10.000	76 s	1
100.000	76 s	1
1.000.000	76 s	1
Default	90 s	5

Table 6.3: Batch size values in MongoDB querying one day of data.

The results from experimenting with batch size proved that there is an advantage to set the batch size rather than using the default value for small queries. The difference proved to be 14 seconds which is a about $\frac{1}{5}$ of the result from the fastest query.

Batch size value	Execution time	Rank
10	3156 s	7
100	980 s	6
1.000	745 s	1
10.000	745 s	1
100.000	813 s	3
1.000.000	922 s	5
Default	821 s	4

Table 6.4: Batch size values in MongoDB querying seven days of data.

Table 6.4 shows a different result than table 6.3. The only certain conclusion from these tests are that setting a low batch size, e.g 10 or 100 gives a low run time. Setting the batch size to 1.000 or 10.000 has the best performance in both tests and indicates that a fixed batch size value performs

¹<https://docs.mongodb.com/manual/core/cursors/> last visited 27.02.16

better than the default setting.

Batch size value	Execution time	Rank
10	10560 s	7
100	3542 s	6
1.000	2744 s	4
10.000	2689 s	3
100.000	2771 s	5
1.000.000	2647 s	2
Default	2570 s	1

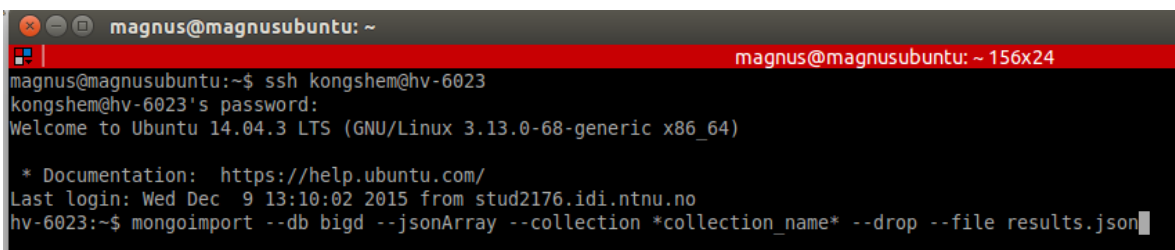
Table 6.5: Batch size values in MongoDB querying thirty days of data.

Table 6.5 shows the results from query testing with different batch sizes using a timestamp range of thirty days. As previous tests revealed, a low batch size is not optimal and has the poorest result for large queries. The results shows that the default value has the best performance by 78 seconds. The previously best batch sizes was 175 and 120 seconds slower.

Based on the results from the three tests, the default batch size has the best performance on the largest queries. This project concludes that the default batch size gives the best results for this project.

6.5.3 Other

MongoDB provides a shell interface to import JSON data to the database. The command in figure 6.2 shows how the import command is performed inside the MongoDB shell.



```

magnus@magnusubuntu: ~
magnus@magnusubuntu:~$ ssh kongshem@hv-6023
kongshem@hv-6023's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-68-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed Dec  9 13:10:02 2015 from stud2176.idi.ntnu.no
hv-6023:~$ mongoimport --db bigd --jsonArray --collection *collection_name* --drop --file results.json

```

Figure 6.2: Import JSON in mongoDB

The import process is seamless and fast in MongoDB. Importing the entire data set in this project had an execution time of 45 minutes. Since this projects data set is a JSON file, the import process

is an important part of the system. If additional data is to be inserted into the database, the same command can be used. In addition to using the MongoDB shell to import data, the API provides the same functionality.

6.6 MySQL

6.6.1 Implementation

MySQL was implemented with the InnoDB engine where the entire data set was stored in one table. Indexes was stored with the B+tree data structure and created based on the fields described in 6.4.

6.6.2 Testing

When the project first began to test the queries, the execution time and explain function in MySQL revealed that some of the queries did not use the created indexes or used the wrong index. Testing MySQL with seven days of data revealed that query number one used the combined index based on the timestamp and hourOfDay fields. Query number five also used the wrong index because it used an index based on timestamp and hourOfDay. Because of this, the project had to use the *USE INDEX()* function in MySQL to aid in the selection of index for each query. Query number one was set to use index **timestamp** and query number five was set to use the index based on the **timestamp and accuracy**. Figure 6.3 shows the test results from the queries described in section 6.4.

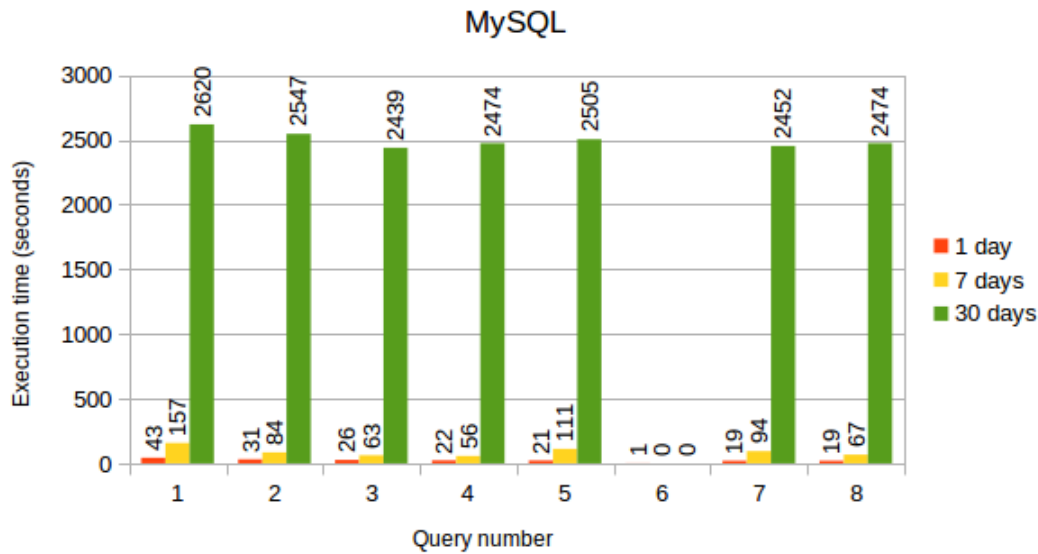


Figure 6.3: A summary of the results from test queries in MySQL.

Since the data schema is structured, the MySQL performance for small amounts of data was expected. An example is query number six where the query is asking for a specific device ID. For this explicit use case, MySQL is extremely fast. The trade off for fast execution time for query number six is the slower execution time for the other queries.

The poor results MySQL produces on the largest queries comes as a result of MySQL's vertical scaling. There are solutions to distribute MySQL across a cluster, but with major trade offs. Managing and maintaining the machines comes at a high cost, the level of difficulty to handle node failures on the code side, and choosing between availability and consistency. Splitting a table and distributing across several servers requires heavy join operations when querying the table which is time consuming. Because of the time limitation this project had, implementing MySQL in a cluster was not prioritized. This priority was based on the read performance results in figure 6.3.

To ensure no data is lost, *mysqldump* can be performed to produce a set of SQL statements that can be used to reproduce the original database and table data. Replication is also possible by keeping an exact copy of the database on a different server. Replication is recommended for a production environment, but not required for testing and development. Replication was therefore not implemented by this project.

6.6.3 Other

Importing the data set is less optimized for MySQL than for instance MongoDB. MySQL does not provide any import commands in the shell and the data set had to be inserted manually through a program written by this project. The program had to deserialize the JSON files and insert the position data in batches. Importing 1.000.000 positions took on average 30 minutes. Importing the entire data set took 83 hours.

6.7 AsterixDB

AsterixDB was implemented and tested with regards to the requirements in section 1.4.

6.7.1 Implementation

AsterixDB was implemented as a standalone system on a single cluster. AsterixDB uses different names on databases, tables and types than what is normally used. Databases are called *data verse* and a table is called a *data set*. When creating a data verse with a data set, a type describing the fields has to be specified. The database was initialized with indexes described in section 6.4. As described in section 5.3.4, AsterixDB supports filter-based LSM index acceleration. Creating a filter based on the timestamp field could potentially save up to 99% of query time.(Alsubaiee et al., 2015) Since the data set does not contain any unique fields to create primary keys on, the database was initialized with a UUID field named *uid*. As of the current version of AsterixDB, 0.8.8, creating filters on a data set with an auto generated UUID field is not supported. In order to test the system with filters the project had to generate a UUID field in the data set before inserting it into the database. This also lead to an increase in the size of the data set.

6.7.2 Testing

Testing AsterixDB was performed in two iterations in the provided web interface of AsterixDB. The first test iteration was performed on a data set where AsterixDB auto generated the UUID field and without using the filter functionality. The second iteration was performed on a data set with a filter based on the timestamp and with individual indexes based on the *floor and id* fields. No combined indexes was created. The DDL of the data sets for the two test iterations is located in appendix D. Appendix A shows a screen shot of the web interface with an example query performed.

There were limited options in AsterixDB to perform a query without returning the entire result set to the console, which in this project is millions of rows. An email correspondence with the AsterixDB

developers, see appendix C led this project to using one of two solutions: Using the **limit clause** on the query to limit the amount of returned rows, and using the **count()** function. In asterixDB, adding a limit to a query does not mean that the query stops when it has reached the number of matching rows. It keeps iterating until all matching rows are found and then returns the amount of rows set by the limit parameter. The **count()** function can be used to force AsterixDB to find every matching row and returning the number of matching rows. This project used the **count()** function to evaluate the queries as recommended by the AsterixDB developers.

Figure 6.4 shows the results from the first test iteration with results for each timestamp range. The performance on the small timestamp range is considered poor compared to the expectations and the other databases tested. The project noticed that the execution time does not increase when increasing the timestamp range, except for query number one and eight. This suggests that AsterixDB is able to scale when working with larger amounts of data. Query number six produces a poor result which is surprising since the query uses the theoretically correct index based on the *id* field. Analysing the queries execution plan, the queries mainly used the indexes based on the *timestamp* and *hourOfDay* field.

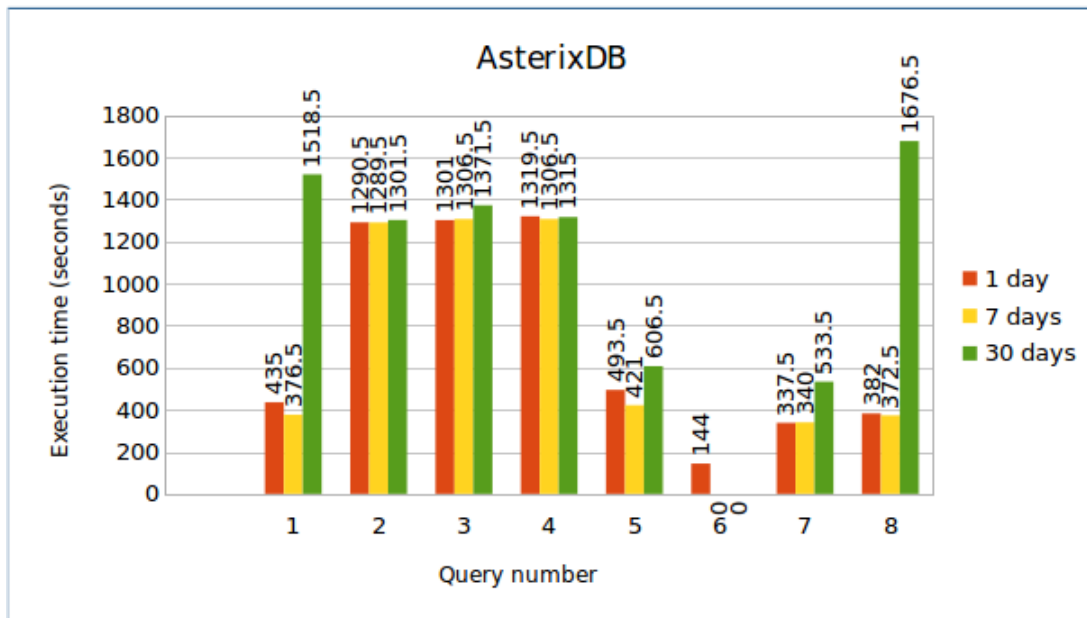


Figure 6.4: A summary of the results from test iteration 1 in AsterixDB.

Figure 6.5 shows the results from the second test iteration performing the test queries on a data set using filter on the *timestamp* field. Looking at the queries with the largest timestamp range, this test iteration has a better performance on seven out of the eight queries compared to test iteration 1. The read performance is therefore considered to be better using the filter-based LSM index in AsterixDB.

The project also observes that the execution time of the queries with different timestamp ranges is nearly equal. The same trend is observed in test iteration 1. The test results for query number six is really poor, even though the query plan shows the usage of the index based on the *id* field.

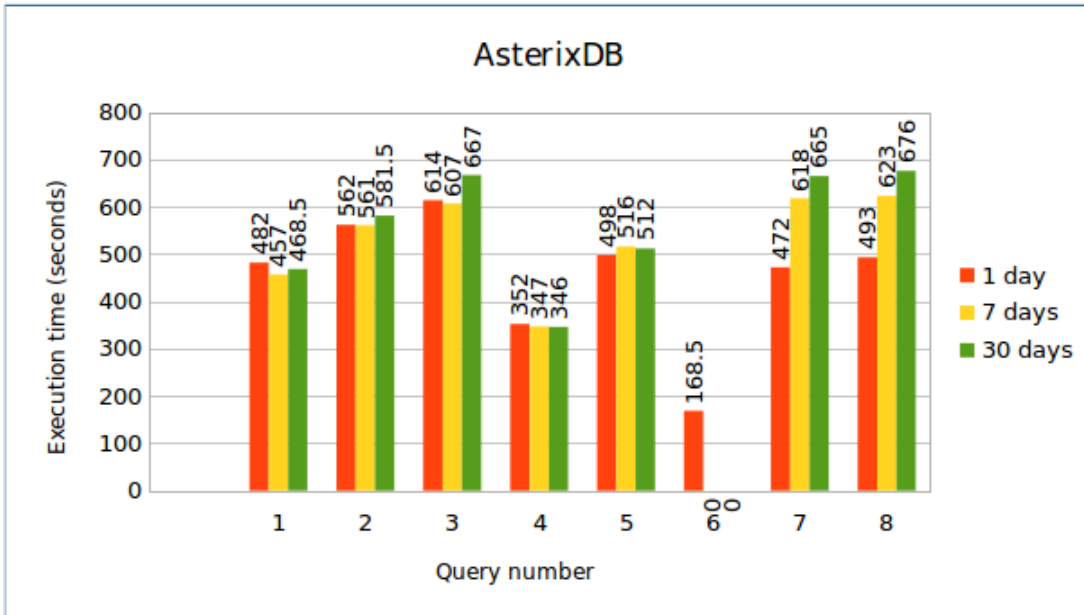


Figure 6.5: A summary of the results from test iteration 2 in AsterixDB.

Comparing the two test iterations of AsterixDB, test iteration two produces the best results and will be used in future comparisons in this thesis.

Related to scalability AsterixDB supports the possibility of implementing the instance on a cluster of multiple machines ². This architecture pattern is based on the Master-Slave technique. The idea is to have a master node controlling the slave nodes where the computations are performed. This pattern supports fault tolerance and parallel computation because the master node distributes the work among the slave components. (Bass et al., 2012)

As described above, AsterixDB can be implemented with the Master-Slave architecture pattern on multiple machines. This architecture pattern supports fault tolerance which satisfies the one of the requirements for this project. (Bass et al., 2012) AsterixDB also supports backup of instances to enable restoring an instance if something unanticipated happens.

²<http://asterixdb.apache.org/docs/0.8.8-incubating/install.html> last visited 02.06.16

6.7.3 Other

The current version of AsterixDB, 0.8.8, has a poor support for querying a sub-string in a cell. Related to the data set for this project, see section 4, searching for a specific floor or building in the hierarchy is a possible use case. This drawback can be avoided by splitting the hierarchy field as suggested in section 6.1.

Transforming the data set to the schema required by AsterixDB was completed with a simple command in Perl. The command removes the [and] brackets indicating a list, and removes the , between each object. The `-0` option in the perl command defines the record separator since the data set did not contain any line breaks. 173 stands for the character { in octal. This forces Perl to read the file in chunks between each { character to avoid loading the whole file into memory.

```
perl -0173 -pe 's/},{/}{g;y/][//d' < source.json > newFile.adm
```

The import process itself had a run time of three hours (11017 seconds). If additional data needs to be added, a complex process has to be initiated. First a new data set has to be initialized and loaded with the new data. Lastly one has to iterate through that data set and insert it into the original data set. An e-mail correspondence with the AsterixDB developers revealed that a simpler way of adding additional data will be supported soon.

Using AsterixDB in an application requires using their *HTTP API* or cloning a snapshot of the AsterixDB source code and include it in the application. This is a cumbersome and time consuming process. Encoding and decoding the url's to match the expected syntax is not well documented ³ and including the source code increases the complexity and scope of an application.

6.8 Comparison

Read performance

As figure 6.6 depicts, MySQL and MongoDB has a better performance when performing the smallest queries. AsterixDB performs extremely poor on these queries compared to MongoDB and MySQL. Query number six covers use case number six, and since AsterixDB uses 241 seconds compared to MySQL and MongoDB using 1 second, tracking the movement of a device is more suited to be performed in MySQL or MongoDB. The results from the other queries shows MySQL and MongoDB take turns having the best read performance. This project therefore concludes that querying a small portion of the data MySQL is the best choice, but MongoDB produces acceptable results.

³<http://asterixdb.apache.org/docs/0.8.8-incubating/api.html>

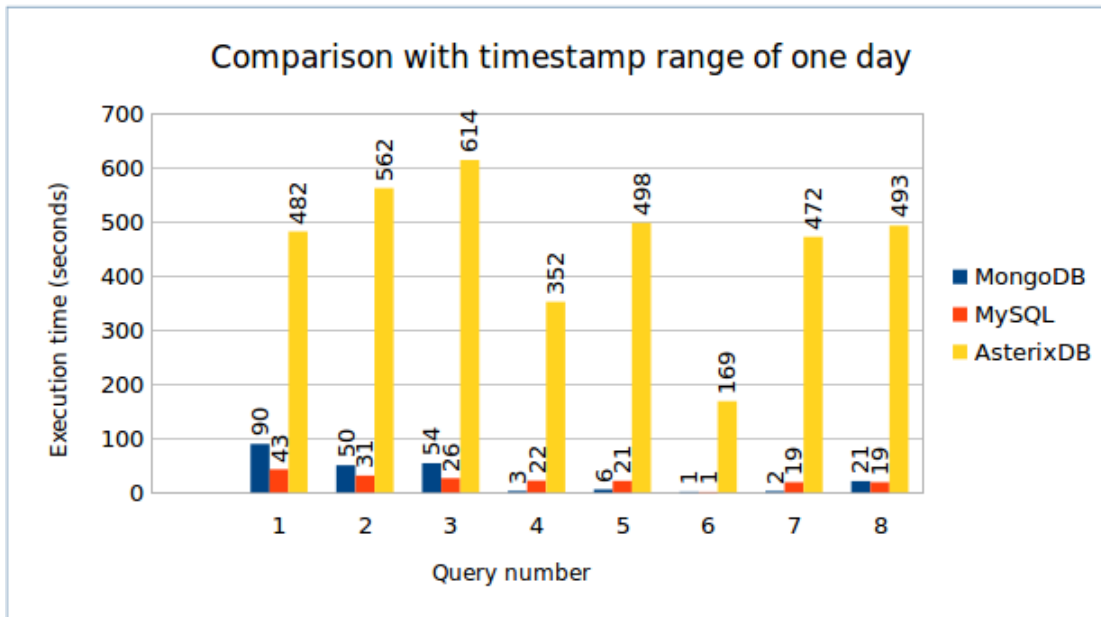


Figure 6.6: Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to one day.

Figure 6.7 shows the results from each database querying a larger portion of the data where the timestamp range was set to seven days. The results show that MongoDB fails to preserve the low execution time from figure 6.6 and passes AsterixDB on query number one. This indicates that MongoDB performs badly when querying larger amounts of data. AsterixDB maintains approximately the same execution time for each query, which indicates that the performance of AsterixDB does not drop on larger queries. MySQL maintained the best read performance for each query which means that MySQL is the preferred database for this amount of data.

Figure 6.8 shows the results from each database querying with a timestamp range of 30 days. The test results reveal that AsterixDB has the best read performance for six of the seven queries, not counting query number six. This might be because the number of matching records (257 715) is significantly lower for this query compared to the other queries. MySQL and MongoDB prove that they work well with smaller queries, but lack the support for larger queries. The spike in read performance for MySQL reveals that MySQL is not meant to handle large amounts of results. MongoDB produces a similar spike in read performance, but not as significant as MySQL. This proves that MongoDB is able to handle larger results better than MySQL.

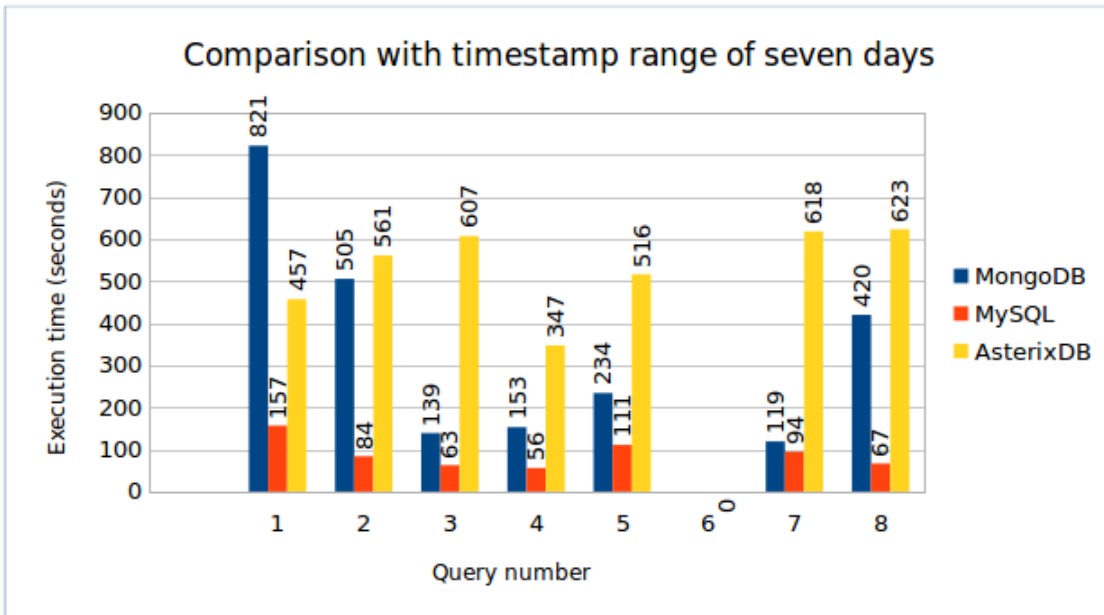


Figure 6.7: Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to seven days.

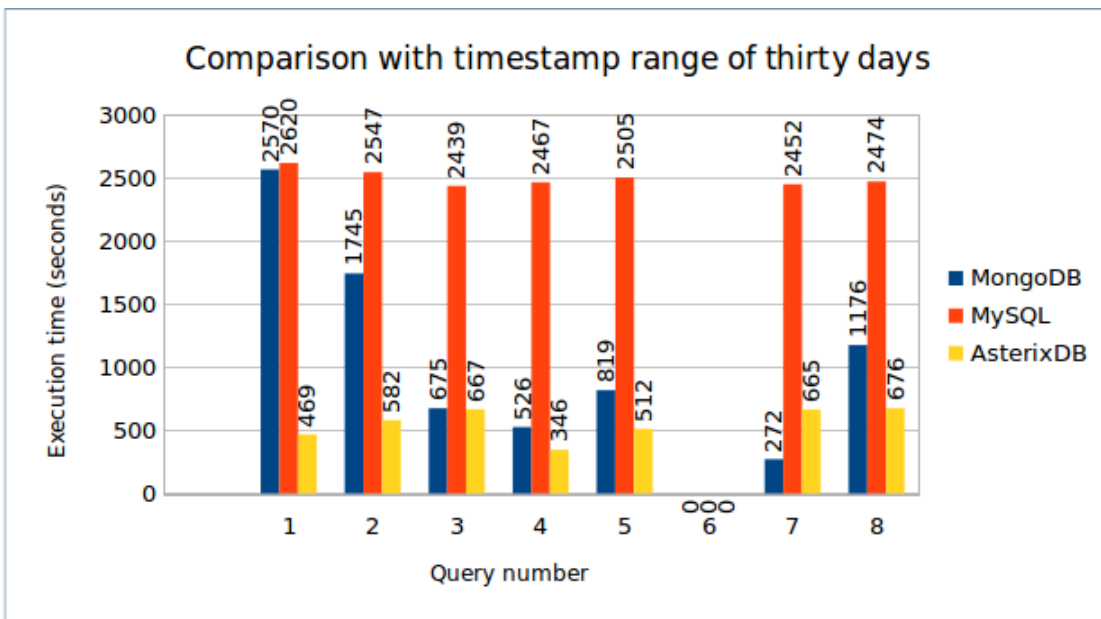


Figure 6.8: Performance results of test queries in MongoDB, MySQL and AsterixDB where the timestamp range was set to thirty days.

MySQL is not a big data tool, hence are the performance results as expected. When querying a specific ID, the timestamp range can only be one day at a time or else the query would return false

information. MySQL and MongoDB shares the best performance for this query. AsterixDB is much slower which is problematic considering the use cases from section 3.6. This is considered a possible trade off with AsterixDB's faster performance related to the other use cases. Theoretically the best solution would be to use AsterixDB for the large and complex queries and use MySQL or MongoDB when performing query number six. This solution would complicate the resulting artifact for several reasons. Firstly, the data set has to be update two places when new data is added. Maintaining two databases with the exact same data is troublesome and not ideal for any project. Keeping track of what data set is added to each database will in this projects opinion cause trouble at some point. Secondly, the resulting artifact has to interact with two databases with different architecture, different syntax for developers and should be installed on two different servers. Another disadvantage is the amount of disk space needed to store two versions of the same data set.

As described in section 5.3.1.1, MongoDB is a document data store. It stores JSON documents which are grouped into collections. It does not impose the data to have a set schema structure and the only limitation are the data types. Since there always is a possibility that the data set could change in the future, MongoDB is more dynamic than any other RDBMSs. MySQL is a relational data store which means the data store requires a strict schema for the data model. All tables has to be created with defined columns. This complicates the development and deployment process because every time the data model needs a modification, the schema has to be changed and the data has to be migrated. The data set used in this project had a fixed schema and continued to have a fixed schema after this project had made the modifications described in section 7.1. MySQL outperformed MongoDB on the smaller queries which proves that MySQL was a liable candidate for this project if the performance for the largest queries was better.

Scalability

MongoDB supports horizontal scaling by distributing the data set over multiple shards. Related to the read performance, MongoDB has trouble keeping a profound execution time. MySQL by default does not support horizontal scaling, but vertical scaling which is not desirable in a big data solution. AsterixDB supports horizontal scaling through it's shared-nothing database structure and the use of the Hyracks engine. Based on these properties and the test results, MongoDB and AsterixDB has the desired scalability property.

Availability

All three databases tested in this project provides a solution to ensure consistency in the data set and provide high availability. MongoDB and MySQL supports replication. Sharding in MongoDB also adds redundancy and fault tolerance to the system. AsterixDB has the basic transactional capabilities

like traditional NoSQL databases such as concurrency and recovery. It also has a solid crash recovery policy to ensure consistency on the data set.

6.9 Choosing the final artifact

As section 6.8 describes, AsterixDB has the best read performance when the query range increases in size. This is a significant part towards selecting the final artifact. Both MongoDB and AsterixDB provides a satisfactory solution to scale the database across multiple nodes and provide parallel processing, MySQL does not. Based on the fact that MySQL falls short on these two measures, this project concludes that MySQL is not a desired artifact for this project. All three databases provide a method to achieve high availability, which makes them all eligible to be chosen as the final artifact based on the availability requirement. Considering all three requirements, read performance, scalability and availability, AsterixDB is the only database that satisfies every requirement and is therefore selected as the final artifact for this project.

Chapter 7

Result

This chapter will present the results of this project. First the solution to transform the data set will be explained. Then the selected solution to decompress the result set will be presented. Finally the artifact selected will be revealed.

7.1 Data set

This section will explain how the data set was processed and altered.

One of the problems with the structure of the data set was that the timestamp did not provide clear information about the day of the week, or the hour the position data was recorded. therefore two new variables named "day_of_week" and "hour_of_day" was added to each data record.

Another issue with the data from the posData server was that the longitude and latitude coordinates were switched. This means that the longitude value actually is the latitude value, and vica-verca. This project therefore switched the values back in correct order.

The hierarchy field in the data set contained a composition of multiple properties for a position point. When querying a specific building or floor, the hierarchy field had to be analysed and broken into three parts in order to collect the required information. This operation was time consuming and unnecessary work for a server to perform. This project divided the *hierarchy* field into three parts, *Campus*, *Building* and *Floor*, and deleted the original hierarchy field.

When the described modifications was performed on the data set, the structure of the data set changed. Figure 7.1 shows the current structure. Appendix B shows the java function used to perform these modifications. The operation was conducted seamlessly with the Jackson Streaming API in Java 7, with a run time of fifteen minutes. The java program is also attached to this thesis.

```

1  [
2  {
3    "campus": "Gløshaugen",
4    "building": "Varmeteknisk og Kjelhuset",
5    "floor": "3. etasje",
6    "timestamp": 1412121600,
7    "dayOfWeek": 3,
8    "hourOfDay": 2,
9    "latitude": 63.41904008880636,
10   "salt_timestamp": 1412121602,
11   "longitude": 10.405133392801218,
12   "id": "4",
13   "accuracy": 221.8943996627205
14 },
15 {
16   "campus": "Gløshaugen",
17   "building": "Gamle kjemi",
18   "floor": "Kjeller",
19   "timestamp": 1412121600,
20   "dayOfWeek": 3,
21   "hourOfDay": 2,
22   "latitude": 63.418086246212226,
23   "salt_timestamp": 1412121602,
24   "longitude": 10.403139597241832,
25   "id": "110",
26   "accuracy": 56.083199914753536
27 },

```

Figure 7.1: The structure of the data set after the modifications performed by this project.

7.2 Matrix

Based on the evaluation in section 6.3 the project chose to implement **Matrix v1** using latitude and longitude with five decimals and utilizing CSR compression. The project wrote a java program that takes a two-dimensional matrix as a parameter and produces a JSON object containing the three one-dimensional arrays and the necessary values needed to decompress the matrix. The java program is attached to this thesis. Attached is also a HTML5 and JavaScript solution to decompress the matrix and paint a Google heat map ¹ based on the matrix. This was provided to give future projects an idea on how to decompress the matrix.

¹<https://developers.google.com/maps/documentation/javascript/examples/layer-heatmap> last visited 04.06.16

7.3 AsterixDB

Based on the technical evaluation in chapter 6, AsterixDB was selected as the final artifact. AsterixDB proved to have the best read performance out of the other tested database systems and provided a satisfactory solution as a database for the data set in this project. AsterixDB fulfills the scalability requirement this project was seeking for the database and provides satisfactory solutions to provide high availability and fault tolerance. AsterixDB was installed on the project server and the web interface (see appendix A) can be accessed at <http://129.241.106.23:19001/> when connected to an access point at NTNU. The dataverse was named *bigd*, the data set was named *posdata3* and the type was named *table5*. The current version of AsterixDB does not provide an way of renaming a dataverse, type or data set. Attached to this thesis is a java program accessing the HTTP API of the AsterixDB instance. The java program performs the test queries and can be used to interact further with the data set. Attached to this thesis is a guide on how to access the AsterixDB instance and interact with it in the terminal. The file is named *How to interact with the AsterixDB instance.txt*. Appendix E shows how future projects can import more data into the database based on the current version of AsterixDB in AQL. The current solution is to bulkhead data to a new temporary data set, and insert it into the desired data set. It is likely that a more dynamic solution will be available in future releases of AsterixDB.

Chapter 8

Discussion and future work

The following chapter will provide a discussion of the results anchored in the research questions for this project and provide recommendations for future work.

8.1 Discussion

The project was able to implement an artifact satisfying the system requirements described in section 1.4. The project revealed that performing large queries on this projects large data set does not produce desirable results in MySQL or MongoDB. The read performance increases as the query results increase which proves that they do not provide a scalable architecture for this projects data set. This evaluation answers RQ1 from section 1.3.

This project chose AsterixDB as the BDMS to evaluate it's performance compared to MySQL and MongoDB. An AsterixDB instance running on a single machine produced better results on the data set, hence outperforming MySQL and MongoDB. This conclusion answers RQ2 from section 1.3.

The alterations this project made to the data set resulted in five new fields and the removal of one field. This made the entire data set increase in size. The original size of the entire data set was 34,2 gigabytes. The new size is 39,1 gigabytes. The project chose a matrix decompression algorithm as it's solution. The algorithm solves the size issue and therefore answers RQ3, but has some disadvantages such as losing attribute values, scaling to a larger area than the campus of NTNU Gløshaugen, handling small queries, representing the data in three dimensions and selecting the matrix dimension. The projects focus on reducing the size of the results did not take into account these disadvantages which has to be re-evaluated if future projects want to retrieve that information.

As described in section 6.1, previous work grouped smaller buildings into larger groups to improve

usability. This project did not alter the data set to support grouping and this was not implemented in the database used in this project. The argument is that this limits the user's ability to view more detailed data. This project's focus was to transform the data set into a format that better explained each position, not limit them.

This project did not implement a clustered version of any of the databases due to time limitation. A big data management solution should be implemented on a cluster of multiple machines to observe the behavior compared to an instance running on a single machine.

8.2 Future work

The purpose of this project was to find and implement a database architecture for position data. With limited time, this project was not able to investigate every possible big data database architecture that exist. Therefore, this section offers recommendations for future work to test their performance against this project's results. Finally, this section will explain why future work should investigate alternatives to compress the data.

8.2.1 Integrating the artifact with the previous created prototypes

Now that this project has provided the original research project with a scalable database for the data set, future work has to integrate the artifact with the visualization prototypes created by Aulie (2015) and Eriksen (2015). The prototypes were created to accept a JSON file containing the result from a query, which has to be altered to accept the matrix this project decided as its solution to send to a web application. Attached to this thesis is a javascript program providing an example on how the CSR matrix can be uncompressed and read to extract the information.

8.2.2 Data cleaning

This project did not perform data cleaning on the data set in terms of obvious inaccurate positions, such as positions registered outside the rectangle depicted in figure 4.3 or positions registered outside a building on higher floor than the ground floor. Future work should consider writing a program that removes noise from the data set before it is inserted into this project's solution.

8.2.3 Investigate other database architectures

This project suggests testing Hadoop on this project's data set. In addition to implementing Hadoop MapReduce jobs, future works should look into applicable tools on top of Hadoop. This project suggests

testing HBase and Spark. HBase is a distributed database that supports structured data storage for large tables, it scales horizontally and provides random, real time read/write access. Spark should also be tested because it uses resilient distributed data sets and it supports large batch calculations in-memory, which can be useful when performing parallel computations.

8.2.4 Workload

Working with big data is a time consuming process. Each big data tool has a different way of importing data into the database, supports different data set schemes, supports different query languages, strict tool dependencies, strict tool version dependencies and various quality in the documentation. This means that based on the principal a project starts with, data transformation is certain to happen. Related to this project, testing AsterixDB required the project to convert the data set from JSON to ADM by altering the structure of the data set. In addition, testing the performance of a database with big data leaves the developer with a long waiting time between the tests. Potential future work should take this into account when continuing the development.

8.2.5 Look at other data compression solutions

This projects solution regarding data compression of the results from a query is efficient and considered satisfactory. As presented in section 6.3.1, the issues related to this solution is selecting the matrix dimension, scaling, loosing data property, small queries and floor issues. This thesis encourages future work to investigate other solutions where these issues does not apply.

Appendices

Appendix A

AsterixDB web interface

The screenshot displays the AsterixDB web interface. At the top, the AsterixDB logo is on the left, and navigation links for "Open source", "File issues", and "Contact" are on the right. The main interface is divided into two columns: "Query" and "Output".

Query Section:

- A text area contains the following SQL query:

```
use dataverse bigd;

from $obj in dataset posdata
where $obj.timestamp >= 1412121600 and $obj.timestamp
<= 1412121601
return $obj;
```
- Below the text area are three buttons: "Select Options", "Clear Query", and "Run".
- An "Output Format:" dropdown menu is set to "ADM".
- A list of checkboxes for output options:
 - Print parsed expressions
 - Print rewritten expressions
 - Print logical plan
 - Print optimized logical plan
 - Print Hyracks job
 - Execute query

Output Section:

- The "Logical plan:" section shows a query plan:

```
distribute result [%0->$$0] -- [UNPARTITIONED]
project ([$$0]) -- [UNPARTITIONED]
select (function-call: algebricks:and, Args:[function-call: al
unnest $$0 <- function-call: asterix:dataset, Args:[AStr
empty-tuple-source -- [UNPARTITIONED])
```
- The "Results:" section displays a list of JSON objects, each containing a "uid" and a "campus" field. The list is truncated with a scroll bar.
- Below the results, a status bar indicates "Duration of all jobs: 0.129 sec".
- A green success message at the bottom reads "Success: Query Complete".

Figure A.1: Screen shot of the web interface of AsterixDB.

Appendix B

Data alteration

This appendix shows the short java function written by this project to create the `hour_of_day` field, `day_of_week` field, switching the latitude and longitude values back to their correct place in each record, and how the `hierarchy` field was broken down into three new fields.

```
protected void alterDataSet(String path, String sourceFile, String outputFile){
    try {
        JsonFactory jsonfactory = new JsonFactory();
        File source = new File (path+sourceFile);
        System.out.println("Leser filen "+source.getName());
        JsonParser parser = jsonfactory.createParser(source);
        JsonFactory jfactory = new JsonFactory();
        /** write to file ***/
        JsonGenerator jGenerator = jfactory.createGenerator(new File(
            path+outputFile), JsonEncoding.UTF8);
        jGenerator.writeStartArray(); // [
        while (parser.nextToken()!=JsonToken.END_ARRAY) {
            jGenerator.writeStartObject(); // {
            while (parser.nextToken() != JsonToken.END_OBJECT){
                String token = parser.getCurrentName();
                if("hierarchy".equals(token)){
                    parser.nextToken();
                    String hierarchy = parser.getText();
                    String [] split = hierarchy.split(">");
                    jGenerator.writeStringField("campus", split[0]);
                    jGenerator.writeStringField("building", split[1]);
                }
            }
        }
    }
}
```

```
        jGenerator.writeStringField("floor", split[2]);
    }
    else if ("timestamp".equals(token)) {
        parser.nextToken();
        long timestamp = parser.getLongValue();
        int dayOfWeek = getDayOfWeek(timestamp);
        int hourOfDay = getHourOfDay(timestamp);
        jGenerator.writeNumberField("timestamp", timestamp);
        jGenerator.writeNumberField("dayOfWeek", dayOfWeek);
        jGenerator.writeNumberField("hourOfDay", hourOfDay);
    }
    else if ("longitude".equals(token)) {
        parser.nextToken();
        double latitude = parser.getDoubleValue();
        jGenerator.writeNumberField("latitude", latitude);
    }
    else if ("latitude".equals(token)) {
        parser.nextToken();
        double longitude = parser.getDoubleValue();
        jGenerator.writeNumberField("longitude", longitude);
    }
    else if ("salt_timestamp".equals(token)) {
        parser.nextToken();
        long salt = parser.getLongValue();
        jGenerator.writeNumberField("salt_timestamp", salt);
    }
    else if ("id".equals(token)) {
        parser.nextToken();
        String id = parser.getText();
        jGenerator.writeStringField("id", id);
    }
    else if ("accuracy".equals(token)) {
        parser.nextToken();
        double accuracy = parser.getDoubleValue();
        jGenerator.writeNumberField("accuracy", accuracy);
    }
}
jGenerator.writeEndObject(); // }
```



```
    }  
    jGenerator.writeEndArray();  
    jGenerator.close();  
}catch (JsonGenerationException e) {  
    e.printStackTrace();  
}catch (JsonMappingException e) {  
    e.printStackTrace();  
}catch (IOException e){  
    e.printStackTrace();  
}  
System.out.println("FINISHED with "+ sourceFile+"!");  
}
```

Appendix C

Email correspondence with AsterixDB developers



Magnus Kongshem <kongshem.entertainment@gmail.com>

Limit clause

Mike Carey <dtabass@gmail.com>

Thu, Apr 7, 2016 at 8:53 PM

Reply-To: users@asterixdb.incubator.apache.org

To: users@asterixdb.incubator.apache.org

Cc: "asterixdb-dev@googlegroups.com" <asterixdb-dev@googlegroups.com>

We also have an apparently undocumented (oops!) option in our HTTP API that serves this case well - it allows you to ask the AsterixDB engine to run the query and return with a handle for the results when done - and then you can go back and use the handle to grab the results. (NOTE: This is different than the completely asynchronous case, which *is* documented, which lets you ask the engine to run the query but return immediately, before the query is done - which was put there for really long queries, where you might want to run them in the background and check on them once every so often.)

On 4/7/16 9:40 AM, Taewoo Kim wrote:

What we usually do is wrapping a query using count() like the following.

```
use dataverse XXX;
count(
from $obj in dataset YYY
  where $obj.timestamp >= 1412121600
  and $obj.timestamp <= 1412726400
  limit 2
return $obj);
```

Best,
Taewoo

On Wed, Apr 6, 2016 at 11:54 PM, Magnus Kongshem <kongshem@stud.ntnu.no> wrote:

Thanks for the detailed reply, I recommend documenting this!

What I'm doing is testing different queries to find their execution time, and if the results from my query contains 12 million objects, the browser is having trouble displaying the entire list(it is chopped of some where along the way). So I tested with the limit clause and got the same execution time (give or take a few seconds). So, is there another way of performing a query to get the execution time without having to print the entire result? Or is limit the best workaround for this use case? Recall my example query below where I use "return \$obj;".

--

Mvh

Magnus Alderslyst Kongshem
+47 415 65 906

On Wed, Apr 6, 2016 at 9:58 PM, Mike Carey <dtabass@gmail.com> wrote:

(And clearly both optimizations are *really* important for performance - both in real use cases and also in benchmarks.)

On 4/6/16 8:37 AM, Taewoo Kim wrote:

Hello Magnus,

Right now, the search does not stop until it finds all tuples that satisfies the given predicate. However, if you have an ORDER BY clause before LIMIT, it can stop sorting process in ORDER BY early. That is, it picks top K (in your example, K is 2) and sorting process stops early.

Actually, if the plan qualifies as an index-only plan (the given predicate can be covered by a secondary index search and you only return secondary key field and/or primary key field), the LIMIT can be applied to an index-search so that an index-search can be stopped earlier after finding K tuples. This feature is already implemented but in the code-review now. It would be applied soon.

Best,
Taewoo

On Wed, Apr 6, 2016 at 7:22 AM, Magnus Kongshem
<kongshem@online.ntnu.no> wrote:

Hey,

Performing a query with the limit clause, does this cause the query to stop searching when it reaches the value of the limit parameter(like in SQL), or does it keep searching for every matching object and then return the number of objects specified by the limit? Example below:

```
use dataverse XXX;
from $obj in dataset YYY
  where $obj.timestamp >= 1412121600
  and $obj.timestamp <= 1412726400
  limit 2
return $obj;
```

Followup questions may occur.

--

Mvh

Magnus Alderslyst Kongshem
[+47 415 65 906](tel:+4741565906)

Appendix D

AsterixDB implementation

D.1 Test iteration 1

```
use dataverse bigd;

create type table as open {
  uid: uuid,
  campus: string,
  building: string,
  floor: string,
  timestamp: int32,
  dayOfWeek: int32,
  hourOfDay: int32,
  latitude: double,
  salt_timestamp: int32,
  longitude: double,
  id: string,
  accuracy: double
}

create dataset posdata(table)
  primary key uid autogenerated;
create index stamp on posdata(timestamp);
create index hour on posdata(hourOfDay);
create index day on posdata(dayOfWeek);
create index id on posdata(id);
```

```
create index acc on posdata(accuracy);
create index building on posdata(building);
create index floor on posdata(floor);
create index stamp_id on posdata(timestamp, id);
create index stamp_hour on posdata(timestamp, hourOfDay);
create index stamp_day on posdata(timestamp, dayOfWeek);
create index stamp_acc on posdata(timestamp, accuracy);
create index stamp_building on posdata(timestamp, building);
create index stamp_hour_day on posdata(timestamp, hourOfDay, dayOfWeek);
create index stamp_hour_building on posdata(timestamp, hourOfDay, building);
```

D.2 Test iteration 2

```
use dataverse bigd;

create type table5 as open {
  uid: string,
  campus: string,
  building: string,
  floor: string,
  timestamp: int32,
  dayOfWeek: int32,
  hourOfDay: int32,
  latitude: double,
  salt_timestamp: int32,
  longitude: double,
  id: string,
  accuracy: double
}

create dataset posdata3(table5)
  primary key uid with filter on timestamp;

create index id on posdata3(id);
create index floor on posdata3(floor);
```

Appendix E

Import more data into the database

```
create dataset posdata_temp(table5) primary key uid;
load dataset posdata_temp using localfs "path="
localhost:///data/path/to/file/file.adm,
localhost:///data/path/to/file/file2.adm,
localhost:///data/path/to/file/file3.adm"),("format"="adm"));

insert into dataset posdata3(
  for $obj in dataset posdata_temp
  return $obj
)
```

Bibliography

- Dirk Ahlers, Kristoffer Gebuhr Aulie, Jeppe Eriksen, and John Krogstie. Visualizing a city within a city: Mapping mobility within a university campus. *Conference on Big Data and Analytics for Smart Cities BigDASC*, 2015. Toronto, 13.10.15.
- Sattam Alsubaiee, Yasser Altowim, Hotham Altwairy, Alexander Behm, Vinayak Borkar, Yingyi BU, Michael Carey, Inci Cetindil, Madhusudan Cheelangi, Khurram Faraaz, Eugenia Gabrielova, Raman Grover, Zachary Heilbron, Young-Seok Kim, Chen Li, Guangqiang LI, Ji Mahn Ok, Nicola Onose, Pouria Pirzadeh, Vassilis Tsotras, Rares Vernica, Jian Wen, and Til Westmann. Asterixdb: A scalable, open source bdms. *University of California, Irvine AND Cloudera Inc AND Google AND IBM AND MarkLogic Corp AND Pivotal Inc. AND University of California, Riverside AND HP Labs AND Oracle Labs*, 2014a. URL <http://sigmod.github.io/papers/p1289-alsubaiee.pdf>. visited on 29.05.16.
- Sattam Alsubaiee, Alexander Behm, Binayak Borkar, Zachary Heilbron, Young-Seok Kim, Michael J. Carey, Markus Dreseler, and Chen Li. Storage management in asterixdb. *PVLDB*, 2014b. URL <http://www.vldb.org/pvldb/vol17/p841-alsubaiee.pdf>. visited on 30.05.16.
- Sattam Alsubaiee, Michael J. Carey, and Chen Li. Lsm-based storage and indexing: An old idea with timely benefits. In *Second International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data*, GeoRich'15, pages 1–6, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3668-0. doi: 10.1145/2786006.2786007. URL <http://doi.acm.org/10.1145/2786006.2786007>.
- Steinar H Andresen, John Krogstie, and Thomas Jelle. Lab and research activities at wireless trondheim. In *Proceedings of IEEE International Symposium on Wireless Communication Systems*, pages 385–389, 2007.
- Rupali Arora and Rinkle Rani Aggarwal. Modeling and querying data in mongodb. *International Journal of Scientific and Engineering Research*, 4(7), 2013.

- Kristoffer Gebuhr Aulie. Human mobility patterns from indoor positioning systems. Master thesis, Norwegian University of Science and Technology, May 2015.
- Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012. ISBN 0321815734, 9780321815736.
- Gergely Biczok, Santiago Diez Martinez, Thomas Jelle, and John Krogstie. Navigating mazemap: indoor human mobility, spatio-logical ties and future potential. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 266–271. IEEE, 2014.
- Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1151–1162. IEEE, 2011.
- Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '09*, pages 233–244, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-606-9. doi: 10.1145/1583991.1584053. URL <http://doi.acm.org/10.1145/1583991.1584053>.
- Rick Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
- Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014. ISSN 1572-8153. doi: 10.1007/s11036-013-0489-0. URL <http://dx.doi.org/10.1007/s11036-013-0489-0>.
- Yongguang Chen and Hisashi Kobayashi. Signal strength based indoor geolocation. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 1, pages 436–439. IEEE, 2002.
- Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.
- Kristoffer Dyrkorn. Hvor ble det av de store dataene? *BEKK OPEN*, 2016. URL <http://open.bekk.no/hvor-ble-det-av-de-store-dataene>. visited on 13.02.16.
- Jeppe Benterud Eriksen. Visualization of crowds from indoor positioning data. Master thesis, Norwegian University of Science and Technology, May 2015.
- Navid Fallah, Ilias Apostolopoulos, Kostas Bekris, and Eelke Folmer. Indoor human navigation systems: A survey. *Interacting with Computers*, page iws010, 2013.
- Alan Hevner and Samir Chatterjee. *Design science research in information systems*. Springer, 2010.

- Jeffrey Hightower and Gaetano Borriello. Location sensing techniques. *IEEE Computer*, 34(8):57–66, 2001.
- Nan Li, Gulben Calis, and Burcin Becerik-Gerber. Measuring and monitoring occupancy with an rfid based system for demand-driven hvac operations. *Automation in construction*, 24:89–99, 2012.
- James Little and Brendan O’Brien. *A Technical Review of Cisco’s Wi-Fi-Based Location Analytics*. Tech. rep., Cisco, 2013.
- Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1067–1080, 2007.
- Phil Simon. *Too Big To Ignore The business case for big data*. John Wiley & Sons, 2013. ISBN 978-1-118-63817-0.
- Michael D Sohn. Occupancy-based energy management in buildings: Final report to sponsors. *Lawrence Berkeley National Laboratory*, 2010.
- Vijay Vaishnavi and William Kuechler. Design science research in information systems. 2004. URL <http://www.desrist.org/design-research-in-information-systems/>. last updated: November 15, 2015. Last visited 06.06.16.
- Colin Ware. *Information visualization: perception for design*. Elsevier, 2012.
- JH Wilkinson. The algebraic eigenvalue problem. In *Handbook for Automatic Computation, Volume II, Linear Algebra*. Springer-Verlag New York, 1971.
- Wen Yao, Chao-Hsien Chu, and Zang Li. The adoption and implementation of rfid technologies in healthcare: a literature review. *Journal of medical systems*, 36(6):3507–3525, 2012.
- Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.