A SCALABLE FRAMEWORK FOR AIS DATA ANALYSIS AND VISUALIZATION

by

Kai Qi

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
June 2016

Table of Contents

List of Figures

Abstract

Maritime traffic data is an important resource to understand vessel activities. Several topics, such as maritime traffic anomaly detection, have been widely studied. With the increase in the number of vessels, location, transponders and satellite receiving statistics, the size of these data is huge and rapidly increasing. A single machine or a sequential algorithm may not be able to handle data at this scale. Due to the lack of scalable tools, many problems on marine data have only been studied on relatively small datasets.

In this thesis, we propose a framework, AIS Data Explorer, to analyse and visualise marine trajectory data. The framework achieves the following goals: scalability, support for big data visualisation and acceleration of large scale data analysis. Users are able to visualise large marine datasets on an interactive map. Client visualisations include heat maps, grid maps and trajectories. Data to be displayed can be filtered by dimensions including area of interest, time period, ship type and ship status. Moreover, it allows user to implement algorithms and analyse data of interest. We designed and implemented a web application for data visualisation and explored the use of vOLAP and Spark platform for data processing. Velocity OLAP (vOLAP) is a scalable, real-time OLAP system designed for high velocity data. It provides efficient query for data aggregations. To support the desired visualisations, a module was implemented to enable vOLAP in order to answer histogram query. Spark is a cluster computing framework designed for big data. It allows user to keep data in cluster memory and query it repeatedly. With Spark as the core, we designed and implemented a pipeline for big data analysis of vessel trajectories. It performs data cleaning, data processing and data analysis. As an example of large scale data analysis, we designed and implemented a distributed DBSCANSD method that discovers vessel traffic patterns from trajectory data. In our experiments, using 5 worker instances for a dataset size of 100 million items, the framework visualised a global heat map in 20 seconds.

List of Abbreviations and Symbols Used

AIS	Automatic Identification System
BSP	Binary Space Partition
CB-SMoT	Clustering based Stops and Moves of Trajectories
COG	Course over Ground
DB-SMOT	Direction Based Stops and Moves of Trajectories
DBSCAN	density based spatial clustering of applications with noise
DBSCANSD	Density based Spatial Clustering of Applications with Noise considering Speed and Direction
GPS	Global Positioning System
HDFS	Hadoop Distributed File System
IMO	International Maritime Organisation
MBR	Minimum Bounding Rectangle
MDS	Minimum Describing Set
MMSI	Maritime Mobile Service Identity
OLAP	Online Analytical Processing System
OLTP	Online Transaction Processing
RDD	Resilient Distributed Dataset
SOG	Speed over Ground

SVM Support Vector Machine

vOLAP Velocity Online Analytical Processing System

Acknowledgements

First, I would like to express my gratitude to my supervisor Dr. Andrew Rau-chaplin for his patience, motivation, encouragement and advice. I cannot imagine a better supervisor and mentor for my research and study. This thesis would not have been possible without his help and guidance.

Besides my supervisor, many thanks are given to my thesis committee: Dr. Qigang Gao, Dr. Evangelos E. Milios and Dr. Christian Blouin, for their encouragement, suggestions and questions. I would also like to express my gratitude to Casey Hilliard, Neil Burke, Rahul Mahadev and Nitin Agrawal for their discussions during our project meetings.

At last, I would like to thank my parents for their support and love.

Chapter 1

Introduction

More than 90 percent of current global trades take place on the ocean [18]. In order to effectively monitor maritime activities and avoid maritime accidents, industries, governments and marine organisations need to understand the movement of ships on the ocean and be able to analyse marine data. This is where the Automatic Identification System (AIS) [18, 47, 35, 33, 14] comes into use. It is designed for tracking vessel activities. Several methods for analysis on AIS data have been developed for applications including traffic route extraction [35], vessel route prediction [47] and anomaly detection [45, 47].

Automatic Identification System (AIS) is a communication system based on radio frequency [18]. Information is exchanged between nearby ships, terrestrial base stations and satellites. The satellite-based system can receive messages from open sea which is far away from coastline. Space-based receivers are mounted on Low Earth Orbit satellites with sufficient satellite coverage, AIS can capture a global view of maritime activities [47]. Captains usually use AIS to ensure the safety of the ships, for example, it can be used as a basic method of collision avoidance. For maritime authorities, they use AIS to track vessels and figure out their locations, directions and destinations. In order to enhance safety and efficiency of navigation, the International Maritime Organisation (IMO) requires AIS transponders to be installed on vessels and more than 60,000 ships have installed this system [18].

AIS messages contain two categories of data: vessel static and dynamic information. Static information includes ship name, ship type, vessel dimensions, vessel call sign, IMO number and Maritime Mobile Service Identity (MMSI). Sometimes, it contains

marine navigation data including vessel destination and estimated time of arrival. Dynamic information includes ship position (latitude and longitude), UTC time, navigation status, Rate of Turn, Speed over Ground (SOG) and Course over Ground (COG). Depending on their speed, vessels broadcast their dynamic information at a variable rate and send their static information every five minutes [47].

AIS data is an important resource for studying maritime activities. One crucial task in marine data analysis is ship trajectory clustering. It is a process to discover vessel traffic pattern from trajectory data. One of the representative clustering methods is density-based spatial clustering of applications with noise (DBSCAN) [30]. Several DBSCAN based clustering algorithms have been proposed. Rocha et al. [37] proposed a clustering method, called DB-SMoT (Direction Based Stops and Moves of Trajectories). It considers the variation of the direction. The proposed method is used to find real places where vessels develop fishing activities. Another DBSCAN based method, called CB-SMoT (Clustering-based Stops and Moves of Trajectories), is proposed in [36]. Different from DB-SMoT, CB-SMoT takes speed into account.

Anomaly detection is another kind of application, which can be used in ocean protection, intrusion detection and military surveillance. It detects vessels that do not conform to vessel traffic patterns [45]. Pallota et al. [47] proposed an online method for anomaly detection. The anomalies can be measured by deviation from the normality learned from the historical data. Gerben et al. [26] proposed a method based on machine learning. Different trajectory alignment kernels are applied with SVMs (Support Vector Machine) for outlying trajectory detection.

1.1 Aim and Objectives

Automatic Identification System (AIS) provides a huge amount of maritime navigation data. It helps researchers better understand maritime activities. Several topics, such as anomaly detection [45, 47], have been widely studied with AIS data. However, most studies work on relatively small datasets. Apparently, the results from large datasets are more general and reliable than those from smaller ones. The whole

AIS dataset is huge and it increases rapidly. Taking Big Data Institute at Dalhousie University as an example, it has accumulated nearly 4 terabytes maritime navigation data within five years from 2010 to 2014. The size keeps increasing incrementally by 100 gigabytes per month. To handle data at this scale, a single machine or a sequential algorithm may be too slow and simply do not have the processing power. In this thesis, we propose a scalable framework, called AIS Data Explorer that is designed to analyse and visualise maritime navigation data at a large scale. It has been designed to solve the following two problems:

1) how to explore AIS dataset and find out interesting data.

2) how to perform data analysis on a large dataset.

1.2 Contributions

This thesis presented the design, implementation and evaluation of a data analysis framework for AIS data called AIS Data Explorer. The framework provides an intuitive way to explore the AIS dataset so that users can find the data of interest easily. The AIS dataset is massive and increasing rapidly. Thus, the framework was designed to be scalable and able to do fast queries on huge datasets. Moreover, new features can be integrated into the existing framework easily, which allows users to do their desired data analysis. To achieve these goals, AIS Data Explorer is designed to be a cloud-based system. Users are able to visualise large amount of data on an interactive map. Client visualisation includes heat maps, grid maps and trajectories. Data to be displayed can be filtered by many dimensions including area of interest, time period, ship type and ship status. Depending on data size, results can be typically visualised within seconds. Furthermore, the framework allows users to implement their own algorithms and run them on data of interest. The computation of the algorithm can be distributed over the whole computing cluster. Also in this thesis, we designed and implemented a distributed version of DBSCANSD [35] that discovers vessel traffic patterns from trajectory data.

Figure 1.1 presents the system architecture of AIS Data Explorer. It consists of two

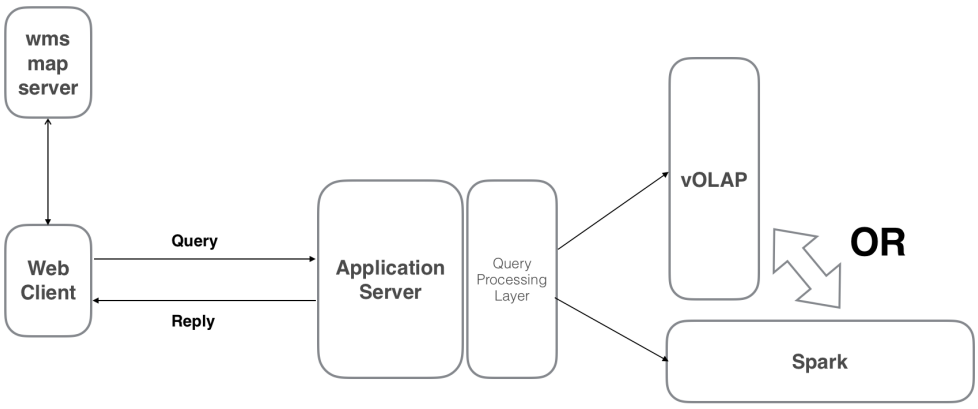

Figure 1.1: System Architecture

main components: a front-end system for user interaction and data visualisation, a distributed framework for marine data processing. The front-end system is made of a web client and a web server. The web client handles user input and translates it into the query. When the reply is received, the web client parses it and visualises the data on the web map. The web server works as a connector between the client and the back-end system. It also provides basic functions like user verification, web application initialisation. For data processing, we explored two types of distributed frameworks: vOLAP and Spark. Velocity OLAP (vOLAP) [28] is a cloud-based real-time OLAP system designed for multi-dimensional data. It can answer aggregate queries efficiently. Spark [38] is a cluster computing framework designed for big data processing. It allows user to implement their own algorithms which can be executed over a cluster.

1.3 Thesis Outline

The remainder of this thesis is organised into five chapters. Chapter 2 describes the technologies used in this thesis. It gives a brief introduction to web development techniques, online analytical processing system (OLAP) and the cluster computing framework (Spark). In Chapter 3, we present the design of our front-end system. To start with, we show the examples of data visualisations and then introduce the used techniques. Then we give an overview of the system architecture. The exploration of vOLAP based query processing is described in Chapter 4. vOLAP provides an

efficient way to query data aggregations. In Chapter 5, we explore AIS data processing using Spark. It allows user to perform complex data analysis on a large dataset. In Chapter 6, we present the results of an experimental evaluation of the AIS Data Explorer. The conclusion of this thesis is in Chapter 7, which summarises the goals we achieved and gives ideas for future work.

Chapter 2

Background

In this chapter, we introduce the technologies used in this thesis. To start with, we first give an overview of Online Analytical Processing System (OLAP) and introduce data structures for OLAP and Velocity OLAP (vOLAP) [28]. vOLAP provides a method to answer real-time aggregate queries efficiently. In Section 2.2, we introduce a cluster computing framework called Spark. It provides a scalable and flexible way to process big data. At last, we give an overview of used web techniques.

2.1 Online Analytical Processing System

Online analytical processing system (OLAP) is at the heart of business intelligence. It is the foundation for many essential business applications including business report for sales, marketing analysis, budgeting and forecasting, performance measurement [44].

OLAP systems allow users to analyse multi-dimensional data from different perspectives. It supports three basic operations: roll up, drill down, slicing and dicing [22]. The aggregation of data can be computed in one or multiple dimensions (roll up). For example, it can compute the number of fishing ships in one area. Drill down is a reverse operation of roll up, which navigates from less detailed data to more detailed data. Given a data cube [32], slicing and dicing generates a sub-cube by doing selections on one or multiple dimensions.

For online transaction processing (OLTP), queries usually access a small portion of the database (e.g. add one new record) [44]. However, queries for OLAP systems are quite different and always need to aggregate a large portion of the whole dataset to support high-level data analysis. Such queries are always computationally expensive.

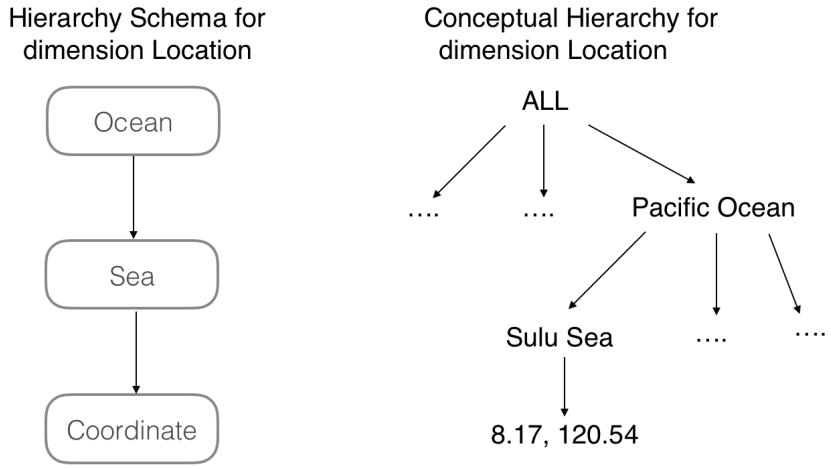

Figure 2.1: Hierarchy Schema and Concept Hierarchy for location

To solve this problem, a static data cube approach is proposed by Jim Gray [32], which materialises a subset of the cuboids of the data cube to improve the query performance. Based on the concept of data cube, many OLAP systems are developed.

However, the traditional OLAP systems can only be updated periodically. In other words, it cannot take the latest information into account and this greatly limits its application. To solve this problem, some real-time OLAP systems are developed. Frank Dehne et al. [28] develop a real-time OLAP system called vOLAP. It is designed for high velocity data. vOLAP system supports dimension hierarchies and exploits both multicore and multiprocessor parallelism. SAP HANA [31] is a cloud-based, in-memory database system designed for business analytics. It works with Hadoop and Spark which supports real-time decision making. Druid is an open-source data store designed for stream data. It supports OLAP queries and is always used to power business analytic applications.

2.1.1 Dimension Hierarchy

A data cube contains two types of attributes : functional attributes which are grouped into dimensions and independent attributes which are called measures [42]. Dimensions are used to identify a subset of a multi-dimensional dataset. If one dimension has multiple attributes, these attributes are organised by a hierarchy schema. Figure

2.1 gives an example, oceans, seas and coordinates make up a dimension for location.

2.1.2 Star Schema

In online transaction processing (OLTP) system, data schema is always highly normalised and its data is time-variant and frequently updated. These systems have many users and are optimised for insertions, updates and deletions. However, OLAP systems have fewer users and are highly optimised for queries [13]. Its data schema is denormalised and has fewer tables, fewer join paths. A data record is always a snapshot and seldom updated. OLAP system usually uses star schemas to represent multi-dimensional data models.

Star schema is a simple style of data schema. It consists of a single fact table and one table for each dimension. The fact table contains measurements for a specific event. It usually contains numeric measurements, such as sales price, sales quantity, distance, and weight. It records events at an atomic level and as a result, fact table usually contains a large number of records. The dimension table has a small number of records compared to the fact table. And each record contains many attributes to describe the fact data. Fact tables change frequently and dimensions do not change, or change slowly over time. Figure 2.2 shows an example for star schema.

Star schemas are highly denormalised. Since it has a small number of tables and few join paths, queries can be processed efficiently. It also accelerates the speed of data loading. The main disadvantage of the star schema is that data integrity is not enforced. Insert operations may result in data anomalies which normalised schema is designed to avoid. And it is not as flexible in terms of analytical needs as a normalised data model.

Some OLAP systems also use snowflake schema to represent data models. Snowflake schema is similar to star schema. It optimises the performance of queries and is easy to understand. But in a snowflake schema, dimensions are normalised into multiple related tables. Compared to star schema, it has more join paths and supports more complex queries. It saves some storage space but the performance is slower.

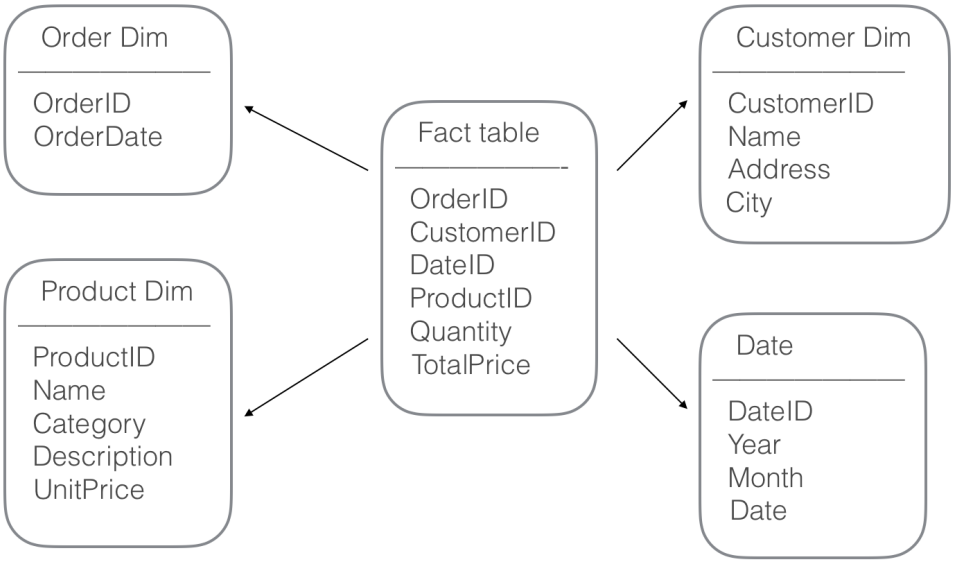

Figure 2.2: Star Schema

2.1.3 Data Structure for OLAP

In this section, we discuss data structures used in OLAP systems. A OLAP system usually supports point data insertion, bulk data insertion, point data query and multi-dimensional query. It does not support delete operation and keeps a historical append-only dataset. In a OLAP system, each data record is a multi-dimensional data point with dimension hierarchies. There are several tree-based data structures used to implement OLAP systems. Here we give a brief overview of these data structures.

R-Tree In order to handle spatial data efficiently, a dynamic tree-based data structure, called R-Tree, has been proposed by Antonin Guttman [16] in 1984. It has significant use in both theoretical and applied contexts. A common usage for R-tree is to index multi-dimensional data like geographical coordinates, rectangles or polygons. It can also accelerate the process of nearest neighbour search, which is useful in many algorithms. R-tree has many variants including R+ tree [46], R* tree [27] and X-tree [15]. The basic idea of R-tree is to use minimum bounding rectangle (MBR) to group nearby data objects. Figure 2.3 shows an example of R-tree for 2d rectangle. MBRs in root node covers all data points in the tree. MBRs in each internal node

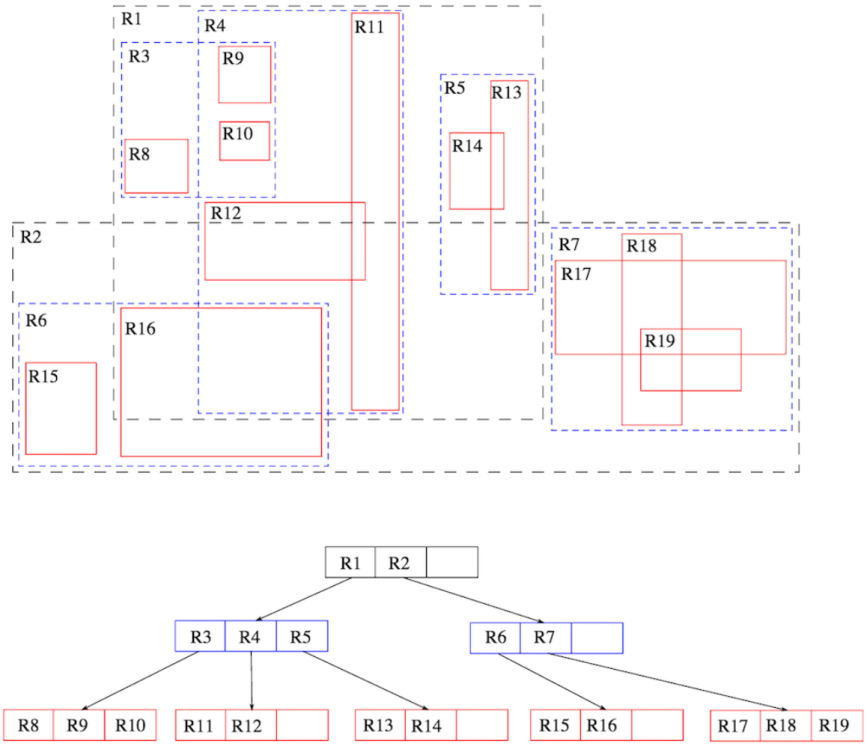

Figure 2.3: An Example for 2D R-Tree [23]

covers all data points in its subtree.

R-tree is a height-balanced tree data structure that consists of directory nodes and leaf nodes. Each directory node contains MBRs and a list of pointers to its children. Since each directory node can have more than two children, the height of R-tree can be very small. As a result, search and insert operations only need to visit a small number of internal nodes. Each leaf node contains links to the actual data that can be stored on the disk. The search process in R-tree is a top-down process which starts from the root node. For each directory node, it checks if its MBR and query box overlap. If the overlap exists, it checks the children of this directory node recursively. If there is no overlap, it skips this node and checks other directory nodes at the same level. At last, the process stops at leaf nodes and then retrieves the data.

Minimisation of overlap and coverage is important to the performance of R-tree. Overlap is the area contained in one or two nodes. Minimal overlap reduces the number of search paths to leaf nodes. Coverage is the area which covers all related bounding

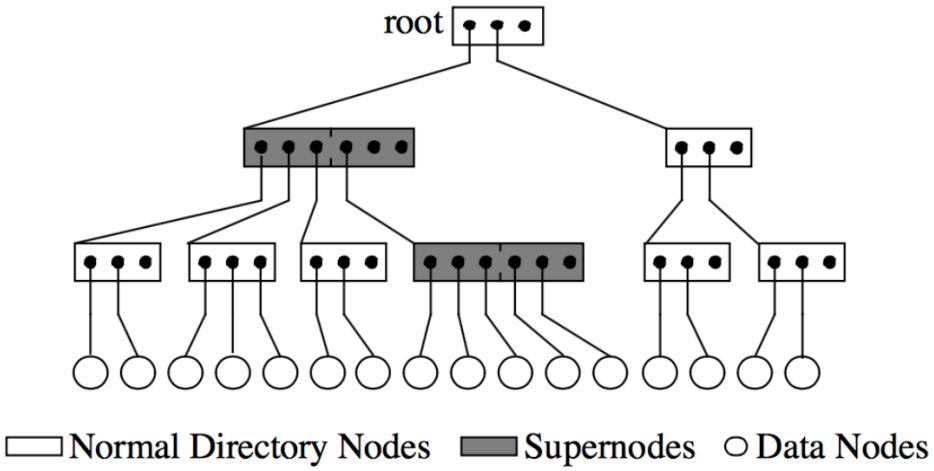

Figure 2.4: An Example of X-Tree [15]

rectangles. Minimal coverage reduces the empty area covered by the nodes of R-tree. Compared to R-tree, R+ tree [46] avoids the overlapping of internal nodes by inserting objects into leaf nodes. It greatly improves query performance but requires more time to construct. R* tree [27] tries to minimise both overlap and coverage by using an improved node split algorithm. It can support point data and multi-dimensional data at the same time.

X-Tree An analysis survey shows that R-tree based index data structure cannot handle high dimensional datasets efficiently. The experiments [15] show that the performance of R*-tree degrades rapidly when the dataset dimensions increase. And it also shows that the overlap of bounding box (MBR) in the directory nodes increases dramatically when the dimensions increase. As we mentioned before, when the overlap increases, the number of search paths increases resulting in slowing down the query performance. To solve this problem, a data structure, called X-tree, was proposed by Stefan Berchtold et al. [15] in 1996. For high dimensional data, it outperforms R*-tree by up to two orders of magnitude in their experiments. Figure 2.4 gives an example of X-tree.

The basic idea of X-tree is to minimise the overlap utilising the concept of supernode. It tries to avoid overlap whenever it is possible. If overlap is inevitable, X-tree uses

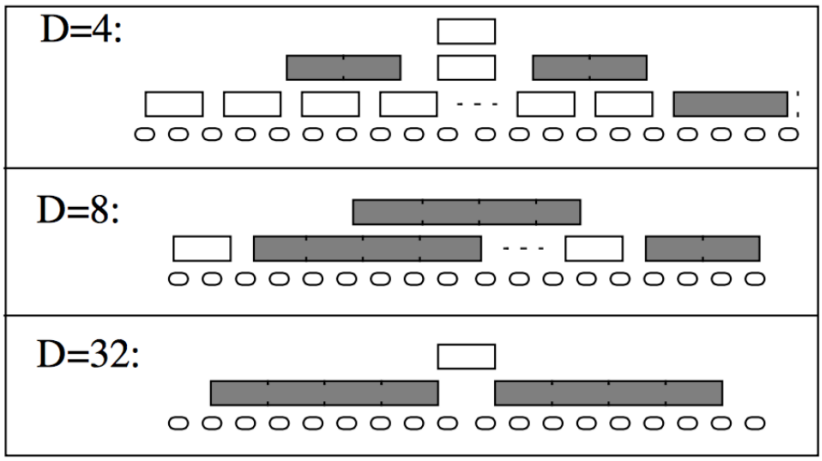

Figure 2.5: Various shapes of X-tree in different dimensions [15]

an extended directory node, which is also called supernode. X-tree has two types of directory nodes: linear array-like directory nodes and R-tree-like directory nodes. In low dimensions, the X-tree has less overlap and less search paths for queries. The number of required access to tree node corresponds to the height of tree. Therefore, R-tree-like organisation is efficient. In high dimensions, X-tree has more overlap and more search paths for queries. Since most of directory nodes need to be searched, a linear scan to the directory nodes will be faster. Therefore, a linear array-like organisation is efficient. In different dimensions, X-tree has different structures to achieve the best performance. Figure 2.5 shows different structures of X-tree under different dimensions.

DC-Tree Typical OLAP systems can be only updated periodically. In other words, data in the system is not up to date. To solve this problem, a dynamic index structure, called DC-tree, is introduced [42]. It exploits the concept hierarchy that is a tree-based data structure which contains all appeared values in one given dimension [42]. DC-tree uses minimum describing set (MDS) instead of minimum bounding rectangle (MBR). MDS is used to describe a bounding box which covers the data stored in the corresponding subtree. Compared to MBR, MDS covers less empty space but it has to store some additional information. A data record consists of dimensions and measures. DC-tree assigns an id to each dimension of the data record. MDS actually

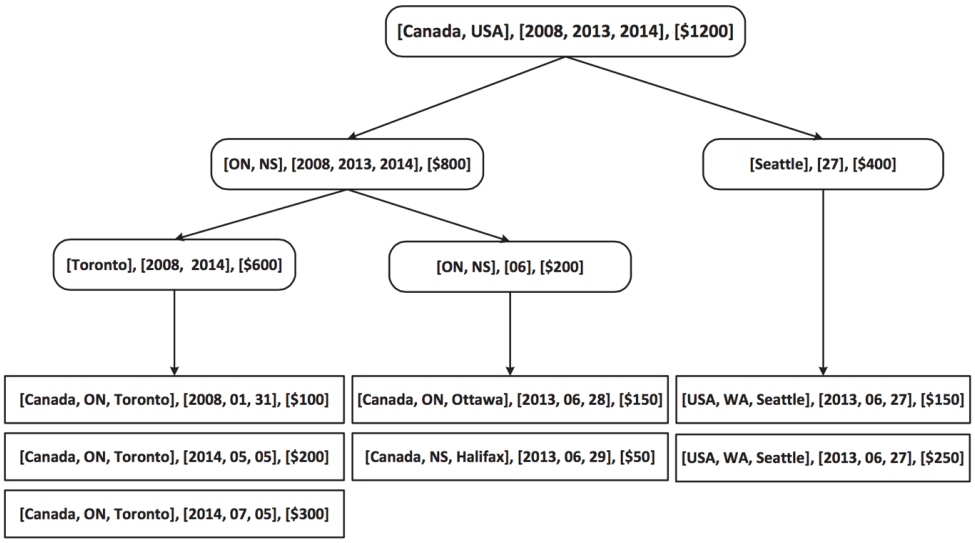

Figure 2.6: Example of DC-Tree [44]

consists of a list of IDs and each ID represents the value at a level of one dimensional hierarchy.

Figure 2.6 shows an example of two dimensional DC-tree. DC-tree consists of leaf node, directory node and supernode. One leaf node contains a MDS and measures. One directory node contains a MDS that covers the data in its subtree, aggregation values of the data in its subtree and links to its children. For example, MDS ([Europe, 2013 06]) contains ([France, 2013 06]) and ([England, 2013 06]). France and England belong to Europe. Search in DC-tree is a top-down process started from the root node. The query is a bounding box like MDS. For each directory node, DC-tree checks if the query and its MDS overlap. If the overlap exists, it checks the children of this directory node recursively. If the overlap is empty, it skips this node and checks the remaining directory nodes. If the MDS of this directory is fully contained in the range of query, aggregation values stored in this directory node are added to the result without visiting nodes in its subtree. This is the reason why DC-tree can answer aggregate query efficiently.

2.1.4 vOLAP

Velocity OLAP (vOLAP) is a scalable real-time OLAP system which works in the cloud environment. It supports data ingestion, OLAP queries but no deletion. Since vOLAP keeps the data in memory, it is able to process OLAP queries efficiently. The system is optimised for stream data and leverages PDC-Tree as a basic building block. It consists of multiple servers and workers. Data is distributed over multiple workers. Each user session is attached to one of servers and this server handles all queries from this client. It redirects these queries to appropriate workers, collects and computes the result and then sends it back to the original client. Since vOLAP is designed for working in the cloud environment. Servers and workers can be added or removed dynamically, which abbreviates the performance bottleneck for large amounts of data. Zookeeper [12] is used to manage global information. And a dynamic load balancing strategy is designed. A manager process monitors the load status of the whole system and sends instructions to workers for global load balancing. Because of the incoming data, the data distribution in vOLAP changes significantly overtime. The experiments conducted in [44] show, using 18 workers and a dataset with 1.5 billion records, vOLAP is able to process streams of inserts and queries at a rate of 200,000 per second.

Compared to vOLAP, other real-time OLAP systems, such as SAP HANA [31] and Druid, still have their own drawbacks. SAP HANA is a distributed, in-memory, data store designed for modern business applications, which consists of multiple data processing and query engines. The problem of SAP HANA is that it only has one active master node. It limits the scalability of the system and may become a bottleneck as client queries increase. vOLAP has multiple servers to handle queries and servers can be added or removed dynamically, which abbreviates this performance bottleneck. Druid is another distributed data store optimised for real-time streaming data. Although Druid supports OLAP queries, it is still quite different from standard OLAP systems. It does not support dimensional hierarchy which is supported in most OLAP systems. vOLAP does not this problem and can handle stream data efficiently.

Figure 2.7 shows an overview of the architecture of vOLAP. It consists of a set of

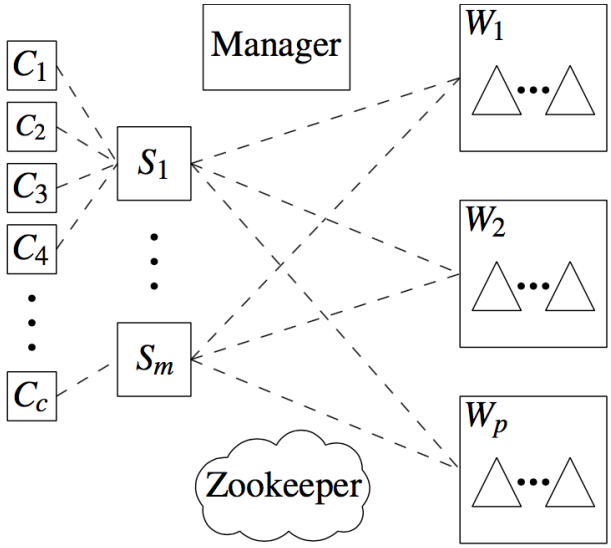

Figure 2.7: System Architecture of vOLAP [28]

servers and workers. Servers are used to handle client queries and workers are used to store data and process OLAP queries. The dataset is partitioned into multiple subsets and each subset is covered by a bounding box (MBR or MDS). One single worker may contain multiple subsets. Zookeeper maintains a global system image and each server keeps a local system image. When one server receives an OLAP query from one client, it checks its local system image and decides which worker the query should go. When one worker receives an OLAP query from the server, it searches in local PDC-tree and then sends the result back to the server. For one OLAP query, one server generates multiple subqueries, then collects data from different workers and at last computes the final result. The whole process is highly parallelised.

vOLAP uses PDC-Tree as a basic building block. Each worker stores one or more subtrees and each server stores the tree hat for routing. When one worker receives a query from a server, it searches in its local PDC-Tree. This is a top-down process. If one directory nodes bounding box and query box overlap, it checks this directory nodes children recursively. And at last, it stops at the leaf node which contains the required data. Three data structures are implemented for query processing: an array for benchmarking, a PDC-MBR tree and a PDC-MDS tree. PDC-MBR tree is

optimised for insert operation. PDC-MDS tree uses cache-efficient MDS implementation and has better performance when tree becomes large. vOLAP connects its components including servers, workers and the manager process, using ZeroMQ [11]. ZeroMQ is a high-performance asynchronous messaging library designed for building distributed applications. It provides APIs for in-process and TCP transport and messaging patterns like request-reply.

The global system state is stored in Zookeeper [44] that provides reliable distributed coordination. With information stored in Zookeeper, the manager process performs load balancing among workers. Each server contains a local system state used for insert and OLAP query operations. Insertion may change the local system state and servers need to update information in Zookeeper. In vOLAP, servers are configured to update Zookeeper every 3 seconds [28]. If the stored global system state changes, Zookeeper would inform servers of this change and then servers update their local information.

2.2 Cluster Computing Framework : Spark

Apache Spark [38] is an open-source cluster computing framework designed for big data. It has been originally developed in AMPLab at University of California, Berkeley and then became an open-source projects supported by Apache Software Foundation. Until now, it has more than 1000 contributors, which makes it one of the most active project for big data analytics. Spark works with a distributed storage system. It supports a wide variety of distributed frameworks including Hadoop Distributed File System (HDFS) [25], Amazon S3, and Cassandra [17], which makes it flexible to be integrated into existing systems. Different from Hadoop's disk-based MapReduce paradigm, Spark uses a multi-stage in-memory paradigm [20]. It allows user to load data into cluster memory and query it repeatedly, which greatly improves the performance of iterative algorithms. Spark also has a set of high-level tools including Spark SQL for structured data processing, MLib for machine learning, GraphX for graph processing and Spark Streaming for real-time data analysis.

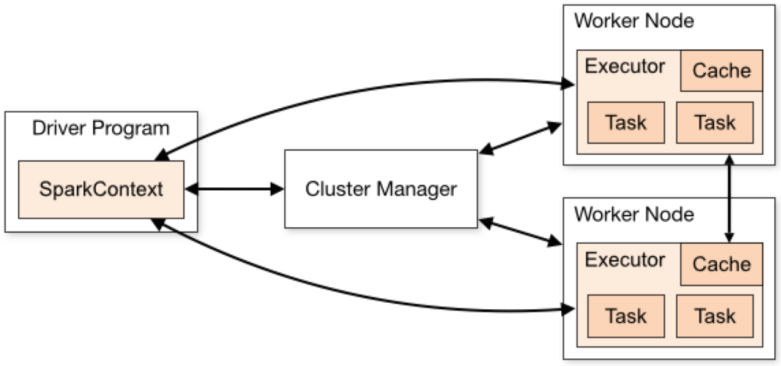

Figure 2.8: Spark Architecture [9]

2.2.1 Spark Architecture

Figure 2.8 shows the architecture of Spark. To start with, any Spark process in the cluster is actually a JVM process. To run a cluster, the driver program connects to the cluster manager and requests some resources. The cluster manager brings up sets of executors which are JVM processes that run computations and store data for user application. Each executor can be viewed as a pool of task execution slots. Each task is a single unit of work executed as a thread in one executor. For example, in a cluster with 6 nodes, if user starts two executors on each node and each executor uses 6 cpu cores. In total the cluster has 72 task slots, which means this Spark cluster would be able to have 72 tasks running at the same time. When user starts a job on the cluster, the processing of a job is split into stages and each stage is split into tasks. The driver program first sends user's application code and then sends tasks to the executors to run.

2.2.2 Resilient Distributed Datasets

One of the main abstractions in Spark is Resilient Distributed Dataset(RDD) [39]. It provides a restricted form of shared memory which is based on coarse-grained transformations. Data reuse is common in iterative algorithms. However, in many distributed frameworks like Hadoop, the only way to reuse intermediate result between computations is to write it back to disk, which greatly slows down the performance. To

solve this problem, RDD allows user to keep intermediate result in memory to avoid overhead caused by data replication and disk I/O.

An resilient distributed dataset (RDD) is a read-only collection of data records. It supports two types of operations: transformations, which create a new RDD from data in stable storage or other RDDs; actions, which return a value after running a computation on an RDD. For example, map is a transformation which applies a function on each data record in a RDD and then returns a new RDD. All transformations are lazy which means Spark remembers transformations applied to the dataset instead of computing the result immediately. Transformations are only performed until a result is required by the driver application. User can specify which RDD will be reused and choose a storage strategy for it (e.g. keep it in memory). Each RDD contains information about how it derives from other RDDs or dataset. If one compute node fails, the lost partitions of an RDD can be recomputed efficiently.

2.2.3 Hive on Spark

Hive on Spark (Shark) [29] is a data analysis system which provides deep data analysis by using the RDD memory abstraction. It combines the SQL query processing engine with analytical algorithms. Compared with traditional data warehouse and distributed framework like Apache Hive, Shark is good at answering ad-hoc queries. By caching required dataset in the cluster memory, queries that take minutes can be reduced to seconds. This improvement is achieved by avoiding overhead due to Disk I/O. Figure 2.9 shows the architecture of Shark.

Shark allows users to manipulate structured data using SQL. Given a Hive query, HiveQL parser is used to parse this query and generate a syntax tree. This tree is turned into a operator tree and eventually turned into transformations which will be performed on RDDs. It also allows users to define their own functions and implement algorithms for special data mining tasks. According to the experiments in [29], Shark is 40X times faster than Apache Hive for query processing and for machine learning algorithms, Shark is nearly 25X faster than MapReduce-based program in Hadoop.

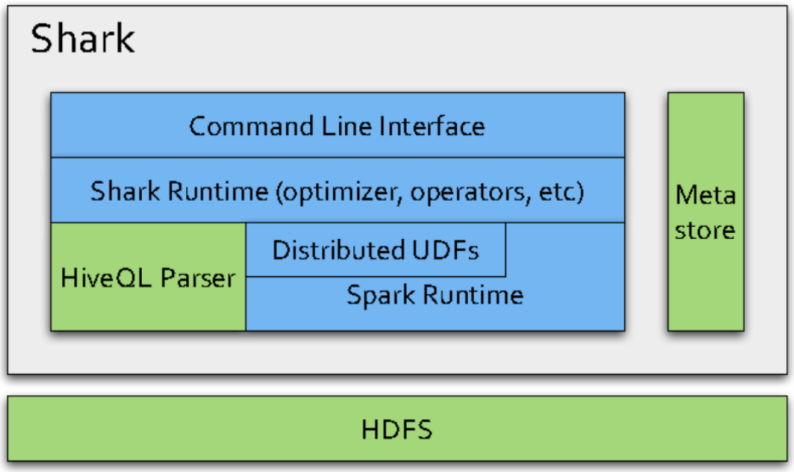

Figure 2.9: Shark Architecture [29]

Chapter 3

AIS Data Explorer : Visualisation Client

In this chapter, we introduce the design and implementation of the visualisation client. To start with, we give an overview of its functionalities, user interface and supported visualisations. In Section 3.2, the techniques used in data visualisations are introduced. Section 3.3 shows the architecture of the visualisation client. A conclusion is lastly given in Section 3.4.

3.1 Introduction

The visualisation client provides an intuitive way to explore the AIS dataset. Users are able to visualise data of interest on an interactive map. It supports three types of visualisations: heat maps, grid maps and trajectories. Data to be displayed can be filtered by dimensions including area of interest, time period, ship type and ship status. In the design and implementation of the visualization client, we encountered into two challenges. The first one is to support different visualizations on the map. We explored web based technologies and proposed a framework for map visualisation. The second challenge comes from the performance of map rendering. The existing methods cannot render the map efficiently. To solve this problem, we implemented a rendering library to support efficient grid map visualisation.

*The user interface, which consists of a web map and a function bar, is shown in Figure 3.1. User can explore area of interest easily using a web map. Function bar, which is on the left side, allows user to select data of interest and specify the type of visualisation. It also supports different types of web maps including satellite map, street map and a customised map with protected areas.

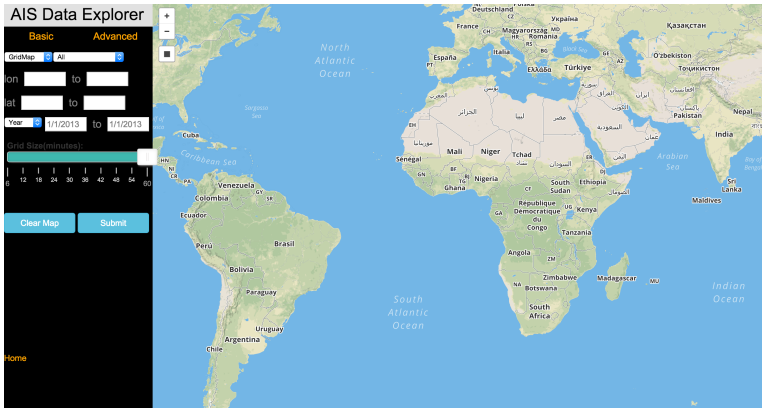

Figure 3.1: User Interface of AIS Data Explorer

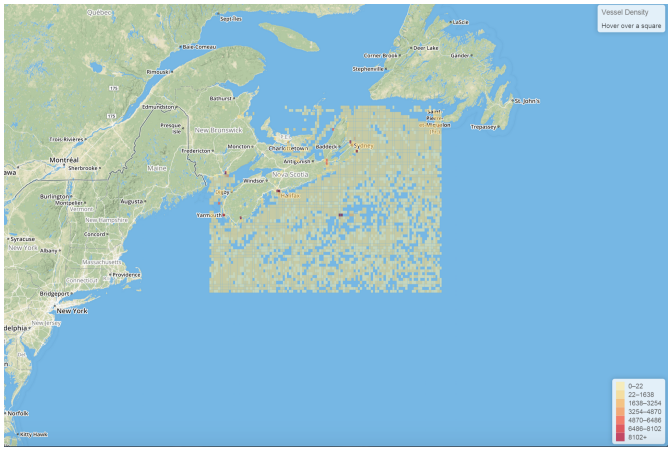

Figure 3.2: Grid Map

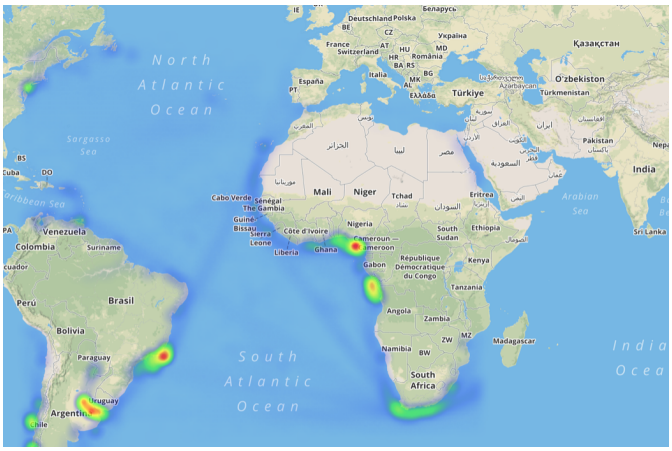

Figure 3.3: Heat Map

Grid map is a graphical representation of data where each individual value in the dataset is represented as colours. Its interface consists of three components: grids, a legend and a label for vessel density. The colour of each cell in the grids is from light yellow to dark red, which correspond the vessel density from low to high. The legend shows the range of vessel density for different colours and it may change according to the data distribution. The label shows the number of vessels in the cell pointed by the cursor. Figure 3.2 gives an example of grid map visualization. It shows vessel density near Nova Scotia, dated from January, 2013 to March, 2013. Heat map is another way to show vessel density in one area. Figure 3.3 shows the example. When the map is zoomed in or out, it will redraw the heat map according to the vessel density of the area. Grid and heat map accept a list of data records as input, each of which contains coordinates that represent a rectangle area on the map and the aggregation of the marine data in that area. The input data can be seen as the result of a two dimensional histogram query. The aggregations of the data are computed in latitude and longitude dimensions.

Figure 3.4 shows vessel trajectories in the first week of January, 2013, near Nova Scotia. Each line represents one vessel sailing trajectory. It also shows the specific channel the vessel sails. Different from heat map and grid map, the input data of trajectory visualisation is a list of vessel trajectories. One trajectory is identified by a unique identifier and contains a list of vessel snapshots, each of which describes its status at one specific instant. The input data can be obtained by a report query

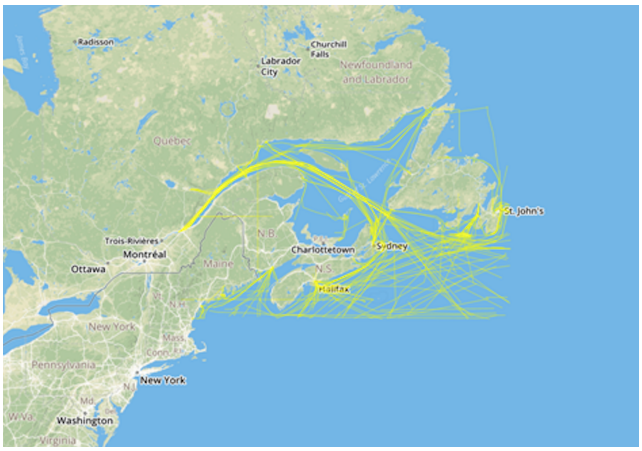

Figure 3.4: Vessel Trajectories

which returns items that satisfy the requirements.

3.2 Techniques for Data Visualisation

Traditional geographical information system, Quantum GIS [7], provides functions to view, edit and analyse geographical data on a desktop. It supports a wide range of data formats including PostGIS layers, OpenStreetMap vectors and Comma Separate data (CSV). However, it cannot process a large amount of data due to the limited processing power of one single machine and is difficult to extend new types of visualizations like grid map. In this section, we explore web mapping and web visualization technologies and then present their applications in the visualization client. These web-based techniques provide a flexible way to analyse and visualise geographical data. Features, like grid map visualization, can be implemented and integrated into the existing framework.

One of the main components of the visualisation client is web map which enabling users to navigate within a single continuous system. For example, by spanning across the map, Toronto, Ottawa and Halifax can be viewed in the same continuous space. By zooming in and out, the amount of details increase from country labels to city streets and buildings. A web map is made of multiple layers, each of which can be images, lines or markers and can be placed on top of each other to build combinations

and mashups. The map layer has abilities to detect click and respond to different mouse events. By defining the corresponding functions, a customised interactive map is created. In the visaulization client, a web map is designed to have two layers: base and feature layer. The base layer is a customised map provided by geographic data provider Mapbox [3]. The feature layer is a visualization layer created by rendering libraries. By combining base and feature layer, users are able to visualise data on a web map. In one user session, the base map is loaded and initialised one time while the feature layer is created and destroyed frequently due to different requests from the user. Thus, the performance of map rendering becomes crucial to user experience.

Several techniques are used to construct the visualisation client. To start with, we introduce Leaflet [2] which is an open-source, client-side only Javascript library designed for building web mapping applications. It provides a basic structure for web map and core methods for manipulating geographical data. Leaflet plays an important role in marine data processing, web map visualisation. A GIS format, called GeoJSON [21], is used to encode marine data. GeoJSON is an open standard format designed for encoding geographic data structures. It is widely supported by mapping and GIS software packages including Leaflet, PostGIS [5] and Mapnik [4]. Several Javascript libraries are used to support data visualisations. Heatmap.js [1], which is a light-weight Javascript library, is used to visualise data in heat maps. It can render thousands of data points in seconds. Grid map is another way to visualise vessel density, which allows user to interact with map and get more information of the data. The creation of a grid map is complicated and the existing methods cannot render it efficiently. Thus, we design and implement a rendering library for grid maps, which provides much better performance.

3.2.1 The Design and Implementation of Gridmap.js

The grid map visualisation has been described in Section 3.1 and here we discuss the design and implementation of the rendering library. The feature layer of grid map can be considered as a set of polygons on a 2D canvas, each of which represents an area of the base map and has features like vessel density. Leaflet has methods to draw

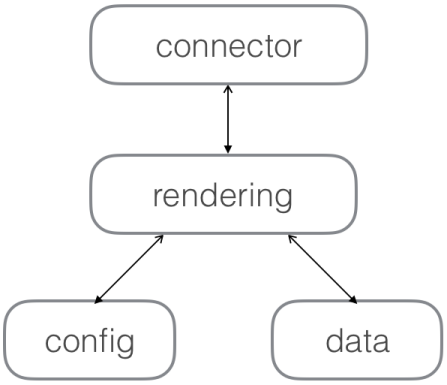

Figure 3.5: The Structure of Gridmap.js

polygons on a map while it is too slow to render thousands of polygons. Instead, we create the feature layer using D3.js and the polygons are represented as SVG [24] elements. The first step of creating the feature layer is to convert geographic coordinates of the data to the corresponding coordinates in the canvas. The rectangles are constructed from the data and then rendered onto the canvas with different colours. For example, if the feature of one rectangle is the largest, it will be filled with dark red to indicate high vessel density in this area. As an interactive map, it will present related information when the mouse moves over one area. To achieve this, a hash table, which maps coordinates to the features of rectangles, is generated. The map layer will detect the coordinates of the mouse, retrieve related data from hash table and then present the data to the user.

Figure 3.5 shows the architecture of Gridmap.js. It consists of four components: a class for configuration, a class for data preprocessing, a class for rendering and a connector which creates feature layers for Leaflet. After the user submits a query, the visualisation client receives the data and then passes it to the rendering library. To start with, Gridmap.js combines the default values and data statistics to generate a customised configuration. Then the data is parsed and converted into an appropriate format. Based on the configuration, Gridmap.js creates a feature layer with the data and then attaches it onto the base map.

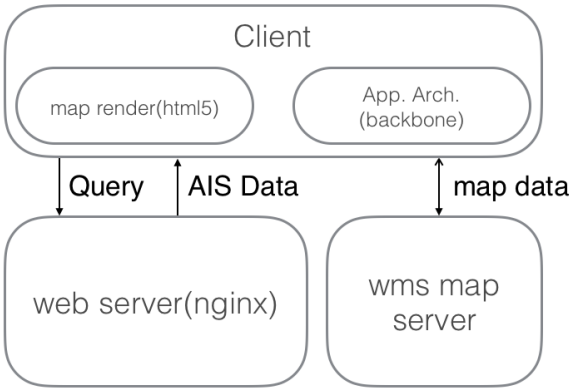

Figure 3.6: The Architecture of Visualization Client

3.3 System Architecture

The architecture of modern web applications can be divided into two categories: server-side oriented and client-side oriented. When a web application is built using a server-side web architecture, work including request parsing, data processing and page rendering, is done by the server. The only task for the client (the browser) is to display the web page it receives. Client-side web application, on the other hand, runs in the browser and connect to a server only for user authorisation and data exchange. Most of the work, including user interaction and page construction, is done by the client. The design of the visualisation client is the latter one. User interaction and data visualisation are handled in the browser. Moreover, the data is cached and can be used repeatedly. For example, user may want to visualise the data in a heat map and then in a grid map. Instead of requiring data from the server again, it simply creates the desired visualisation from the cached data. This design saves the resource of the server and takes advantage of the processing power of the client machine.

Figure 3.6 shows the architecture. When a user logs into AIS Data Explorer, the browser first gets a bunch of JavaScript codes. Then the code begins to construct the web page and requests a base map from a map server. The whole process can be done in seconds. When a request is submitted, the browser analyses it and then performs one asynchronous GET request to the server. The web server verifies user identity, retrieves the data from the back-end system (vOLAP or Spark) and sends it

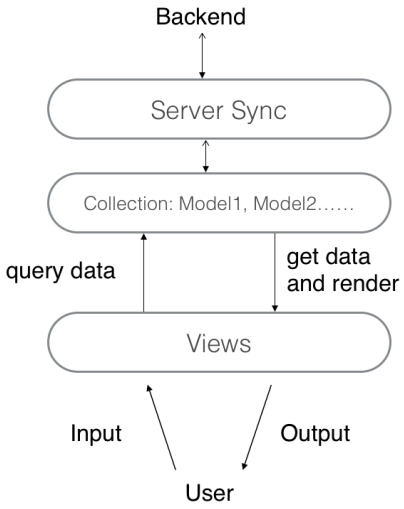

Figure 3.7: Web Client Architecture

to the client. At last, with the received data, the browser creates the visualisations on the web map.

3.3.1 Web Client Architecture

The client-side web architecture is quite similar to the server-side architecture, which follows a variant of model-view-controller design paradigm. Data is represented as Models which can be created, destroyed and saved to the server. Models connect to a back-end server for data exchange. A View is an atomic chunk of user interface, which renders the data from Models. It listens to model changes and renders itself from scratch. Figure 3.7 shows the architecture of client-side web application. Views listen to user input and call model methods to fetch the data. When data is received, views update their UI components. Here, the data is passed to the rendering library which creates visualisations on the web map. New visualisations can be extended simply by adding rendering libraries.

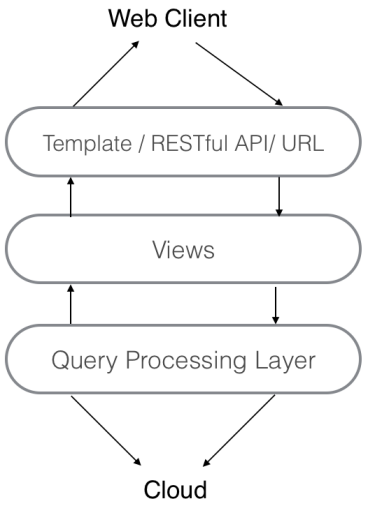

Figure 3.8: Web Server Architecture

3.3.2 Web Server Architecture

A typical Django application follows the model-view-template(MVT) architecture pattern. While the web server uses a modified architecture to satisfy its own requirements. It uses a RESTful API to exchange data with the web client. The view functions parses the request and translates the data into the appropriate format. The query processing layer provides methods to connect to different back-end systems. Figure 3.8 shows the architecture of the web server. When a request is received from the client, the view function parses it and calls methods in query processing layer to construct and send the query to the corresponding back-end system. Then the view function encodes the retrieved data in JSON and sends it back to the client. The web server works like a bridge between the client and the back-end system.

3.4 Conclusion

In this chapter, we introduce web mapping and web visualisation technologies and present their applications in the visualisation client. A rendering library, called gridmap.js, is implemented to support efficient grid map visualisation. At last, we present the system architecture.

Chapter 4

AIS Data Explorer : vOLAP based Query Processing

In this chapter, we introduce AIS Data Explorer, which has Velocity OLAP as backend to process marine data. To start with, we give a brief introduction to vOLAP and then explain how it achieves our design goals described in Chapter 1. In Section 4.2, we give an overview of data preparing. By analysing marine data, dimension hierarchies are defined and a star schema for vOLAP is designed. Section 4.3 introduces our contribution to vOLAP, which is the design and implementation of a module to enable vOLAP to handle histogram queries. In Section 4.4, we present the system architecture and explain how vOLAP works as a component of AIS Data Explorer. Lastly in Section 4.5, a conclusion is given.

4.1 Introduction

Velocity OLAP (vOLAP) [28] is a scalable, real-time OLAP system developed by Risk Analysis Lab at Dalhousie University. It supports operations including data point insertion, bulk insertion and aggregate query. In an experimental evaluation of vOLAP, using 18 worker instances for a dataset of 1.5 billion items, it can process streams of inserts and OLAP queries at a rate of approximately 200,000 queries per second [44]. vOLAP achieves our design goals for the back-end system: scalability and fast query on huge datasets. The challenge in this part is the limited functions of vOLAP. For heat and grid map visualizations, the system is required to support histogram query while vOLAP does not. Thus, we dig into the implementation of vOLAP and implement a module for histogram query.

Grid and heat map visualisations accept a list of data records as input, each of which contains coordinates of a rectangle area and aggregations of the data in that area.

Thus, a query for grid and heat map can be seen as a two dimensional histogram query which can be decomposed into many aggregate queries. This is where vOLAP comes into use. It can query the aggregation of multi-dimensional data efficiently and process thousands of such queries in parallel. However, vOLAP cannot support histogram query directly and does not have an interface to communicate with the visualisation client. In the section, the module we implemented handles these problems.

4.2 Data Preparation and Exploration

4.2.1 Data Preprocessing

The dataset, collected by AIS, is a big collection of marine related data which includes vessel static messages, vessel dynamic messages, configuration information and safety-related messages. A few of these messages may be incomplete or have illegal values, for example, one dynamic message does not have the information for latitude and longitude. Moreover, the messages could be faked by senders. Lack of extra information, it is impossible to fix errors and detect dummy messages. In the first step of data preprocessing, vessel static and dynamic messages are extracted from the AIS dataset. The incomplete and illegal messages are abandoned while the rest of messages are kept. The accuracy of the data is crucial in some scenarios. But for applications in this thesis, the data is appropriate to use.

A ship snapshot is defined as a data record which consists of MMSI, time, ship type, ship status, longitude, latitude, SOG and COG. It describes ship status at one specific instant. MMSI is a series of nine digits which are sent over a radio frequency channel in order to identify ship stations. Here, we assume it identifies one ship uniquely. Ship status describes vessel's navigation status that could be using engine, at anchor or moored. SOG is short for speed over ground, which is the speed of vessel relative to surface of the earth. COG is short for course over ground, which describes the direction over ground along which the vessel is currently moving. A ship snapshot is built from the preprocessed data.

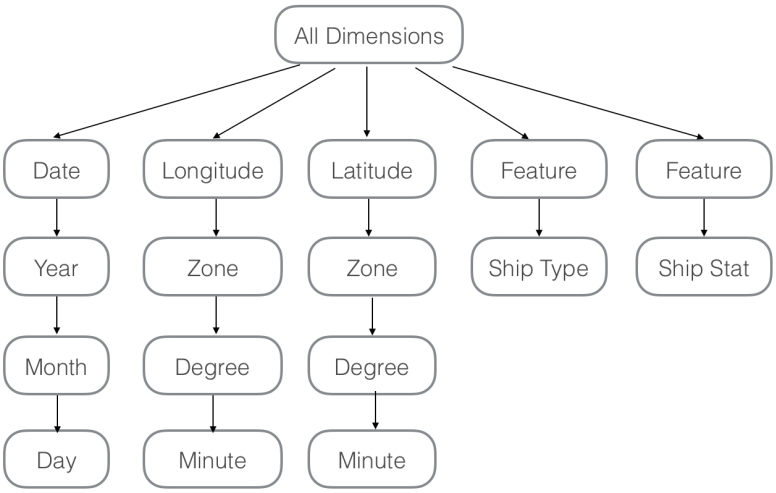

Figure 4.1: Dimension Hierarchies

4.2.2 Star Schema

vOLAP uses star schemas to represent multi-dimensional data models. Star schema is highly denormalised and query optimised, which enables vOLAP to achieve efficient query and data insertion. In this section, based on the requirements of data visualisations, we first define dimension hierarchies and then present the star schema designed for vOLAP.

In grid and heat map visualisations, data to be displayed can be filtered by dimensions including area of interest, time period, ship type and ship status. Thus, we need to define the corresponding dimensions in the star schema and area of interest can be converted into latitude and longitude dimensions. The attributes of one dimension can be organised as a hierarchy if they have parent-child relationships where a parent member summarises its children. Figure 4.1 shows the dimensions and the respective dimension hierarchies for each dimension. The first box for each dimension denotes the dimension name while the boxes below denote hierarchy levels from highest to lowest. Time dimension is designed to have three levels which are year, month and day. Considering the revisit time of the satellite, it is more accurate to generate data visualisations based on days than any smaller time unites like hours. Latitude and longitude dimensions are designed to have three levels which are zone, degree and

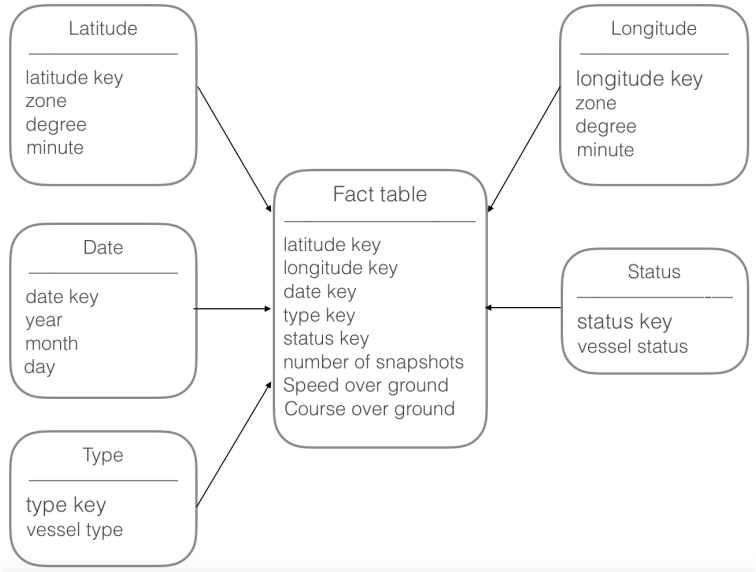

Figure 4.2: Star Schema

minute. In previous experiments [28] of vOLAP, it leads to a better performance if the number of attributes in the first level of the dimension hierarchy is small. Thus, we create an attribute, which is called zone, in the first level of the hierarchy. Take longitude dimension as an example, it is divided into four zones in the first level. Each zone covers ninety degrees and each degree covers sixty minutes. Since attributes in ship type and ship status dimensions do not have any parent-child relationships, they are organised as one level in the respective dimension.

The visualisation client allows user to visualise aggregations of the data on a web map. In the correspondence, the fact table of the star schema contains the numeric measurements including the number of vessel snapshots, SOG and COG. Figure 4.2 shows the star schema designed for vOLAP. It consists of one fact table and five dimension tables. One query consists of a set of values, each of which corresponds to one dimension. If the value for one dimension is not specified, vOLAP will aggregate all the data in this dimension. For example, by specifying ship type, ship status and time, we can create an aggregate query like ([tanker], [at anchor], [2013,01]). vOLAP will return the number of vessel snapshots, the sum of SOG and COG.

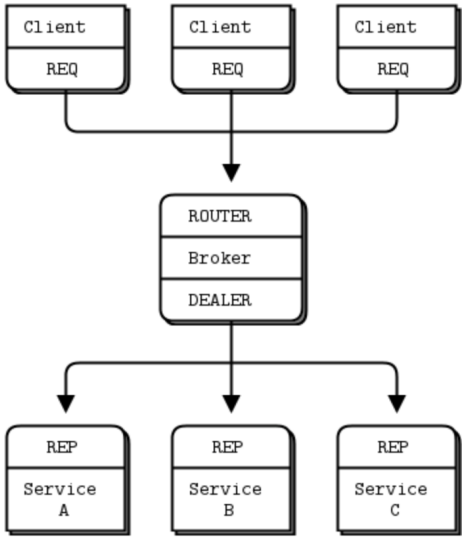

Figure 4.3: Router and Dealer [11]

4.3 Histogram Query in vOLAP

vOLAP supports basic operations including data point insertion, bulk insertion and aggregate query. A new module is implemented inside vOLAP to support histogram query. To start with, it analyses histogram queries from the client and generates a set of aggregate queries for vOLAP servers. Then it collects the replies, constructs the result for each histogram query and then sends them back to the corresponding client. Several techniques are used in this module. Zeromq [11] provides a basic structure for distributed messaging, which connects different components of the system. Protobuf [6] is used to serialise the structured data.

Zeromq [11] provides a set of messaging patterns which are used to build distributed applications. Router and dealer is one type of extended request-reply pattern, which connects multiple clients to multiple servers. This pattern centralises the knowledge of topology in a queueing broker, which works as a bridge between clients and servers. Figure 4.3 gives an example. Here, we use this pattern to build the module for histogram query and this design has two benefits. First, vOLAP can have multiple servers to handle query streams. And based on the amount of queries, servers can be added or removed dynamically. Since the queueing broker hides the topology

Figure 4.4: Message Structure

of vOLAP from clients, changes in vOLAP will not influence communications with clients. Second, it provides a way to support histogram query in vOLAP. The code in the queueing broker translates a histogram query into a set of aggregate queries. Then it collects and sends back the result to the original client.

Protobuf [6] provides a language-neutral, platform-neutral method for serialising structured data. Data structures are first defined by users and then special source code is generated by protocol buffers, which allows user to read and write data structures from and to a variety of data streams using a variety of programming languages. By using protocol buffer, an interface is designed and implemented, which enables vOLAP to communicate with the visualisation client. For communications inside vOLAP, it uses its own function, which is efficient and simple. For communications outside, it exchanges structured data with clients using protocol buffers. Take the visualisation client as an example, it is a web application written in javascript and python while vOLAP is written in C++. With protocol buffers, we can exchange structured data between python and C++ easily.

The queueing broker receives multiple histogram queries from different clients. Then these histogram queries are translated into thousands of aggregate queries. Since queries and replies are processed asynchronous, we need to identify each reply and reconstruct the result for each histogram query. The queueing broker also needs to keep the client address and send back the result to the correct client. Several methods are used to solve these problems. First, a message structure is designed to contain queries and replies. Figure 4.4 gives an example. A message from a client consists of the address and the actual data that could be a histogram query or an aggregate

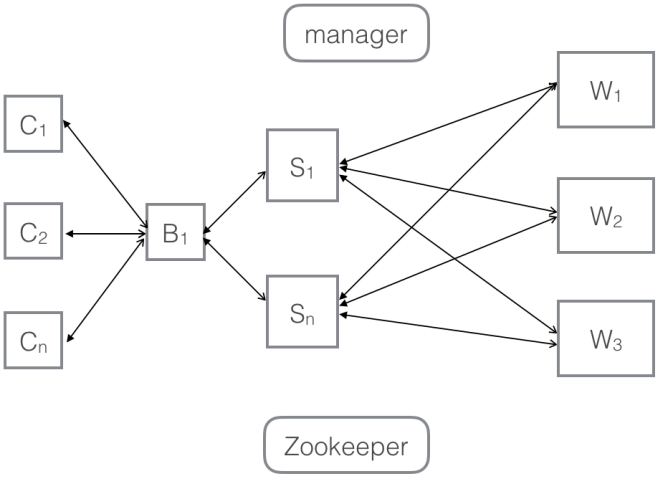

Figure 4.5: The Architecture of vOLAP

query. Message inside vOLAP consists of a key for identification and data. Second, a class is designed for histogram query which has three parameters: client address, the number of aggregate queries it generated and a pre-allocated data structure for results. It provides methods for handling histogram query, for example, translating it into aggregate queries, serialising and deserialising the data.

Figure 4.5 shows the architecture of vOLAP which supports histogram query. It consists of a queueing broker, a manager, a Zookeeper, multiple servers and workers. The queueing broker works as an interface for vOLAP. Since techniques used in the broker are widely supported by different programming languages, vOLAP can communicate with a variety of clients. Here, we show how a histogram query is processed. A message is received from a client which contains the client address and the serialised data. The broker firstly initialises an instance of the histogram query class. The instance extracts and keeps the client address. Then it parses the serialised data and generates subqueries with a group ID. The broker sends these subqueries to vOLAP servers and collects a set of replies, each of which is identified by the group ID and saved to the corresponding instance. Each time a reply is received, the broker checks if all replies are received for one histogram query. If it does, the result is serialised and sent back to the client.

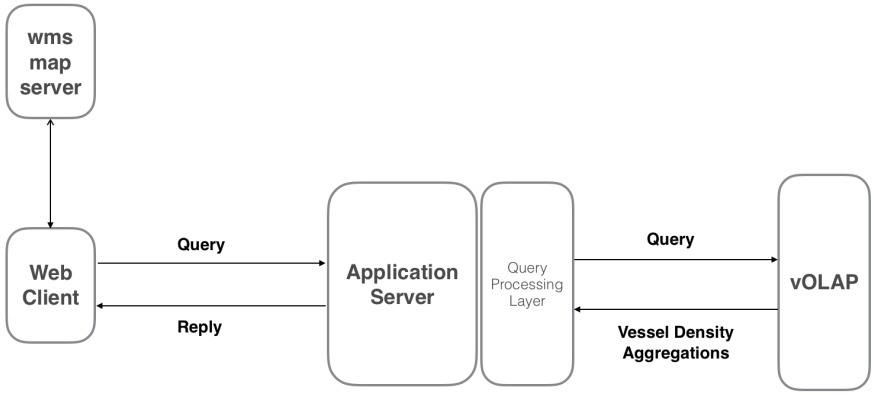

Figure 4.6: System Architecture

4.4 System Architecture

With vOLAP as the back-end system, AIS Data Explorer is able to support grid map and heat map visualisations. As dataset increase, vOLAP can be deployed on more machines to provide better performance. Aggregate values are computed and stored in memory while data records are inserted, which allows vOLAP to answer histogram queries efficiently. Figure 4.6 shows the architecture of AIS Data Explorer with vOLAP. It has two main components: the client for data visualisation and vOLAP for marine data processing. In the query processing layer, a module is implemented to communicate with vOLAP, which uses ZeroMQ [11]for message passing and Protocol Buffers [6] for data serialisation. It translates user input into OLAP queries. When reply is received, it parses data and converted it into a format accepted by the visualisation client.

4.5 Conclusion

In this chapter, we introduce vOLAP and its applications in AIS Data Explorer. vOLAP is a cloud based OLAP system which uses PDC-tree as a basic building block. A module is implemented inside vOLAP to support histogram queries. It also enables vOLAP to communicate with the visualisation client. However, vOLAP has limited functions and is difficult to extend. The desired system should provide

a flexible way to analyse data. In next chapter, we explore the use of the cluster computing framework Spark. It allows user to implement their own algorithms and the computation can be easily distributed over the whole cluster.

Chapter 5

AIS Data Explorer : Spark based Query Processing

In this chapter, we introduce AIS Data Explorer, which has Spark as backend to process big data. To start with, we give a brief introduction to Spark and explains how it achieves our design goals described in Chapter 1. In Section 5.2, we introduce how to support data visualisations. Spark provides a SQL interface for structured data processing. In Section 5.3, we introduce how to support data analysis. A distributed clustering method is designed and implemented. In Section 5.4, we give an overview of system architecture. At last, we conclude this chapter.

5.1 Introduction

Spark [38] is a cluster computing framework designed for big data. Different from Hadoop [25], it keeps intermediate results in memory. Iterative algorithms, which require to access data repeatedly, can be executed efficiently. Spark also powers a set of libraries including Spark SQL for structured data, MLib for machine learning and GraphX for graph processing. By using Spark, we can achieve our goals easily : scalability, fast query on huge dataset and data analysis at a large scale. As an example for data analysis, we design and implement a distributed version of DBSCANSD [35] that learns vessel traffic patterns from trajectory data. The challenge in this part is the design and implementation of distributed DBSCANSD. Pre- and post-processing of the data are performed to enable parallel computation.

The main abstraction in Spark is resilient distributed dataset (RDD), which is an immutable collection of elements partitioned across the cluster which can be processed in parallel [39]. Normally, Spark tries to create partitions automatically based on the cluster. Users can also develop their own strategies to partition the data. During

computation, a single task will operate on a single partition. For example, if one cluster has 40 task slots, it will have at most 40 partitions processed in parallel. User can apply a function to either each data record or each partition in a RDD. As we can see, RDDs are suited for applications that apply the same operations to all elements or subsets of a dataset while they are not suited for applications that make fine-grained updates to a shared state.

5.2 Data Visualisation

AIS Data Explorer allows user to explore and visualise maritime data in a browser. To answer such query, a subset of data is extracted and then the aggregations are computed. Sometimes, it may run computations on the whole dataset to get the result, for example, generate a global heat map of all the time. It requires the system to be scalable to handle large dataset. Spark SQL is a suitable system to satisfy such requirement, which is built on RDDs and able scale to multiple machines.

Different from relational databases, Spark SQL use DataFrames to represent structured data. A DataFrame is a distributed collection of data which is equivalent to a table. Maritime data is loaded from HDFS and then stored in one RDD. Based on the user defined data schema, the RDD is translated into a DataFrame. Then the DataFrame is registered as a table and cached in memory. User can run SQL queries to the dataset. Inside Spark, any SQL query will eventually translate into a series of parallel operations on RDDs. A two dimensional histogram query cannot be easily supported in SQL. To accelerate the process of histogram query, one extra value, called aggregationID, is computed. The map can be divided into many equal-sized boxes. Each box represents a rectangle area and is assigned to an index number. For each ship snapshot, we calculate the index number of the box it belongs to. A histogram query can be easily supported by grouping snapshots with the same index.

AIS Data Explorer supports three kinds of data visualisation: heat maps, grid maps and trajectories. Heat and grid map visualisations usually require aggregations of the data of interest. The aggregations could be the number of vessel snapshots, the

number of ships, the average of SOG or the average of COG. Trajectory visualisation requires a set of vessel snapshots. These snapshots are grouped by MMSI and sorted by timestamp. These queries can be expressed in SQL. The code below gives an example, which returns vessel density near Halifax Habour.

```
SQL Code : select aggregationID, count(distinct mmsi) from vessel
           where lat > 44.51 and lat < 44.66 and lon > -63.61 and
           lon < -63.40 group by aggregationID;
```

5.3 Data Analysis

In the past decades, the use of tracking systems, such as GPS (Global Positioning System) and AIS (Automatic Identification System), has greatly increased, which leads to an increasing number of tracking applications. One important task of these applications is trajectory clustering that is a process to discover traffic patterns from trajectory data. In [35], Liu Bo et al proposed a clustering method, called DBSCANSD (Density-Based Spatial Clustering of Applications with Noise considering Speed and Direction), which is designed for vessel trajectory clustering. One limitation of DBSCANSD is the performance. It takes nearly 10 minutes to run a sequential DBSCANSD on a dataset of 100 thousands data points. Since the time complexity of this algorithm is $\mathcal{O}(n^2)$, it is not able to process large amount of data. By applying index structures, such as k-d tree [43], R-tree, its time complexity could be $\mathcal{O}(n \log n)$. But it is still not scalable as the dataset increase. In this section, we design and implement a distributed version of DBSCANSD.

5.3.1 Representing Trajectory Data

Definition 1. (Trajectory in Maritime Domain) A trajectory is a finite sequence T = ((x_1,t_1),(x_2,t_2),...,(x_n, t_n)) where x_i is a set of < Latitude, Longitude, COG, SOG > and t_i is the timestamp.

A ship trajectory can be defined as a sequence of multi-dimensional data points,

each of which describes ship status at a specific instant (Definition 1) [35]. A ship trajectory point is defined as a vector, x_i. In the rest of this section, we also use point or data point instead.

5.3.2 Density-Based Spatial Clustering of Applications with Noise considering Speed and Direction

Under different circumstances, ships follow different moving patterns. It is common that different types of ships sail with different speed. For example, a high speed craft (HSC) usually moves faster than an oil tanker. Ships heading in different directions can behave differently. For example, cargo ships may sail into port at low speed while they may leave at high speed. Based on these observations, maximum speed variance (MaxSpd) and maximum direction variance (MaxDir) are considered in DBSCANSD [35]. As we know, the key idea of DBSCAN is to group points that are close to each other into one cluster. DBSCANSD adopt this idea and modify the definition of Eps-neighbourhood in [30] to the following:

Definition 2. Given a database D of trajectory points in one area, the Eps - neighbourhood of a trajectory point p, denoted by $N\epsilon(p)$, is defined by $N\epsilon(p) = \{q \in D \mid dist(p,q) < \epsilon$ and $|p.SOG - q.SOG| < MaxSpd$ and $|p.COG - q.COG| < MaxDir\}$

DBSCANSD requires 5 parameters (Data, eps, MinPts, MaxDir), MaxSpd. Data is a list of trajectory points. Eps is the reachable distance and MinPts is the reachable minimum number of points [30]. MaxDir and MaxSpd are the maximum direction variance and the maximum speed variance. Considering the Earth's curvature, DBSCANSD calculates the Geographical Distance [19] between two points.

5.3.3 Distributed DBSCANSD

In this section, we develop a parallel algorithm called distributed DBSCANSD which is based on the algorithm presented in [40]. Distributed DBSCANSD requires 6 parameters (BR, eps, MinPts, MaxDir, MaxSpd, MaxPts) as input. To start with, the

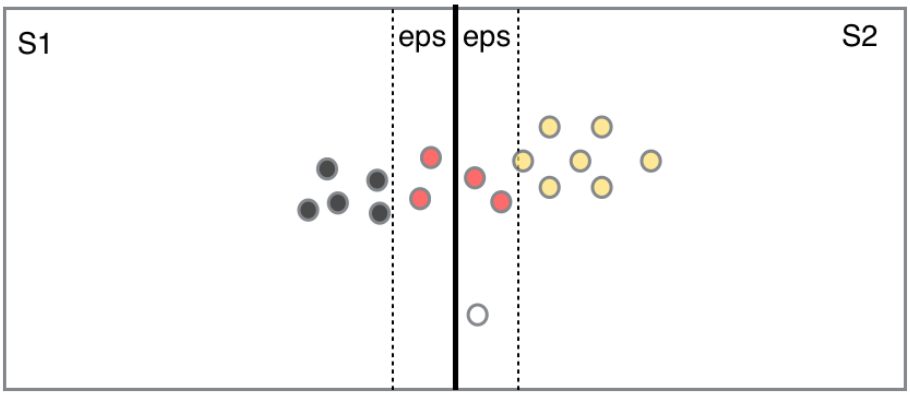

Figure 5.1: Clusters in different partitions

algorithm splits the input data into parts of approximately equal size. Then a sequential DBSCANSD is performed on each partition. At last, it collects outputs from different partitions and compute the final clustering result. BR is a rectangle bounding box that contains all the trajectory points. For each trajectory point, three extra attributes are assigned: pointId that is a unique identifier, clusterId that denotes the cluster it belongs to, partitionId that denotes its partition. The trajectory points are assigned unique identifiers before the partition stage. MaxPts is the maximum number of points expected in one partition. The pseudo code of distributed DBSCANSD is presented in Algorithm 1. **mapPartitions(func, params)** is a method of the RDD. It is runs the function on each partition of the RDD in parallel and then returns a new distributed dataset.

As discussed in [40], one crucial requirement of a partition strategy is that a DBSCAN based algorithm is able to work on the data in one partition independently without the knowledge of the data in other partitions. It has been proved in [34] that it is possible to execute DBSCAN in multiple partitions independently and get the same result as if DBSCAN is executed on the original space. Figure 5.1 shows an example. A rectangle bounding box S is divided into two non-overlapping boxes S_1 and S_2, each of which includes the core points that reside in it (the box) and the border points that are within one eps of its borders. In other words, some points may appear twice. Then DBSCAN is executed on S_1 and S_2 separately. Two clusters c_1 and c_2 are identified. The points can be divided into four categories: the black ones that belong to c_1, the yellow ones that belong to c_2, the white ones that belong to no cluster, the

red ones that belong to c_1 and c_2. Since some points belong to both clusters, there is a cluster c_3 that includes all the points in c_1 and c_2. And c_3 is the cluster founded by executing DBSCAN on the original space S. This is the basic idea of distributed DBSCANSD.

Algorithm 1 Distributed DBSCANSD

1: **procedure** DISTRIBUTED DBSCANSD(*BR, eps, MinPts, MaxDir, MaxSpd, MaxPts*)
2: $partitionedPoints \leftarrow densityPartition(BR, eps, MaxPts)$
3: $partitions \leftarrow expandPartitions(partitionedPoints)$
4: $partialClusteredPts \leftarrow partitions.mapPartitions(DBSCANSD, eps, MinPts, MaxDir, MaxSpd)$
5: $clusteredPts \leftarrow merge(partialClusteredPts, eps)$

Partition A partition algorithm is introduced in [40], which uses binary space partitioning (BSP)[41] to split the data into parts of roughly equal size. We adopt its idea and make modifications to improve the algorithms' efficiency. The new partition algorithm requires three parameters (BR, MinSize, MaxPts) as input. BR is a rectangle bounding box that contains all the points. MinSize is the minimum size of one partition and MaxPts is the maximum number of points in one partition. To start with, the algorithm splits the bounding box into two parts along its longest dimension. The assumption behind this is that data points are distributed evenly. Then the algorithm recursively splits the box until one of these boxes has less points than MaxPts or smaller than MinSize and this box is selected as one partition. At last, it generates a list of partitions of roughly equal size. When it comes to the implementation, the whole hierarchy of boxes is precomputed and the number of points in each box is counted. Then each box is checked whether it should be selected as a partition. Trajectory points are partitioned and each point is assigned to a partitionId. The time complexity of this algorithm is $\mathcal{O}(n)$. The pseudo code of the algorithm is presented in Algorithm 2.

Algorithm 2 Density Partition

```
1: procedure densityPartition(BoundingRectangle,MinSize,MaxPts)
2:     toSplit ← {BoundingRectangle}
3:     partitions ← ∅
4:     for BR ∈ toSplit do
5:         if POINTSIN(BR) ≥ MaxPts & isBigEnough(BR, MinSize) then
6:             (S1, S2) ← splitByMaxDim(BR)
7:             toSplit ← toSplit ∪ {S1, S2}
8:         else
9:             partitions ← partitions ∪ {BR}
10:    return partitions
11: procedure isBigEnough(BR,L)
12:    return LATITUDE(BR) ≥ L orLONGITUDE(BR) ≥ L
```

Local DBSCANSD After partition, trajectory points are split into partitions. To run DBSCANSD on each partition independently, each partition is expanded to include core points and border points. The output of local DBSCANSD is a set of trajectory points that are assigned clusterIds. If one point belongs to no cluster, its clusterId is set to be -1. Some points may appear multiple times and have different clusterIds.

Cluster Merging One cluster may spread across multiple partitions. We need to identify such clusters and put their parts together. The main goal of cluster merging is to assign a global identifier to trajectory points that belong to one cluster but reside in different partitions. As we discussed before, if one border point belongs to two clusters, then points in these two clusters belong to one bigger cluster. The cluster merging can be completed in two steps. First step, we find out local cluster identifiers correspond to a single cluster that spread across multiple partitions and then map these identifiers to one global unique identifier. Second step, based on the mapping, all trajectory points are relabeled. The algorithm presented in [40] provides a sequential method to merge local clusters. We adopt its idea and develop a parallel algorithm for cluster merging. Its pseudo code is presented in Algorithm 3

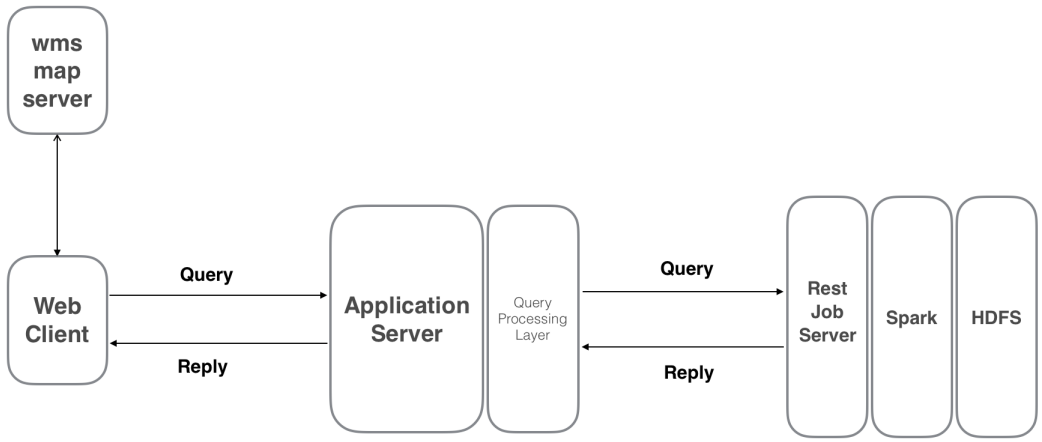

Figure 5.2: System Architecture

and Algorithm 4.

To start with, trajectory points that belong to no cluster or not within one eps of the borders of their partitions are filtered out and the remaining points are grouped by partitionId. The algorithm generates cluster identifier pairs in each partition and merge these pairs to generate a mapping. At last, all trajectory points are relabeled using this mapping. For simplicity, we only showed how to process points that belong to clusters. Points that belong to no cluster need to be processed separately.

5.4 System Architecture

Figure 5.2 shows the system architecture. Several techniques are used to build the back-end system. HDFS is used as a distributed storage system which keeps the AIS dataset on disks. Spark is worked as a distributed database and a computing engine. It loads data from HDFS and do cluster computing. Spark Jobserver [10] provides a RESTful interface for managing Spark jobs and job contexts. It works as a bridge between Spark and the visualisation client. To start with, the whole AIS dataset is preprocessed and then loaded into Spark SQL. The data is cached in memory for efficient query. After a SQL query is received, Spark SQL analyses it and the query will eventually translate into a series of operations on RDDs. The query result can be returned directly or further processed by algorithms like DBSCANSD.

Algorithm 3 Clusters Merging

```
1: procedure MERGE(Points, eps)
2:     borderPoints ← Points.mapPartitions(borderPointsFilter, eps)
3:     borderParts ← groupedByPartitionId(borderPoints)
4:     pairs ← borderParts.mapPartitions(generateClusterPairs)
5:     mappings ← mergePairs(pairs)
6:     partialClusteredPts ← shrinkPartitions(Points)
7:     clusteredPts ← partialClusteredPts.mapPartitions(relabelPoints, mappings)
8: procedure BORDERPOINTSFILTER(EPS)
9:     points ← ∅
10:    for Pt ∈ partition do
11:        if Pt within eps of partition borders & Pt is assigned to a cluster then
12:            points ← points ∪ Pt
13:    return points
14: procedure GENERATECLUSTERPAIRS(part)
15:    points ← ∅
16:    pairs ← ∅
17:    for Pt ∈ part do
18:        if Pt ∈ points then
19:            (c1, c2) ← (Pt.clusterId, prevPt.clusterId)
20:            if (c1, c2) ∉ pairs then
21:                pairs ← pairs ∪ {(c1, c2)}
22:        else
23:            points ← points ∪ {Pt}
24:    return pairs
25: procedure RELABELPOINT(mappings)
26:    for Pt ∈ partition do
27:        Pt.clusterId ← mappings[Pt.clusterId]
28:    return partition
```

Algorithm 4 Merge Pairs

```
1: procedure MERGEPAIRS(pairs)
2:     id ← 0
3:     g ← empty graph
4:     map ← empty map
5:     for (c_1, c_2) ∈ pairs do
6:         if c_1 ∉ g then
7:             add node c_1 to g
8:         if c_2 ∉ g then
9:             add node c_2 to g
10:        if edge(c_1, c_2) ∉ g then
11:            add edge(c_1, c_2)to g
12:    for subgraph ∈ g do
13:        for c ∈ subgraph do
14:            insert (c,id) to map
15:        id ← id + 1
16:    return map
```

5.5 Conclusion

In this Chapter, we gave an overview of Spark platform and explained how it achieved our design goals. A pipeline for marine data processing was built, which performs data cleaning, data processing and data analysis. We design and implement distributed DBSCANSD and integrated it into the existing system, which allows user to visualize the clustering result in the visualisation client.

Chapter 6

Evaluation

In this chapter, experiments are conducted to evaluate the performance of the proposed framework. To start with, we conducted experiments with the visualisation client and evaluated the performance of map rendering. The results are presented in Section 6.1. Then in Section 6.2, we evaluated and compared the performance of vOLAP and Spark. Section 6.3, we present the results of an experimental evaluation of distributed DBSCANSD.

6.1 Performance of the Front-end : Data Visualization

In this section, we compared the performance of different visualisations including grid maps, heat maps and trajectories. For grid and heat maps, we performed a two dimensional histogram query on the AIS dataset dated from 1 Feb, 2013 to 30 April, 2013 and used its result as test data. The test dataset consists of 41374 data records, each of which contains the coordinates of one rectangle area and the number of vessels in that area. For trajectory visualisation, the test dataset consists of 112784 ship snapshots dated from 1 Feb, 2013 to 3 Feb, 2013, near Libreville, Gabon. The experiments are conducted on a laptop of Mac OS X with an Intel I5 processor (2.4 GHz), Intel Iris graphics (1536 MB) and a RAM of 8 GB (1600 MHz DDR3). The software is Google Chrome (version 50.0.2661, 64-bit). The application is a Javascript program running in the browser.

Figure 6.1 shows the results. Two methods for grid map visualisation are evaluated and compared. A native method is provided by Leaflet and an improved method is implemented by us. As the number of points increase, the time for the native method

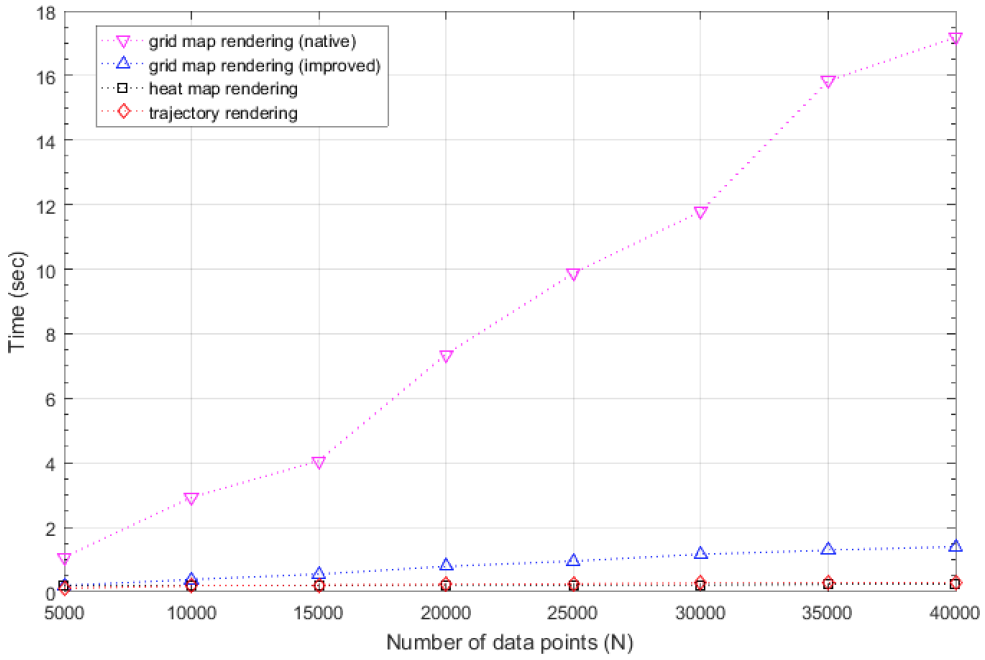

Figure 6.1: Time (seconds) for map rendering as a function of the number of data points

increase dramatically while that for the improved method increase slowly. The performance of our method outperforms that of the native method by ten times. Although the outputs of two methods are similar, the underlying rendering techniques are different which determines the performance gap. The implementation of heatmap.js [1] is based on html5 canvas, which enables efficient heat map rendering. The time for heat map is less than 0.5 second although the number of points increase from 5000 to 40000. Because of the simplicity of trajectory rendering, it is faster than other visualisations.

6.2 Performance of the Back-end: Data Loading and Query Processing

A test dataset of 100 million ship snapshots is created from a subset of the AIS data dated from February 1, 2013 to April 30, 2013. Each ship snapshot contains the following fields: maritime mobile service identity (MMSI), timestamp, ship type, ship status, longitude, latitude, speed over ground and course over ground. The test dataset is used to evaluate the performance of the back-end system. The experiments

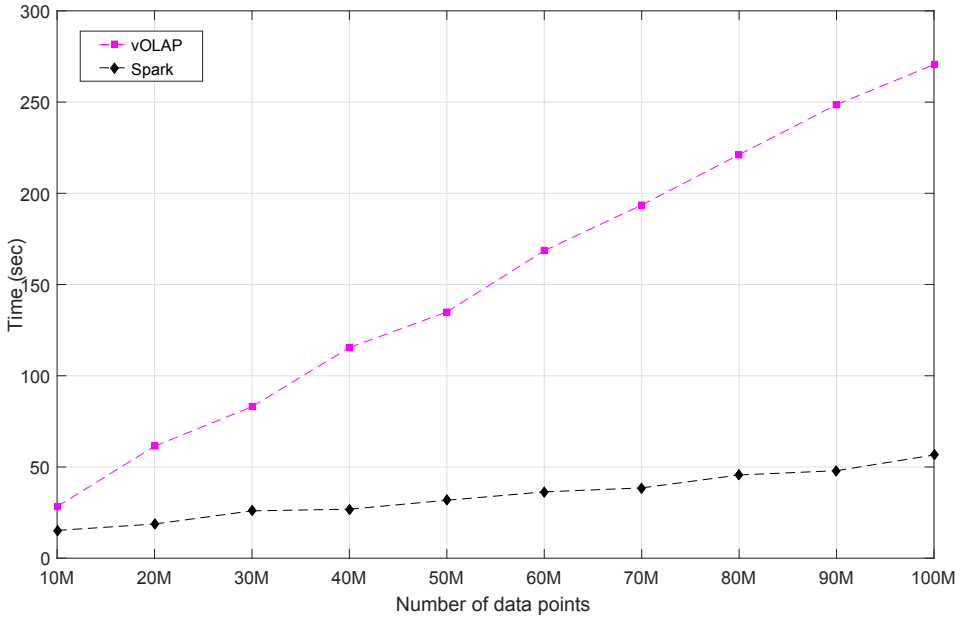

Figure 6.2: Time (seconds) for a query as a function of the number of data points

were conducted on West Cloud of Compute Canada. For each worker instance, it has 4 virtual cores (Intel Xeon CPU E5-2650 v2 @ 2.60GHZ) and 15 GiB memory. The OS image used was the standard CentOS 7.0. All instances run in the same network and talk to each other with private IPs. In the local network, data transfer can be as fast as 100 megabits per second. A public IP is provided for user access.

6.2.1 Data Loading

We tested how the time for data loading changes when the size of data increase. 5 workers are used in this experiment. Figure 6.2 shows the results. As the number of data records increase from 10M to 100M, the time for data loading increases slowly. Spark shows better performance than vOLAP. For data loading, vOLAP does more work than simply loading the data into the system. When data records are inserted, it precomputes the aggregate values, which improves the performance for queries.

6.2.2 Query Processing

We test how the time of different queries for vOLAP and Spark changes as we increase the number of workers. To start with, 30 million items from the test dataset are inserted into vOLAP and Spark. Then a two dimensional histogram query is executed. It computes aggregations of the data in a rectangle area (N 10° 0′W 40° 0′,S 10° 0′E 10° 0′), dated from February 1, 2013 to April 30, 2013. One rectangle area is represented by a pair of points which denote upper left corner and lower right corner respectively. The aggregations are the number of snapshots and vessels. A report query, which work on the same data as histogram query, is also executed. There is always some performance fluctuation on the cloud. Based on the result of experiments, the average of 10 same queries is considered as a stable measure. The evaluation results are shown in Figure 6.3 and 6.4.

Spark is good at answering queries that apply the same operation to a subset of the data. For one histogram query, it applies a series of parallel operations on RDDs, which use the resource of the cluster effectively. In vOLAP, a histogram query will translate into a set of aggregate queries, each of which is processed separately. These aggregate queries are sent to workers which have the data they required. Thus, only some of the workers are engaged in the computation and the processing power of the whole cluster is not fully utilised. As we can see, Spark shows better performance on query processing.

6.3 Performance of the Back-end: Distributed DBSCANSD

We tested the performance of Distributed DBSCANSD on two different dataset. A test dataset of 3 million trajectory points is generated from a subset of AIS data dated from February 1, 2013 to March 1, 2013. All these points are resided in one rectangle area (N 29° 0′E 47° 0′, N 2° 37′E 100° 0′) that includes Arabian Sea and Bay of Bengal. Another one is a synthetic dataset generated by scikit-learn's dataset loading utilities [8]. This dataset consists of 3 million trajectory points and contains 100 different clusters. The used hardware has been described in Section 6.2. The

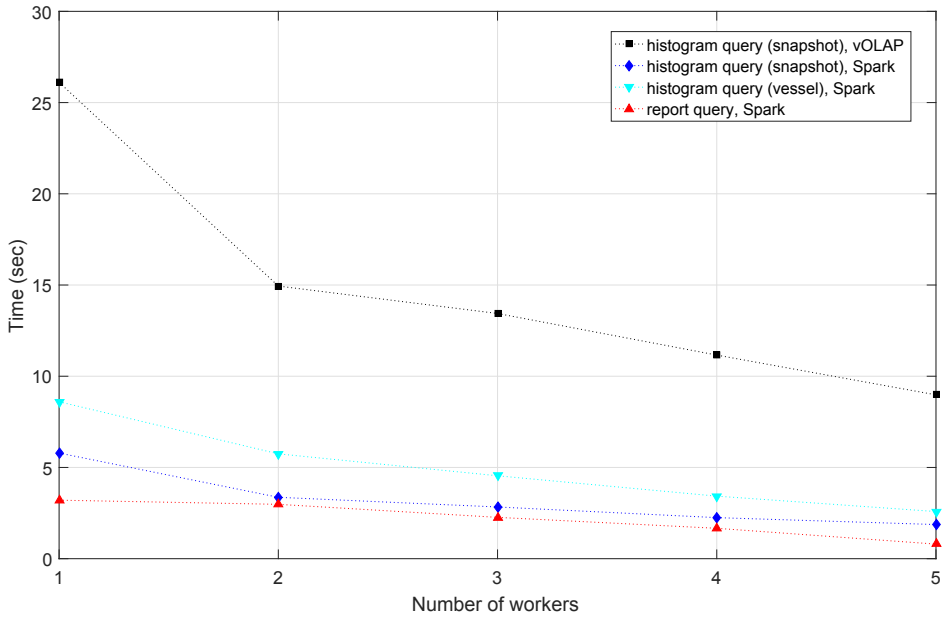

Figure 6.3: Time (seconds) for a query as a function of the number of workers

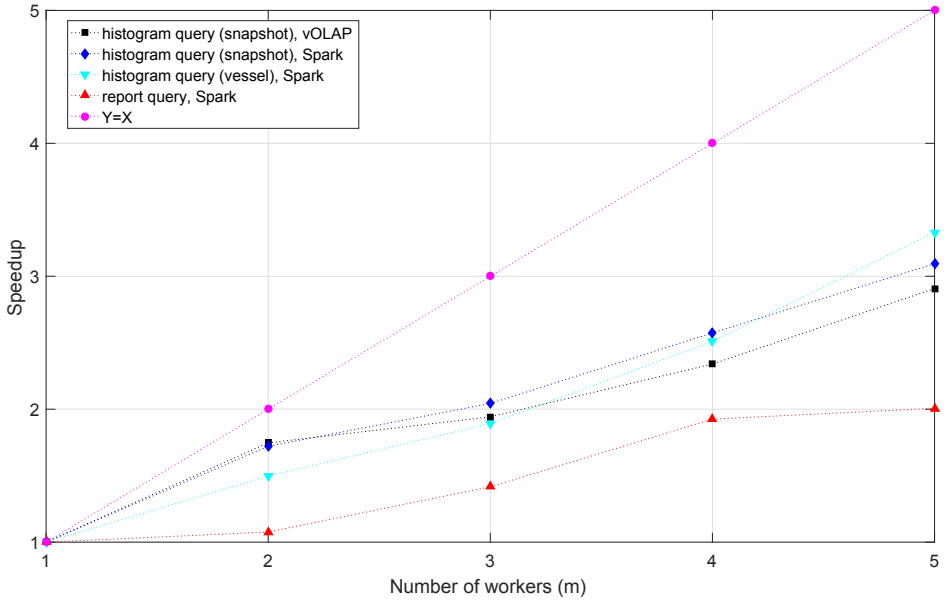

Figure 6.4: Speedup for a query as a function of the number of workers

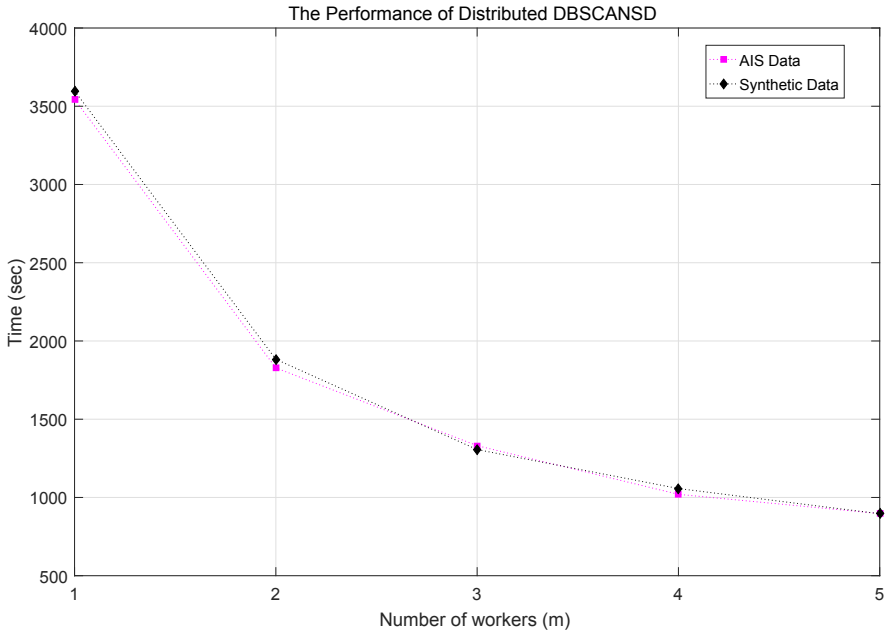

Figure 6.5: Time (seconds) for distributed DBSCANSD as a function of the number of workers

parameters of the algorithm are set to be 10km (eps), 100 (MinPts), 20 (MaxDir), 200 (MaxSpd) and 10000 (MaxPts). Since no auxiliary index structure is used, the time complexity of local DBSCANSD is $\mathcal{O}(n^2)$ where n is the number of points in that partition. We test how the time of distributed DBSCANSD changes as we increase the number of workers. And the impact of system size is also evaluated. The results of our experiments are shown in Figure 6.5, 6.6 and 6.7.

As we can see, although data is not partitioned evenly, distributed DBSCANSD can still scale as the number of workers increase. This is due to the design of Spark task scheduling. The number of tasks that can be executed in parallel is determined by the number of cpu cores in the cluster. One task is assigned to one partition. Partitions without tasks are queued in memory. Once the work on one partition is complete, Spark will assign a new partition to the task. The number of partitions generated by Distributed DBSCANSD is much more than the number of tasks. Thus, more partitions are processed in parallel as the number of workers increase. As a result, the program becomes faster.

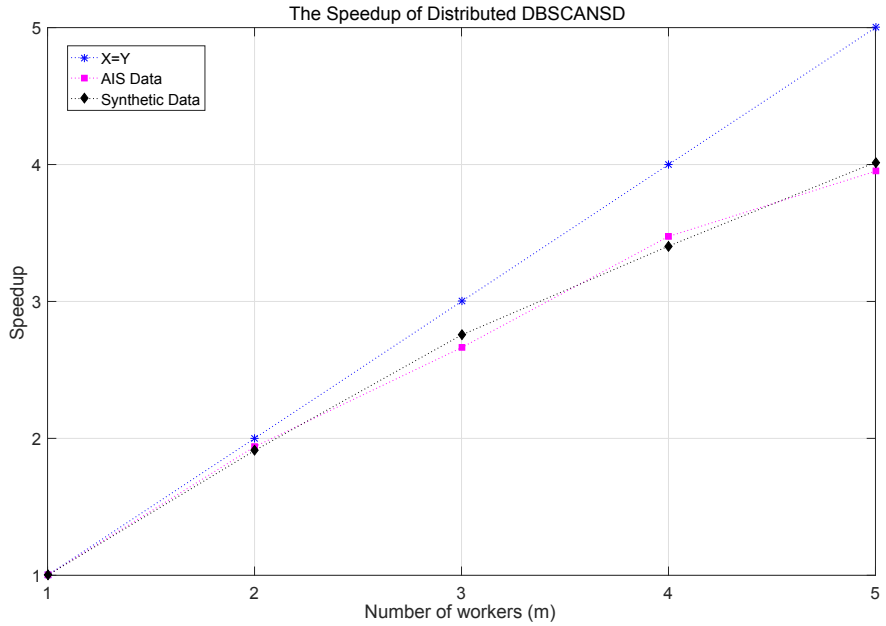

Figure 6.6: Speedup for distributed DBSCANSD as a function of the number of workers

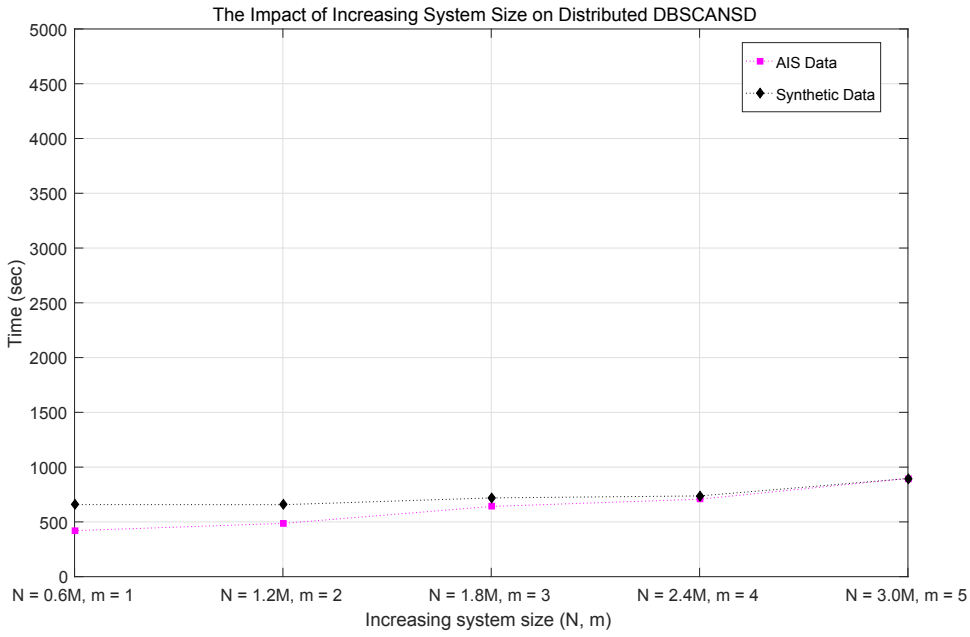

Figure 6.7: Time (seconds) for distributed DBSCANSD as a function of system size

Chapter 7

Conclusions and Future Work

In this thesis, we propose a scalable framework, AIS Data Explorer, to analyse and visualise marine navigation data. The main challenge of this work is how to effectively handle data at a large scale. The main contribution of this work is to provide a method to help researchers explore and analyse huge datasets.

With increasing in the number of vessels, transponders and satellite receiving statistics, the size of marine navigation data is huge and increasing rapidly. In five years from 2010 to 2014, Big Data Institute at Dalhousie University has accumulated nearly 4 terabytes data and the size keeps increasing by 100 gigabytes per month. Lack of scalable tools, many problems are studied on relatively small dataset.

AIS Data Explorer is designed to solve this problem. Users are able to explore the dataset by visualising data of interest on an interactive map. We also design and build a data processing pipeline that performs data extraction, data cleaning, data analysis and data visualisation. As an example of large scale data analysis, we design and implement distributed DBSCANSD that discovers vessel traffic patterns from vessel trajectories. Two different techniques are explored to process marine navigation data. vOLAP is able to compute aggregations of structured data efficiently. However, its functions are limited and difficult to extend. Different from vOLAP, Spark is optimised for iterative jobs and interactive analytics. Based on resilient distributed dataset, Spark is extended to support different applications including structured data management, streaming data processing, machine learning and graph processing. Although vOLAP and Spark are designed for processing big data, ideas behind these two techniques are quite different. The design of vOLAP follows client-server architecture. It explores multicore and multiprocessor parallelism. While the original purpose of Spark is to improve Hadoop. The intermediate result in Hadoop needs to

be written back to disks which involves overhead caused by data replication and disk I/O. To solve this problem, Spark allows user to keep data in memory and query it repeatedly. Thus, iterative jobs and interactive analytics can be well supported. At last, we evaluated the performance of AIS Data Explorer on cloud. The experiment results demonstrate its efficiency. Depend on data size, visualisations can be typically generated in seconds. Distributed DBSCANSD is able to work on millions of trajectory points, which is impossible for sequential DBSCANSD.

As for future work, we would like to build interested applications for researchers in AIS Data Explorer. And this framework can also be reused in other domains where visualisations and data analysis are required. Another work is to improve distributed DBSCANSD. Vessel trajectories are spatial-temporal datasets that can be better split by a 3 dimensional partition strategy.

Bibliography

[1] *Heatmap.js, Dynamic Heatmaps for the Web.* https://www.patrick-wied.at/static/heatmapjs.

[2] *Leaflet, a Javascript library for interactive maps.* http://leafletjs.com.

[3] *Mapbox, An Open Source Mapping Platform.* https://www.mapbox.com.

[4] *Mapnik, A Open-source Toolkit for Server-based Map Rendering.* http://mapnik.org.

[5] *PostGIS, Spatial and Geographic Objects for PostgreSQL.* http://postgis.net.

[6] *Protocol Buffers.* https://developers.google.com/protocol-buffers.

[7] *QGIS, An Open Source Geographic Information System.* http://www.qgis.org/en/site/.

[8] *Scikit-learn, data loading utilities.* http://scikit-learn.org/stable/datasets.

[9] *Spark, cluster mode overview.* http://spark.apache.org/docs/latest/cluster-overview.html.

[10] *Spark Job Server.* https://github.com/spark-jobserver/spark-jobserver.

[11] *ZeroMQ, The Guide.* http://zguide.zeromq.org/page:all.

[12] *ZooKeeper, A Distributed Coordination Service for Distributed Applications.* http://zookeeper.apache.org/doc/trunk/zookeeperOver.html.

[13] *Oracle Data Mart Builder Population Guide : Understanding Star Schemas.* Oracle, 2003.

[14] *M.1371 : Technical characteristics for an automatic identification system using time-division multiple access in the VHF maritime mobile band.* Electronic Publication, 2014.

[15] Berchtold Stefan, Keim Daniel A. and Kriegel Hans-Peter. *The X-tree : An Index Structure for High-Dimensional Data.* Proceedings of the 22nd VLDB Conference (Mumbai, India): 2839, 1996.

[16] Guttman Antonin. *R-trees: a dynamic index structure for spatial searching.* Vol. 14. No. 2. ACM, 1984.

[17] Lakshman, Avinash, and Prashant Malik. *Cassandra - A Decentralized Structured Storage System.* ACM SIGOPS Operating Systems Review 44.2:35-40, 2010.

[18] Heather Ball. *Satellite AIS for Dummies.* Wiley, Mississauga, ON, 2013.

[19] Veness, Chris. *Calculate distance, bearing and more between latitude/longitude points.* `http://www.movable-type.co.uk/scripts/latlong.html`.

[20] Wikipedia contributors. *Apache Spark.* Wikipedia, The Free Encyclopedia, 2016.

[21] Wikipedia contributors. *GeoJSON.* Wikipedia, The Free Encyclopedia, 2016.

[22] Wikipedia contributors. *Online analytical processing.* Wikipedia, The Free Encyclopedia, 2016.

[23] Wikipedia contributors. *R-tree.* Wikipedia, The Free Encyclopedia, 2016.

[24] Wikipedia contributors. *Scalable Vector Graphics.* Wikipedia, The Free Encyclopedia, 2016.

[25] Borthakur, Dhruba. *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2007.

[26] De Vries, Gerben Klaas Dirk and Maarten Van Someren. *Machine learning for vessel trajectories using compression, alignments and domain knowledge.* Expert Systems with Applications, 39(18):1342613439, 2012.

[27] Beckmann, Norbert, et al. *The R*-tree: an efficient and robust access method for points and rectangles.* Vol. 19. No. 2. ACM, 1990.

[28] Dehne. F., et al. *Scalable real-time OLAP on cloud architectures.* Journal of Parallel and Distributed Computing 79 : 31-41, 2015.

[29] Engle, Cliff, et al. *Shark: fast data analysis using coarse-grained distributed memory.* Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012.

[30] Ester, Martin, et al. *A density-based algorithm for discovering clusters in large spatial databases with noise.* Kdd. Vol. 96. No. 34, 1996.

[31] Frber, Franz, et al. *SAP HANA database: data management for modern business applications.* ACM Sigmod Record 40.4:45-51, 2012.

[32] Gray, Jim, et al. *Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals.* Data mining and knowledge discovery 1.1:29-53, 1997.

[33] Harati-Mokhtari, Abbas, et al. *Automatic Identification System (AIS): data reliability and human error implications*. Journal of navigation, 60(03), 373-389, 2007.

[34] He, Yaobin, et al. *Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce*. Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE, 2011.

[35] Liu Bo et al. *Knowledge-based clustering of ship trajectories using density-based approach*. In Big Data (Big Data), 2014 IEEE International Conference on (pp. 603-608). IEEE, 2014.

[36] Palma, Andrey Tietbohl, et al. *A clustering-based approach for discovering interesting places in trajectories*. In Proceedings of the 2008 ACM symposium on Applied computing (pp. 863-868). ACM, 2008.

[37] Rocha, Jose Antonio MR, et al. *DB-SMoT: A direction-based spatio-temporal clustering method*. Intelligent systems (IS), 2010 5th IEEE international conference. IEEE, 2010.

[38] Zaharia, Matei, et al. *Spark: Cluster Computing with Working Sets*. HotCloud 10 : 10-10, 2010.

[39] Zaharia, Matei, et al. *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[40] Cordova Irving and Teng-Sheng Moh. *DBSCAN on Resilient Distributed Datasets*. High Performance Computing and Simulation (HPCS), 2015 International Conference on. IEEE, 2015.

[41] Berger Marsha J. and Shahid H. Bokhari. *A partitioning strategy for nonuniform problems on multiprocessors*. Computers, IEEE Transactions on 100.5: 570-580, 1987.

[42] Ester Martin, Jorn Kohlhammer and Hans-Peter Kriegel. *The dc-tree: A fully dynamic index structure for data warehouses*. Data Engineering, 2000. Proceedings. 16th International Conference on. IEEE, 2000.

[43] Bentley, Jon Louis. *Multidimensional binary search trees used for associative searching*. Communications of the ACM 18.9 : 509-517, 1975.

[44] Kong Quan. *Scalable Real-time OLAP System for the Cloud*. 2014.

[45] Laxhammar, R. *Anomaly detection in trajectory data for surveillance applications*. 2011.

[46] Sellis Timos, Nick Roussopoulos and Christos Faloutsos. *The R+ Tree: A Dynamic Index for Multi-Dimensional Objects*. 1987.

[47] Pallotta Giuliana, Michele Vespe and Karna Bryan. *Vessel pattern knowledge discovery from ais data: A framework for anomaly detection and route prediction.* Entropy, 15(6), 2218-2245, 2013.