A Big Data Infrastructure for Real-time Traffic Analytics on the Cloud

Yikai Gong
ORCID: 0000-0003-3082-9859

Submitted in total fulfilment of the requirements of the degree of
Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

November 2019

Produced on archival quality paper.
Abstract

With the increasing urbanisation occurring globally, cities are facing unprecedented challenges. One major challenge is related to traffic and the increasingly common congestion issues that arise in cities. At the same time, digital data is being created across all walks of life by industry, governments and society more generally. The term “big data” has now entered common vernacular. Big data can include officially captured data, e.g. from traffic measurement systems from government organisations such as VicRoads in Australia, as well as other forms of data generated by the population at large, e.g. social media.

This thesis explores the unique characteristics of traffic related data and focuses on the development and evaluation of an underpinning Cloud-based platform that can tackle some of the unique big data challenges related to such data. In particular, the thesis focuses on challenges related to the volume, velocity and variety of traffic data. We explore how different forms of data including official sensor data such as the Sydney Coordinated Adaptive Traffic System (SCATS) that is widely rolled out across Victoria and supported by VicRoads can be processed in real time, as well as how social media data such as Twitter can be used as a cheaper proxy for SCATS to better understand traffic in cities. We also develop novel real-time clustering algorithms that tackle the unique spatial and temporal aspects of traffic related data.
Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,

2. due acknowledgement has been made in the text to all other material used,

3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

__________________________________________
Yikai Gong
ORCID: 0000-0003-3082-9859, November 2019
Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Richard O. Sinnott for his continuous support of my Ph.D. study, for his patience, motivation and immense knowledge. His guidance helped me throughout my research and the writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD.

I would also like to thank the rest of my thesis committee: Prof. Stephan Winter, Dr. Shiping Chen, and Dr. Paul Rimba, for their insightful comments and encouragement, as well as for the questions which motivated me to broaden my perspectives on the research challenges explored in this thesis.

In addition, I would like to thank NeCTAR Research Cloud for the free use of the cloud resources during my PhD study. I also thank VicRoads for access to much of the data used in my research.

Last but not the least, I would like to thank my parents for supporting me spiritually throughout writing this thesis and my overseas life in Australia.
The published academic papers of this work are listed below. Elements from these articles are included in this thesis. The inclusions of the papers are highlighted in the relevant sections within this thesis.

**Research Publications**


Several other publications were also produced throughout the course of this PhD including:


The software developed for this thesis are open-source and available at my GitHub project \(^1\). A brief introduction to these software components is provided below. For further details, please check the README.md file in each code repository. In total, there are 17,614 lines of code written in Java, Scala, Bash, JavaScript and Docker Script.

**Software Outcomes**

- **SMASH Cluster** \(^2\): This repository contains all of the source code for deploying the SMASH cluster including the Docker source files, configuration files, and bash scripts. The details of the cluster software components and their Docker-based deployment are introduced in Chapter 5. A command line based tool is available to deploy and manage the remote SMASH cluster. There are 3,960 lines of code in this repository.

- **Social Media Harvesters** \(^3\): This repository contains the social media data harvesters written in Java. Twitter4j \(^4\) and Java Spring framework \(^5\) were used as the major libraries. There are 2,672 lines of code in this repository.

- **Command-line-based library for managing remote SMASH cluster** \(^6\): This repository contains the JavaScript library used to ease the deployment and management of the SMASH cluster. It requires nodeJS \(^7\) as a run-time engine. The Grunt \(^8\) library is used as a framework for the task runner to execute a variety of commands related to the SMASH cluster. There are 4,305 lines of code in this repository.

---

\(^1\)https://github.com/project-rhd
\(^2\)https://github.com/project-rhd/SMASH_cluster
\(^3\)https://github.com/darcular/TwitterProject
\(^4\)http://twitter4j.org
\(^5\)https://spring.io/
\(^6\)https://github.com/project-rhd/grunt-clouddity
\(^7\)https://nodejs.org
\(^8\)https://gruntjs.com/
Data (real-time) analyzers on SMASH: This repository contains all of the developed applications running on SMASH that were used in this thesis. It is composed of multiple modules (managed by Maven/Java) for handling different types of data and different type of processing (e.g. real-time stream processing and batch-based processing). GeoTools is the most heavily used library in this repository as most of the data processing are related to spatio-temporal calculations. There are 6,677 lines of code in this repository.

https://github.com/project-rhd/smash-app
https://geotools.org/
Contents

1 Introduction 1
   1.1 Research Question ................................................. 2
   1.2 Big Data Challenge and Solutions ................................. 3
       1.2.1 High Performance Computing and Parallel Computing ...... 5
       1.2.2 Grid Computing .............................................. 5
       1.2.3 Highly-scalable Computing .................................. 7
       1.2.4 The Development of Cloud Computing ....................... 7
       1.2.5 Hybrid Cloud Computing ................................... 13
   1.3 Cloud Computing for Tackling Big Data Challenges ............. 14
   1.4 Container-based Cloud Computing ................................. 16
   1.5 Social Media and Urban Traffic Data ............................ 18
       1.5.1 Twitter and Instagram Data ................................ 18
       1.5.2 Official Traffic Resources .................................. 20
       1.5.3 Research Hypothesis ........................................ 21
   1.6 Thesis Structure .................................................. 22

2 Literature Review 25
   2.1 Urban Traffic Challenges and Research Context ................... 25
   2.2 Big Data and Cloud Computing ................................... 29
       2.2.1 Background and Architecture of Big Data Systems ........ 32
       2.2.2 Cloud Computing and Big Data-as-a-Service ............... 34
       2.2.3 Challenges and Open Research Issues ....................... 38
       2.2.4 Feature-oriented BDaaS for Urban Traffic Analytics ...... 41
   2.3 Social Media data Analysis and Related Work .................... 43
   2.4 Related Work on Clustering Algorithms ........................... 45
   2.5 Conclusions ...................................................... 49

3 Data Structures and Data Collection Systems 51
   3.1 Social Media Data .................................................. 52
       3.1.1 What is Twitter and Twitter data? .......................... 52
       3.1.2 Discussion on the Geo-location Accuracy of Twitter Data ... 57
       3.1.3 Collecting Twitter Data ..................................... 61
6.3.4 Parallel Processing Tasks For Each Tick 119
6.4 RT-DBSCAN Implementation 133
6.5 Case Study Benchmarking RT-DBSCAN using Spark Streams 136
   6.5.1 Cloud Resource Usage 136
   6.5.2 Software Stack and Dockerized Deployment 137
   6.5.3 Stream Data Source for Case-studies 137
   6.5.4 RT-DBSCAN Performance Benchmarking 138
6.6 Conclusions 146

7 Urban Traffic Data Analytics on SMASH 147
   7.1 Data Description 148
      7.1.1 SCATS Traffic Data 148
      7.1.2 Social Media Data 149
   7.2 Observations on the Social Media Clusters via SMASH 150
   7.3 Correlation between Social Media and Urban Traffic Data 156
      7.3.1 Identifying Abnormal Urban Traffic Volumes and Social Media Data 156
      7.3.2 Traffic Data-driven Abnormality Analysis 160
      7.3.3 Social Media Data-driven Abnormality Analysis 165
   7.4 Conclusions 170

8 Conclusions and Future Work 173
   8.1 Summary of this Thesis 173
   8.2 Future Work 177
      8.2.1 Potential Extensions to the Research 177
      8.2.2 Other Areas 184

Glossary 187

Bibliography 189
List of Figures

1.1 Parallel workloads and their corresponding computer platforms 13
1.2 The architecture of Docker containers running on a Cloud VM 17

2.1 Taxonomy of Big Data as a Service (BDaaS) offerings in the Cloud 38
2.2 Available Cloud services on the AWS Dashboard 40

3.1 Screenshots of the Twitter App — Home page and tweeting page 54
3.2 An Example of a Typical Tweet in JSON format 56
3.3 Positioning modes of the 3G iPhone: (a) GPS position; (b) WiFi position; and (c) cellular position [207]. The blue disc indicates the estimated positional error 58
3.4 Web page of the Traffic Signal Volume Data on the DataVic Portal 63
3.5 Web page of the Traffic Signal dataset on the DataVic Portal 64
3.6 Overview of the AURIN web interface 65
3.7 Example of Shop PSMA Street Network Data on AURIN 67

4.1 PSMA Road Data example from AURIN 72
4.2 Harvesting Tweets from the Street Network (black centroids represent request areas) 73
4.3 Harvester label added to a given Tweet when stored in the database 75
4.4 Software Architecture for Harvesting and Clustering Tweets 79
4.5 Clustering of Tweets in Sydney on May 2015 (small triangles are potential congestion (tweet clusters) 80
4.6 Example of real-time social-media analytics 81
4.7 Screenshots of the Waze App — Road Map Page and Reporting Page 83

5.1 SMASH Architecture compared to the traditional Hadoop ecosystem 91
5.2 Traditional Hadoop Distributed File System Architecture 91
5.3 Hadoop MapReduce model 93
5.4 SPARK workflow on SMASH 96
5.5 High-level SMASH Software Architecture and its Dockerized Implementation 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>Geographic bounding box of the case study and the locations of the SCATS sensors</td>
<td>161</td>
</tr>
<tr>
<td>7.8</td>
<td>The Processing Pipeline for the Traffic Data-driven Analysis</td>
<td>162</td>
</tr>
<tr>
<td>7.9</td>
<td>A detected spatio-temporal social media cluster and its surrounding traffic flow volume status</td>
<td>166</td>
</tr>
<tr>
<td>7.10</td>
<td>Surrounding Traffic Abnormality Detection with Different Sizes of Social Media Clusters</td>
<td>169</td>
</tr>
<tr>
<td>8.1</td>
<td>Traffic flow heatmaps of the Adelaide CBD on 2012-05-17 at 0:00, 6:00, 12:00 and 18:00 created through the SMASH platform</td>
<td>180</td>
</tr>
<tr>
<td>8.2</td>
<td>Heatmaps of total aggregated traffic flow in Victoria in 2008 and 2014 created through the SMASH platform</td>
<td>181</td>
</tr>
</tbody>
</table>
Traffic is a common and critical issue facing all modern cities due to the increased urbanisation of populations and the increased demands for mobility of people in urban contexts. Traffic can bring both positive and negative impacts on urban and human society. On the positive side, it can bring mobility, prosperity and social well-being, e.g., social activity, traveling, goods delivery, etc. Nevertheless on the negative side, it can bring pollution, noise and congestion in cities. This issue is especially noticeable in rush hour. The volume of vehicles on the road network has many impacts other than simply causing delays. For example, it can have impacts on health [28, 58, 87, 93].

Traffic issues can trouble every citizen and cost huge amounts of money to address — typically at the taxpayers expense. As a single example, the Australian
Government made a 10-year, $100 billion budget available for transport infrastructure from 2019 [170]. Optimally using such vast amounts of money is an important task for city planners and society more generally. This raises many questions regarding urban infrastructure and especially in understanding the status of daily traffic and steps that may be taken to reduce and/or manage traffic congestion.

1.1 Research Question

Traffic analysis has been explored by many researchers. Historically, much work focused on traffic simulation [31, 106, 201] and not on “actual” traffic data. Capturing and processing larger scale data sets and the technologies and infrastructures used to support this is now more feasible than ever before, especially with larger scale Cloud-based systems now widely available. However, traffic data has several unique characteristics and challenges with regards to the volume (amount of the data), velocity of the data (speed of production) and the variety (heterogeneity of data) that must be addressed. The first question we consider in this thesis is whether modern Cloud-based capabilities provide all features required for real-time big data and if not then, can a targeted traffic-based data capture, processing/analysis and visualisation system be developed that addresses these limitations? Furthermore, capturing accurate, real traffic information can be expensive and can involve use of GPS tracking systems and/or sensor networks on the road networks. Our second research question in this thesis is whether we can find more accessible and cheaper data resources that can be used to understand and analyze urban traffic? That is, is there a proxy that can be used to substitute for more accurate but expensive to collect official traffic data? Thirdly, traffic data has specific spatio-temporal characteristics. Our third question is whether new algorithms can be identified that address the specific spatio-temporal challenges of traffic analytics. When looking around the data that is created in urban settle-
ments (cities) where most traffic issues arise, it can be observed that a staggering amount of data across all walks of human life is now being generated. This covers data as diverse as social media data, transport data, health data with data collected by government, by industry and often across a wide array of research endeavors. Some data such as traffic data can be expensive to collect. As such they are often limited in their spatio-temporal coverage. Some data are very cheap such as social media data, e.g. Twitter. Such data are huge in scale and relatively easy to collect, with in many cases spatial information provided, i.e. the tweet may have a latitude and a longitude given directly. Given the widespread adoption and availability of social-media data, we consider whether it can be used as a proxy to analyze urban traffic phenomenon.

These three research questions form the hypotheses of this thesis which are presented in Section 1.5.3.

1.2 Big Data Challenge and Solutions

Urban research data sources like social media and traffic data are considered as “Big Data”. Such data are often huge in size (Volume), fast in generation and transmission (Velocity) and can be highly heterogeneous (Variety). These features make it hard or even impossible to handle such data within a single machine (server) due to the limited hardware capability and the associated data deluge. For example, the speed of newly arriving data can be faster than the speed of data processing in a single computation node. This can result in a cascading effect that slows down and eventually shuts down the server. By offering more powerful hardware with increased clock speeds and larger storage volumes with faster read/write speeds, computer scientists have developed many solutions to tackle aspects of such Big Data challenges. However, the data volume, velocity and variety means that historical existing solutions cannot adequately meet Big Data demands.
Ideally solutions would offer a scalable group of computational resources to meet the real time requirements of such Big Data challenges. The size of the computational group should ideally be simply and automatically adjusted by adding or removing resources based on actual (real-time) demand. These resources may be servers or other software-based solutions that can be dynamically deployed as required. Such capabilities have been explored in Distributed Computing [139] for many years, however the capabilities needed to tackle real world Big Data challenges like traffic has hitherto not been fully realised. The software components for storing, querying and processing Big Data need to be distributed/shared across multiple computation nodes to improve efficiency and performance. For example, large data sets should be processed in parallel across multiple nodes. The system should also scale up and down as required, e.g. use more resources for processing data at rush hour and less resources when traffic reduces.

There are two key challenges to apply distributed computing to Big Data. Firstly, computation infrastructures need to provide collections (clusters) of nodes connected by high throughput networks and handle the heterogeneity. Secondly, coordination frameworks are needed to allow the nodes to tackle potentially huge and volatile (bursty) workloads. This impacts directly on performance and scalability. Here we regard performance as the capability of particular computation components to provide a certain amount of capacity or throughput, while scalability is considered as the ability of a system to expand or contract to meet varying demand [23]. For many purposes, scalability and performance are orthogonal [23]. Many existing distributed computing infrastructures have been designed driven by these two dimensions, i.e. Peer-to-Peer Computing [16] and High Performance Computing [30].
1.2 Big Data Challenge and Solutions

1.2.1 High Performance Computing and Parallel Computing

High Performance Computing (HPC) and Parallel Computing trace their origins back to the 1960s [166]. They represent early examples of distributed computing. HPC typically focuses on a centralized system which provides multiple computation nodes (servers) that are connected by a high-speed intra-network. Parallel Computing is a general term focused on decomposing computing tasks into sub-tasks so that they can be computed in parallel by multiple executors to improve the throughput and efficiency. There are many different methods to distribute data across computation nodes, so that tasks can be broken into many parallel tasks. The most commonly used parallel methods in HPC is based on the Message Passing Interface (MPI). MPI is a programming paradigm/architecture for communication and sharing data between processes, allowing to take extremely large data sets and process them in parallel across given clusters. In MPI-based computing, large volumes of data can often be transmitted frequently between nodes and the processing required on one node can be based on the results from other nodes. MPI-based computing requires very low latency for data transmission whilst offering high performance in single clusters through the high bandwidth interconnects. This makes it an ideal parallel computing architecture for HPC and hence a widely adopted model for high performance and high throughput programming.

1.2.2 Grid Computing

Grid Computing is another form of computing infrastructure used for parallel computing. The idea of Grid Computing was originally raised in the early 1990s as a metaphor to make computer power as easy to access as the electric power grid, where clients/users do not need to know the source and location of the power — just how to access and use it through the electric socket [64, 66]. In Grid Computing, a collection of computers/nodes (servers) owned by potentially
multiple parties (organisations) in different locations are loosely coupled together so that users can share the combined power of those computational resources. Nodes in Grids typically involve independent/non-interactive tasks involving a large number of files accessible over the Internet or lower-speed networks. In contrast to HPC, Grid Computing applications often focus on optimizing computing and data access for larger latency wide-area networks, while HPC applications often focus on optimizing computing and data access for low-latency local networks. MPI-based parallel computing can also be conducted on Grid Computing platforms however the network can usually be a bottleneck when dealing with large data volumes, and hence computation nodes may become idle. As a result, Grid Computing applications often have independent tasks running on each node and larger scale movement of data for live data processing is typically non-optimal.

The method for breaking a given task into multiple non-interactive tasks is often referred to as embarrassingly parallel computing [195]. In contrast to MPI-based parallel computing, embarrassingly parallel computing has no data sharing requirements between computational nodes. Instead a large data set can be chopped into pieces, distributed to a large pool of workers for processing, and then the processed data brought back and reassembled, e.g. by a master process. Nowadays, embarrassingly parallel computing approaches are commonly applied on top of MPI clusters where a cluster of HPCs forms the grid. Examples include the European Grid Infrastructure (EGI) [53] and the China National Grid (CNGrid) [42]. Some Grid Computing infrastructures also utilize peer-to-peer architectures [10] or Volunteer Computing [8], e.g. OurGrid [130], World Community Grid [88] and XGrid [89].
1.2.3 Highly-scalable Computing

Internet pioneers such as Amazon and Google focused attention on the performance of computation nodes whilst dealing with network latency through consolidation of resources in major data centres to deliver core services, e.g. online shopping in the case of Amazon and web search in the case of Google. They identified that there was an opportunity to offer such infrastructure to others as a core business offering. They recognised that the cost-effectiveness of highly scalable computing could be achieved by offering access to such resources on a pay-as-you-go model. The web servers for Amazon’s retail website used to run at a very high transactions per second (TPS) on each server. Later, they migrated their service to a more scalable architecture capable of running at a much lower TPS. Although each web server had lower individual performance, the entire system became significantly more scalable and cheaper whilst achieving a higher overall performance.

With the growing needs of network services and the storage demanded for increasing amounts of data, industry giants such as Amazon realised the importance of such scalability and cost/performance issues. Cloud Computing was proposed to maximise the cost-effectiveness of such services. Cloud Computing has since become very popular to both industry and academia alike in a very short time period [92].

1.2.4 The Development of Cloud Computing

The term Cloud Computing has had a lot of different meanings in its relatively short history, but nowadays it can be seen as a combination of a variety of services built on top of highly scalable computing platforms. The evolution of Cloud Computing can be divided into four key stages:

- Centralized mainframes:
The Cloud concept was first introduced in the 1960s and evolved from the concept of the Intergalactic Computer Network [109]. This referred to a shared-access mainframe which allowed multiple users access to a central computer (through terminals). Similar to the modern Cloud Computing, organizations like IBM (https://www.ibm.com) built such mainframes for cost-effectiveness, because it was too expensive to buy and maintain a computer for each employee (at that time).

• **Mainframes running isolated computing environments:**
  In the 1970s, virtual machine (VM) technology was invented [75, 169]. This took the 1960’s shared-access mainframe/Cloud to the next level. VM technology made it possible to run multiple operating systems simultaneously in isolated environments on a single physical machine. For the Cloud at the time, each user of the shared-access mainframe was able to access their own computing environment (VM) without being disturbed by other users at the operating system level, whilst they collectively shared the same physical machines and hardware comprising the mainframe.

• **Virtualized services over distributed networks:**
  Initially, telecommunications providers only offered dedicated point-to-point network connections. In the 1990s, they started offering virtual private network (VPN) services with comparable quality of service, but at a lower cost. This topology of the network provided users with shared access to services. Before this point, ordinary users were required to have all their facilities, e.g. storage, servers, printers, and software services connected over the same network. The term Cloud Computing extended the boundary to cover all the connected services and their underlying network infrastructures.

The term Cloud was initially used to express the space between and users and the service providers; it was later extended to represent these virtual-
ized services accessible over networks. In 1999, Salesforce (https://www.salesforce.com/) became one of the first major organisations to successfully utilize Cloud Computing as a core business capability. They delivered their software services to end users via the Internet, offering the Software-as-a-Service (SaaS) based on Cloud [49]. In this model, their applications could be accessed by any customer with Internet access, provided that the customers had purchased their software. Importantly, this was offered in an on-demand, cost-effective manner over the network.

- **Modern Cloud Computing for accessing resources/services anytime/anywhere:**

  In the 2000s, a large number of companies saw the benefits of Cloud Computing and the Cloud industry grew. This marked the beginning of modern Cloud Computing. In 2002, Amazon introduced its web-based retail services based on an underlying Cloud Computing Infrastructure. This gave them the flexibility to use their excess computing resources/capacity much more efficiently. At that time, Amazon only used 10% of Amazon’s cloud capacity for its core marketplace business. In 2006, Amazon created the subsidiary Amazon Web Service (AWS — https://aws.amazon.com) and introduced their Elastic Compute Cloud (EC2). These made Amazon’s Cloud facilities/services available to the public by cost-effectively charging costumers according to their resource usage. After several years of development and evolution, AWS now provides a wide range of Cloud-based services over the Internet through the AWS dashboard. Examples of these include:

  - **Storage Services:** including Simple Storage Service (S3), Elastic Block Storage (EBS), Elastic File Storage (EFS) and others;
  
  - **Database Services:** including Amazon RDS, which provides a SQL Database service; Amazon Redshift, which provides a scalable data
warehouse as service; Amazon Neptune, which provides a graph database service, amongst many others;

- **Network Services**: including Amazon Route 53, which provides a Domain Name service; AWS Virtual Private Network, which provides a secure tunnel service for accessing AWS networks; Amazon Virtual Private Cloud, which provides a private subnet service, amongst many others;

- **Compute Services**: including Amazon Elastic Compute Cloud (EC2), which provides virtual machines as a service; Amazon Elastic Container Service (ECS) which provides containers (Docker — https://www.docker.com) and related container orchestration services; AWS Lambda, which provides an environment service for running code directly based on a serverless architecture, amongst many others others;

- **Application Services**: including Amazon Simple Notification Service (SNS), which provides message delivery services; Amazon SageMaker, which provides a platform service for scalable machine learning; Amazon ElasticSearch Service, which provides a platform service for managing ElasticSearch, amongst many others;

Among all these services, EC2 is the core part of AWS which provides virtual server instances (VMs) as a service to customers. This type of service, also called Infrastructure-as-a-Service (IaaS), is the most basic service underpinning modern Cloud Computing. Users of IaaS typically come from a technology background and have degrees of IT expertise. Through IaaS, users are able to access computing power (VMs) without being responsible for the installation or maintenance of the underlying hardware and networking systems.

In 2006, Google launched various Cloud production services such as Google Docs (https://docs.google.com). Google Docs provides a Software-as-a-
Service (SaaS) service that allows users to create, update and share documents remotely over the Internet. This has since grown to support a wide array of capabilities, e.g. Google Sheets, and associated storage capabilities.

In 2007, Netflix (https://www.netflix.com) started its streaming video service using the AWS Cloud. In 2008, Google launched a second Cloud service: the Google App Engine (https://cloud.google.com/appengine/) for application development. In the following years, Google released a series of cloud computing platforms, including Google Compute Engine, Google Cloud Datastore, Google Cloud Storage, Google Cloud SQL, and others. All of these platforms form the base for the Google Cloud Platform (https://cloud.google.com) which provides a complete Cloud Computing ecosystem for numerous organisations. Similar to the Amazon Cloud, Google use its Cloud internally for its end-user products, such as Google Search and YouTube (https://www.youtube.com). It also offers similar Cloud resources and associated infrastructure and services to public customers as AWS and their IaaS and SaaS offerings.

Around the same time, IBM (IBM Cloud — https://www.ibm.com/cloud/), Oracle (https://cloud.oracle.com) and Microsoft through their Azure platform (Azure — https://azure.microsoft.com) launched Cloud platforms for IaaS, SaaS and other Cloud flavours. Many other major companies including Intel, eBay, PayPal later established their own Clouds either individual private Clouds or by using facilities from existing Cloud providers.

In 2010, Rackspace (https://www.rackspace.com) and NASA (https://www.nasa.gov) jointly launched an open-source Cloud software initiative known as OpenStack. The OpenStack project was intended to help organizations offer Cloud computing services running on standard hardware. It provided a standardised deployment for building IaaS Clouds. This free platform has since been used by many Cloud users/providers including
NASA, Intel and eBay, to address some of the key issues of heterogeneity that arose with commercial offerings and hence avoid vendor lock-in.

One of the key benefits of using Clouds as computing platforms is the price. Cloud providers offer many flexible charging methods that can be adapted to suit different types of ad hoc users and organisations. Users are charged by the amount of resource they use, where the resource can be compute, storage, networking or access to and use of applications. Many companies and organizations are happy to run their businesses on the Cloud instead of owning or renting traditional servers/facilities. They save money by not having to maintain such machines and only pay for the services they need when they need them. Some of the cost benefits of Clouds are discussed in [168].

Software such as distributed file systems and distributed data processing engines ease the way in which large-scale data (Big Data) can be processed over Cloud infrastructures. As a result, an increasing number of Grid Computing workloads and HPC workloads with relatively high network latency demands are now being conducted on Cloud platforms. According to a recent benchmark [123], the performance of Cloud-based HPC is competitive to traditional HPC systems for many types of workload. Figure 1.1 illustrates the common types of parallel workloads and their associated/suggested distributed computing platforms. As seen, Cloud Computing is suitable for the majority of parallel computing tasks, which were historically dominated by HPC and Grid Computing.

Furthermore, since 2010 with the widespread use of smart phones and the development of mobile networks, people (owners of any network connected devices) have now been able to access any Cloud-based services or resources at any time and anywhere using a variety of mobile devices. Looking forward, it is expected that Cloud Computing will be closely integrated
with the Internet of Things (IoTs) and combined with Artificial Intelligence (AI) technologies, it is expected that Cloud Computing will encompass and support many aspects of everyone’s daily life.

1.2.5 Hybrid Cloud Computing

Clouds that are limited to a single organization are often called private Clouds or enterprise Clouds. Clouds that are available to many organizations are typically called public Clouds. Amazon AWS is the largest public Cloud. It had 1.4 million servers across 28 availability zones in 2014 [68]. The combined usage of private Clouds, public Clouds, and third-party computing resources is called Hybrid Cloud Computing. This offers the benefits of multiple deployment models. In early 2008, NASA’s OpenNebula (https://opennebula.org) became the first open-source software for deploying private and hybrid clouds.

Even though renting servers/facilities from a public Cloud can be cheaper, many organizations still want to physically have their own servers on their private Cloud, e.g. to store sensitive data in-house. For organizations that want
to save the cost of implementing all services/applications on private Clouds, a Hybrid Cloud architecture can give businesses more flexibility and offer richer data processing options. Organizations are able to gain flexibility and leverage the computing power of public Clouds for basic and non-sensitive tasks, while keeping business-critical applications and data on-premises. Another use-case of Hybrid Cloud Computing is to handle fluctuations in processing demand on private Clouds. An organization can temporarily use resources from the public Cloud to meet the overflow needs from the private Cloud [120]. Cloud bursting is a deployment model whereby applications can run on a private Cloud and (seamlessly) burst to a public Cloud when the demand for computing capacity increases. The organization only needs to pay for the extra resources from the public Cloud when they are needed. A hybrid Cloud usually involves heterogeneous hardware/systems across multiple Cloud infrastructures. They are also called Cross-platform Hybrid Clouds. Re-implementing and redeploying/configuring applications to run across different Cloud providers and geographic locations remains a challenge for using hybrid Cloud models effectively, especially when dealing with larger scale data sets. Methodologies like Container technology cater for aspects of this challenge. They support lighter weight and more portable solutions for more rapid deployments across Cloud environments [134].

1.3 Cloud Computing for Tackling Big Data Challenges

According to a recent report [118], approximately 2.5 quintillion bytes (2.3 trillion Gigabytes) of data were created daily in 2018 and this rate of growth keeps accelerating. By 2020, 43 zettabytes (43 trillion Gigabytes) of data will be created and 1.7 MB data will be created every second. In 2018, approximately 90 percent of all existing data was generated in the last 2 years.

Another report [51] focusing on the amount of social media data generated every minute of the day identified that 473,400 tweets were sent on Twitter; 49,380
photos were posted on Instagram; 2,080,333 snaps were sent on Snapchat, and 12,986,111 text messages were sent.

Such volumes of data cannot be handled and processed through traditional computing infrastructures and software techniques. Through hardware virtualization technology [103], Cloud Computing provides a range of key capabilities for such Big Data challenges. These include:

- **Agility**: A traditional computing infrastructure usually needs system administrators to install/configure physical servers with the required systems and applications. Cloud Computing allows users to focus on the business of Big Data without worrying about the underlying hardware/software infrastructure. With the help of server clones and more recently, container technologies, a Cloud user can launch thousands of VMs or containers to work on data processing demands in only a few minutes. In addition, many popular Big Data applications are provided as part of PaaS (Platform-as-a-Service) or SaaS offerings from Cloud providers, e.g. database clusters and data processing engines. Such ready-to-use, production-oriented Cloud services can save both time and money in processing Big Data.

- **Reduced Cost**: While the size of Big Data keeps growing, the analytic requirements of Big Data continually evolve over time. In many cases, users can scale up/down their Cloud resources usage on demand and in a flexible manner. The associated price of the Cloud resources can also vary over time [173]. This gives users some flexibility to reduce their overall cost.

- **Improved Data Analysis**: Many Cloud-based Big Data Analytics tools have been developed, e.g. BigTable storage system (https://cloud.google.com/bigtable/), MapReduce models for Big Data processing [47] and distributed in-memory processing engines [205]. Such applications/platforms form an ecosystem for Big Data processing on the Cloud and provide users with a
variety of options to meet the requirements of their own specific needs and demands with regards to Big Data analytics.

1.4 Container-based Cloud Computing

A key capability of Clouds is that they should be scalable. Container technology has arisen in the last few years to minimize the manual labour required for software/system deployment and to speed up service deployment across Clouds. A container is an isolated virtual environment in which software runs directly on the kernel of the host operating system with access to a restricted subset of the underlying resources. Docker is the most popular container application at present. A container also refers to a package of software and dependencies that run inside a virtual environment. A binary file that represents such a package is called a container image. An image can be used to launch containers with precisely the same configuration/environment irrespective of the operating systems on which they run.

The virtual machines offered by the Cloud providers are typically generated in a variety of ways, e.g. using hardware-assisted virtualization [174]. Examples of such systems include KVM (https://www.linux-kvm.org) and Xen (https://xenproject.org). In hardware-assisted virtualization, the hardware provides architectural support to allow guest operating systems to be run in isolation on the hardware platform. Container virtualization such as Docker, CoreOS rkt (https://coreos.com/rkt/), OpenVZ (https://openvz.livejournal.com), Mesos Containerizer (https://mesos.apache.org), LXC Linux Containers (https://linuxcontainers.org), by contrast offer operating-system-level virtualization.

Figure 1.2 illustrates the typical architecture of Docker containers/applications running on a Cloud VM. The Docker engine is running as a middleware on the host operating system of the VM and provides environment isolation to the upper containerized applications (containers). There are many benefits in using
container technologies such as Docker for urban transport analysis:

- Several software components can rely on the same dependency but require different versions of the code. Containers such as Docker save the configurations and help to resolve conflicts that might arise.

- The containers naturally decompose a software stack into modules thereby making it much easier to maintain and install software systems.

- The modules of software components and data often need to be packaged/stored in separate container images and with mounted volumes. This supports scaling/migrating traffic data analytics platform across the Cloud, since the data can be hosted (stored) in a more permanent form outside of the given container.
The Docker engine has become very popular for Cloud computing and various extensions and projects have arisen for automating deployment, scaling, and management of the containerized applications on the Cloud. Examples of orchestration and management tools for images include Docker Swarm and Kubernetes [163].

It was identified that official traffic data is typically expensive to collect [102]. In this thesis, we consider the extent that social media can be used as a proxy for official traffic data. The question that immediately arises is: what social media platform is suitable for traffic analysis?

1.5 Social Media and Urban Traffic Data

As identified, social media is now abundant and used by all walks of society. A key requirement for the selection of the social media platform to use is the programmatic access to data. In this work, the target social media data resource was Twitter and Instagram data, due to their abundance and availability of programmatic APIs that allow such data to be collected.

1.5.1 Twitter and Instagram Data

Both Twitter and Instagram provide public APIs for accessing/collection data. Twitter opens its data stream to the public to allow authenticated users access to near real-time tweet streams for free. It is noted that whilst not incurring a cost to access the data from Twitter, the processing and storage of tweets and posts does incur some form of indirect cost. In this thesis ‘free’ social media data implies that it does not incur a cost to be paid to the data provider (i.e. Twitter, Instagram). Much of the research reviewed in Chapter 2 focus on establishing

---

1It is noted that Instagram refined the process to access their data in the course of this thesis — largely as a result of the Cambridge Analytica issue that arose with Facebook in 2018. See https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal
knowledge from social media for different purposes. Most of these research utilize methods such as Natural Language Processing (NLP) to try to ‘understand’ the content of social media data, i.e. extracting knowledge from the text that is sent. However, one of the most significant issues with such data is that the quality of any given tweet or post on a social network cannot be guaranteed. People can send meaningless content or even tell lies. Fake News is now a widespread term used to capture such erroneous reporting [5, 157].

Different to such research efforts, the research interest in this thesis is the use of spatio-temporal information captured and sent from devices by social media users, i.e. the latitude/longitude of the tweet/post. Indeed, one of the hypotheses of this thesis is that certain aggregations/collections of spatio-temporal information contain sufficient useful knowledge that can help in understanding urban traffic status/issues. Importantly, this location information leads directly to privacy issues. This privacy issue is explored in detail in many works, e.g. [19, 67, 158].

It is the case that much social media data does not have a location (GPS, tag) attached, e.g. users may decide to turn off their location-based service on their phone. To leverage the aggregated effects of spatio-temporal information, a sufficient amount of GPS tagged data needs to be generated in time periods sufficiently close to each other. This could be a problem in validating the method more generally, since sufficient amounts of spatio-temporal data may be limited. For example, some urban traffic issues may not be correlated to nearby social media data because there is simply not enough data. This occurs especially in sparsely populated areas, e.g. rural locations. However, in recent years, there are an increasing amount of geo-location based mobile apps such as taxi/hotel booking apps, navigation apps and apps for finding nearby services/stores. The platform and methods purposed in this thesis have been designed to be generic so that all of the data from these applications can be used for urban traffic analysis.

Patterns in social media, when used for spatio-temporal analysis of traffic,
needs to be validated. In this work, we consider the Sydney Coordinated Adaptive Traffic System (SCATS — https://www.scats.com.au) and the official road network data for Australia from the Public Sector Mapping Agency (PSMA — www.psma.com.au).

1.5.2 Official Traffic Resources

SCATS is a fully adaptive urban traffic control system that is used in 27 countries for capturing and (potentially) augmenting the decision-making related to traffic flows. In Australia, the SCATS system utilizes a wide coverage sensor network on major roads and junctions to monitor (count) the traffic flow status. It is an official public urban traffic data source provided by the Australian government agency. In Victoria, the SCATS data is supported by VicRoads. SCATS data can be directly used for detecting traffic issues. However it has its limitations:

- **Expensive**: The installation and maintenance of the SCATS sensor network is expensive, and devices are only deployed at major roads and predominantly in cities.

- **Lack of real-time support**: There is no live stream of the SCATS data open to the public. Information about urban traffic is typically, very time-sensitive. It is almost impossible to provide real-time traffic analysis by using historical SCATS data alone.

In this thesis, the SCATS volume data was collected from the Victorian Government Data Directory (https://www.data.vic.gov.au). It is used as the official source of real traffic data in the case studies including exploring the potential of using social media data as a proxy for traffic status agencies. The SCATS volume data provides structured data with fields such as the UID of the sensor site, the name of the street section, the volume of passing vehicles every 15 minutes and the timestamps for the signals. There is a supportive data set to the SCATS
volume data that provides geographic information, e.g. latitudes and longitudes of the sensor sites. Since the focus of this research is to explore spatio-temporal relationships between social media and official traffic data, these two data sets need to be converted and joined into a single geographic data form.

Geographic data or geodata is a kind of standard structured data defined by ISO/TC 211 (https://committee.iso.org/home/tc211/). GPS tagged social media data can be transformed into geodata since it often contains the precise information of the geo-location and time. Since the SCATS volume data and its supportive data set share the same UID of traffic sensor/device sites, the SCATS volume data set can be converted into a structured geographic data set with shapes of site points and road lines through a spatial join operation.

PSMA Australia is a company that offers sustainable access to authoritative national location data to industry, government, and academia. The PSMA data used in this thesis are available and collected from the Australian Urban Research Infrastructure Network (AURIN — https://aurin.org.au). AURIN is a collaborative platform for researchers in Australia. It provides a one-stop online workbench with access to more than 4800 multi-disciplinary datasets, from over 100 different (definitive) data agencies with a range of analytical/visualisation tools. The PSMA data used in this thesis’ work is PSMA Road data. This data provides accurate geographic information of the complete road network of Australia. This data is also used to identify the social media data created near/on the street network.

1.5.3 Research Hypothesis

The central hypotheses of this thesis is that:

- “it is possible to develop a big data processing platform for urban transport that targets the specific needs, demands and challenges of big data including the volume, velocity, and variety of urban traffic data”;
• “it is possible to find certain ‘cheaper’ real-time accessible data sources that can be used as proxies for real-time urban traffic”.

These hypotheses give rise to several further questions and tasks that need to be addressed:

• What are the specific needs of traffic analytics when dealing with Big Data and what kind of technologies can satisfy the specific traffic data capture, processing, analysis and visualisation requirements?

• Can we collect social media data just from the road network and use this or clusters of such data as a cheaper proxy for official traffic data. If this is possible, then can we identify and develop new algorithms/methods needed to address the specific challenges around the analysis of such heterogeneous real-time spatio-temporal traffic data?

1.6 Thesis Structure

The structure of this thesis is organized as follows:

In Chapter 2, related works are reviewed across four related areas. The first topic reviews work on urban traffic analysis methods. The second topic reviews work in the area of big data and Cloud computing. The third topic reviews work specifically on social media analyses including using the social media for analyzing urban traffic. Given their importance to the challenges that are raised, the last topic reviews clustering algorithms and how they can be applied to (freely available) social media data, with a specific focus on its use as a proxy for traffic data.

Chapter 3 provides an overview of the data used in this thesis and how it was collected including Twitter data, SCATS Traffic Volume data and PSMA Road Network data.
Chapter 4 focuses on targeted harvesting of social media data. In particular, the chapter shows how data can be collected just from the road network and used/processed for traffic analysis. This ability to collect (free) social network data from the road network is unique and the first contribution of this thesis. This forms the data collection system that is benchmarked with more official (SCATS) data.

Chapter 5 presents a novel dedicated platform targeted specifically to the needs and demands of traffic data. Data such as social media is typically created in urban areas are high in velocity and volume. This is often referred to as the data deluge [101]. Therefore, Big Data solutions need to scale to tackle such challenges. Similarly, traffic related data and associated analysis have both spatial and temporal challenges. To tackle these challenges, efficient spatio-temporal capabilities are required. This chapter covers the design of such a platform — SMASH. A benchmark of this platform is provided at the end of this chapter. This is the second contribution of this thesis.

Chapter 6 proposes a novel density-based algorithm for clustering spatio-temporal big data streams in real-time. The motivation of this chapter is to use clusters of social media as detected features and analyze the surrounding traffic status as given by the official SCATS data. The work offers a unique algorithm realised as an extension to the widely available and adopted clustering solution DBSCAN to meet the requirements for real-time, parallelised clustering. This is the third contribution of this thesis.

In Chapter 7 case studies on the SMASH platform are presented. The chapter considers how social media data such as Twitter and Instagram can be used with more formal data such as the PSMA road network data and SCATS data to identify clusters of activity on the road network reflecting traffic issues, e.g. congestion. In particular, several case studies are explored:

- whether it is possible to find nearby traffic abnormalities when a social media abnormality is detected?
• what are the odds and risk ratio from traffic abnormalities to social media abnormalities?

• does the occurrence of abnormal social-media clusters have an overlap with the abnormal traffic volumes, i.e. an abnormality of social media is represented when the nearby spatio-temporal density exceeds given thresholds.

• is the abnormality of traffic volume defined by the average traffic flow values and their standard deviations?

Chapter 8 concludes the work as a whole and identifies areas of potential future work.
This chapter reviews related research and state-of-the-art technologies related to the research described in this thesis. This chapter thus provides the research context to understand the contribution of the work as a whole.

2.1 Urban Traffic Challenges and Research Context

In this section, existing urban traffic analysis methods are reviewed. The requirements and challenges of these methods form the core objectives in the design and realisation of a new Cloud-based platform for urban traffic analytics that represents a key contribution of this thesis.

Urban transport and traffic more generally represent an area that many re-
searchers have explored from a variety of perspectives [106] [72] [200]. One common demand is for traffic forecasting [106, 199, 201]. Precise prediction of traffic congestion can help planners make informed decisions on traffic flows and thus lead to savings of travel time, fuel and reduce the overall environmental impact.

There are many different approaches to traffic forecasting. A typical class of methods is modeling traffic data for subsequent traffic flow detection and forecasting. Such traffic models can be classified into two groups. The first group is based on mathematical models such as auto-regressive moving averages [86] and Kalman filtering [184]. The calculations for building such models are simple and fast. However, they are limited by the extent that they capture the actual complexity, uncertainty, and changes in actual traffic flows.

Another group of approaches is knowledge-based intelligent models, which include support for vector-based regression models [31], artificial neural network models [201] and their various extensions [106, 107, 199]. These methods usually involve applying some form of machine learning algorithm on a given set of training data. They are much more computationally complex compared to the first group of models. Since the models are geared toward building forecasting applications, the speed of processing is often a key requirement. One challenge in applying these models is the demand for large-scale historical training data that can often be required since, as discussed above in Chapter 1, a single machine cannot store and process voluminous amounts of data efficiently (or at all!). For real-time forecasting, a high throughput solution is needed to tackle the speed of the incoming data while continuously updating the model. Performance and scalability are thus key requirements that underpin such methods. A scalable Cloud-based platform offering an efficient computation engine is thus highly desirable.

Transport data is often collected by sensors on vehicles or on road-networks where location and temporal information used for generating traffic models is
captured. As noted in defining the 3rd V of Big Data (in Section 2.2) however, there are a variety of types of data. For example, [200] extracts vehicles and their motion estimation from airborne LiDAR data/images. They apply image-processing algorithms based on an object-based image analysis framework targeted to LiDAR data for recognition of vehicles from the LiDAR point-cloud data. The motion of each vehicle is estimated by analyzing its shape, e.g. a stretched parallelogram with a tilt angle can be used to indicate the vehicle motion. With the deployment of satellite and airborne LiDAR, this method is powerful for analyzing traffic flows since it captures vehicles directly from images taken in the real world and potentially through drone technology in real time. However, there are challenges in adopting this method for traffic flow forecasting due to the demands incurred when processing large-scale image data at city-scales and in real-time. This problem is exacerbated since high-resolution point clouds are usually huge. These can be of the order of many Terabytes when used to cover larger scale areas, e.g. at the city level. Applying analysis algorithms on such data sizes can thus take considerable time and require significant computational resources. In addition, traffic vector data that are captured from the images need to be stored and indexed for further analysis.

Social media data is another class of data, which can be used for traffic detection. Through the use of mobile devices with location-based services, social media data can often reflect an individual’s daily movements and events including their commuting routes in cities. Token extraction and sentiment analysis are common approaches in social media analysis [186]. With the GPS information commonly associated with social-media, Twitter data can be clustered in time and space on the road network.  This can then be used to detect traffic accident blackspots [161] or explore the relationship between such clusters to actual traffic issues, e.g.

\[\text{Y. Gong, F. Deng, and R.O. Sinnott, Identification of (near) Real-time Traffic Congestion in the Cities of Australia through Twitter, Understanding the City with Urban Informatics, in Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM), Melbourne, Australia, October 2015.}\]
the impact on urban traffic caused by public events.

In Australia, many transport agencies also collect traffic data through targeted data capture systems, e.g. the Sydney Coordinated Adaptive Traffic System (SCATS) that is used to capture individual vehicle location information. SCATS has been deployed as a large-scale sensor network on the road network across Australia to capture the volume and motion information of vehicles on each lane of the road network. At present, there are over 11,000 SCATS locations across Victoria and the processing of all this data in real time has data bottlenecks.

Processing and monitoring urban traffic data on the Cloud is not a new idea. For example, [108] use Cloud computing to support intensive Floating Car Data (FCD) models [141] for traffic monitoring. FCD is a method to determine the traffic speed on the road network by analyzing the GPS trajectory data of vehicles. They utilize Apache Hadoop and HBase to store and index the spatial data. Hadoop MapReduce augmented with Message Passing Interface (MPI)-based capabilities [135] have also been used for traffic parameter computation, e.g. vehicle speed. The results of such raw data processing can be exported to Advanced Traveller Information System [65] and/or Advanced Traffic Management Systems [72] for subsequent traffic management and control. The infrastructure for this is dedicated to FCD data; however, it is limited in the kinds of traffic data it supports and the algorithms that can be run to analyze such data. Nevertheless, their framework provides a working example supporting traffic analytics on Cloud. Their use of the Hadoop Distributed File System (HDFS) provides a distributed fault-tolerant file system. HBase and MapReduce provide an established approach for querying distributed datasets. MPI-based solutions are still the dominant technology for efficient and performance-oriented parallel computing with many years of support for complex computation capabilities on large-scale supercomputing infrastructures.
2.2 Big Data and Cloud Computing

With the rapid growth of portable computing devices, sensor networks and Internet of Things (IoTs), an unprecedented amount of structured, semi-structured and unstructured data are being generated at every moment all around our daily lives. In 2017, about 2.3 trillion gigabytes of data were created every day. As noted in Chapter 1, by 2020 the volume of generated data every day is estimated to exceed 43 trillion gigabytes. Such volumes of data are now commonly referred to as “Big Data”. This data has several properties that make them challenging when used for research purposes. These are often referred to as the Vs:

- **high Volume** — the size of the data is one of the key challenges to be tackled. Data of such sizes cannot reside in a single server or database, and algorithms that once worked will not scale when tackling such huge data volumes. With the growing use of Web, IoT and the application in smart cities, the generation of such data is likely to continue to increase for the foreseeable future. This raises questions to industries, researchers and society more generally: How can we use these large-scale data sets? Recently, much research such as [37, 52, 140, 145] used such Big Data collected from facilities around Smart Cities to analyze urban traffic and human mobility. While such methodologies showed that a lot of large-scale data sets could be used for urban traffic analytics, it was identified that there was a lack of generic infrastructure for adapting the methods to production environments and hence an inability to tackle real-time Big Data challenges in producing real world analytical results on the fly. A generic and scalable Cloud-based infrastructure for realistic urban traffic analytics is still unsolved.

- **high Velocity** — the speed at which potentially vast volumes of data are being generated, transmitted and analyzed. The social media data explored in this thesis is a typical example of a potentially high velocity data source. Nearly 8,000 new tweets are created in every second of every day. This
requires an infrastructure to deal with such high velocity data using a scalable architecture (infrastructure) augmented with parallel stream analytic processing capabilities.

- **high Variety** — Big data can involve both structured data (e.g. tabular data) and unstructured data (e.g. messages, documents, photos). This raises a common challenge in storing and manipulating the data since they cannot be mapped into a pre-designed table as is the case when storing structured data into a relational database. In this thesis, the social media data used for traffic analytics are one example of unstructured data or semi-structured data (data cannot reside in a single relational table), while the SCATS traffic volume data and PSMA road data are structured data. A data processing and storage layer is required to tackle such data heterogeneity and support the storage, indexing and subsequent querying of this and potentially other kinds of traffic-related data.

In addition to the 3Vs of Big Data, several other Vs that are now considered part of the Big Data challenge including veracity [162], volatility [136], visualization [95], variability [172]. We consider especially **Veracity**, **Volatility** and **Value** because they are central to the use of Big Data within urban contexts and especially for urban traffic analysis:

- **Veracity** — is related to the origin and accuracy of data so that it can be used to ensure the quality of analytic outcomes. It often refers to the accuracy of measurements in records from definitive agencies and their relation to reality. In the context of social media data as discussed in Section 1.5.3, people can send random tweets in their posts and hence the accuracy of social media content (text) is often ambiguous and with low veracity, i.e. people can post fake information either purposefully or by being mislead). However the accuracy of the spatio-temporal information reported in social media through smartphones is much more reliable and has a higher level of
veracity. This fact is used when utilizing this information for urban traffic analysis in this thesis. The time/date values associated with social media data is recorded and synchronized using the Coordinated Universal Time (UTC) through the Network Time Protocol (NTP) [122]. NTP can usually maintain time to within tens of milliseconds over the public Internet. In the context of urban traffic issue, a few milliseconds difference can largely be ignored. The geo-location of social media data (if the user turns their GPS/geo-tagging on) is measured by the location service of the user device and its accuracy can vary somewhat depending on different situations. The details of the geo-location accuracy of the social media data used in this thesis is discussed in Chapter 3.

- **V**olatility — is another challenge for Big Data analytics particularly with regards to real-time data streams. In the case of real-time urban traffic analytics using social media data, spikes in data can occur, e.g. during rush hour or when events occur. Coping with such volatility can be essential for some domains, e.g. to ensure that systems do not become overloaded or starved of data.

- **V**alue — is used to refer to the potential insights that can be extracted from the data itself. For example, social media data with geo-tags are considered to be more valuable than the non geo-tagged data when conducting location-based analysis. The information posted by official accounts, e.g. @VicRoads, are typically more trustworthy than others. Social media data with hashtags (e.g., #traffic) can be used to extract knowledge related to specific topics/events. When processing large volumes of data which arrives at high velocity, it can be important to put focus on the more valuable data and scan/skip less valuable data depending on the analysis purpose and methods.

Big Data analytics crosses many research fields and has attracted the attention
of researchers, industry and government alike. Information or knowledge can be extracted from Big Data in many ways to expose hidden patterns existing within such massive data sets. Researchers, decision makers, and problem solvers now view Big Data applications as a key tool to revolutionize many sectors and industries including health, retail, advertising, education, city planning, and public administration amongst many others [26, 36, 96, 144, 150, 191].

In this thesis, the challenges of Big Data and support for associated data analytics can be broken down into handling the challenge of the 6Vs. Cloud Computing is one of the predominant solutions for Big Data that is especially suited for tackling aspects these 6Vs. The basic premise of Cloud computing is to store, manage and process potentially infinite data on external, web-accessible scalable platforms, or at least give the perspective that the resources available are infinite. In reality of course, this is never the case. The agility and affordability of Cloud computing make it an ideal combination to tackle such issues with Big Data. This section includes a review of Big Data and Cloud computing including the core background, broadly accepted models and current challenges.

2.2.1 Background and Architecture of Big Data Systems

As discussed in Chapter 1, the Internet is now pervasive across all aspects of our daily life. Whether it is people browsing the web for information, using social media applications, or businesses and governments using such platforms to provide services, there few areas have not been touched by the digital revolution. With the growth of smartphones and mobile network devices now making it easier than ever for people to access online services and resources, the information (data) that is now created covers all aspects of life at a scale that has never been seen before.

Such huge amounts of data and the creation of more data is likely to continue into the foreseeable future [17]. In this context, data analysis has become an essen-
tial tool across many fields: from engineering, city planning, product marketing, to health and many other areas. Unlike much web-based data, mobile data often has a unique characteristic: it includes location information. This information can be used to optimize and personalize mobile services, e.g. route finding or identifying the best restaurant in a given area. The fact that the mobile platforms are now global, the location data that is generated has far-reaching possibilities and dangers, e.g. erosion of privacy [182].

Such data can be extremely large (petabytes, exabytes, zettabytes) and include complex and fluid data structures. Traditional data processing on a single machine cannot tackle such sizes of data. Cloud computing offers features that can meet some of the needs of Big Data processing systems. The vision (marketing) is to offer infinite and scalable computing power and data storage, where the end users do not need to concern themselves with the lower level details of managing large scale computational infrastructures. In industry, Cloud providers such as Amazon, Google and Microsoft now provide network-accessible Cloud computing platforms and a multitude of related tools for storing, analyzing and interrogating the data. Users (individuals or businesses) are charged by the time and amount of computing resources that are leased [48].

Algorithms and many open sourced software solutions have been developed for handling certain aspects of Big Data. Distributed file systems like Hadoop [177] and the Hadoop File Storage System (HDFS) [159] provide capabilities that can tackle some aspects of big data, e.g. storing and querying massive scale data in parallel across distributed computational nodes. Batch-processing models, such as MapReduce [47] support data coordination, combination, and processing using multiple computational sources.

Big data technologies have been shown to tackle some other aspects of the 6Vs [62]. The selection of particular Big Data technologies in building Big Data applications depends on the data processing needs. For example, what is the speed of creation of the data? What kind of data analysis needs to be performed?
What is the nature of the source data and how many data sources are involved? Are there sensitivities around the data, e.g. are privacy or Intellectual Property considerations important to consider. The answer to these kinds of questions can result in diverse technical solutions.

2.2.2 Cloud Computing and Big Data-as-a-Service

Cloud computing is a broad concept. In general, the most widely accepted definition [120] is that it is ‘a pool of network-accessible computation resources which can be rented to multiple remote users.’ The computation resource is also a broad concept and can include hardware resources like CPU, RAM, storage through to software resources like databases and other applications. Historically, Cloud services were classified into one of three Cloud models:

- Software as a Service (SaaS);
- Platform as a Service (PaaS), and
- Infrastructure as a Service (IaaS).

In some aspects, SaaS is very similar to (older) thin-client models of software provisioning, where clients provide the point of access to software running at some remote location (potentially on the Cloud). In this case, the remote resource provider manages the software and its deployment for users. SaaS probably is the most familiar form of Cloud service to users who do not come from a research or computer science background. There are now many robust and widely adopted SaaS applications such as Google Apps, communication tools like Slack and storage solutions like Dropbox. Users interact with remote services provided by others. They can personalise and use these for their own needs and demands without needing to understand the lower level information related to the infrastructure that they run upon. Use of SaaS applications reduces the cost of software
ownership by removing the need for technical staff to install, manage, and up-
grade software. This can also reduce the cost of licensing software [188]. It has
been identified however that SaaS is inflexible for more complex applications and
especially data processing applications which involve multiple diverse sources of
data since the data continually evolves and the analytics must evolve around this.
Thus, while Dropbox provides a generic storage solution adopted by many, the
data analysis for the data stored in Dropbox is not undertaken. Many domains,
e.g. bioinformatics, continually develop new tools and methodologies based on
the evolving data sets and the associated research landscape and knowledge [112].

PaaS functions at a lower level than SaaS, typically providing a targeted
platform on which software can be developed and deployed. PaaS gives clients
an environment in which the operating system and server software, as well as
the underlying server hardware and network infrastructure, are all taken care of.
Unlike SaaS, PaaS leaves the users free to deploy their applications and services
with their specific configurations. As with most Cloud services, PaaS is built on
top of virtualization technology. Clients can apply for resources as needed and
scale as demand grows. Examples of PaaS providers include Apache Stratos,
Heroku, Google App Engine and AWS Elastic Beanstalk [104].

Compared to SaaS, PaaS offers more freedom to specialized users by allowing
them to develop and deploy their software on Cloud, however some constraints
still exist. The service provided by PaaS is a pre-defined platform to which
specified development and deployments tools are bound. Developers are not
completely free to choose the technologies or program languages and they have
to follow a set of rules to make their software artifacts reside in the PaaS. With
the advent of Container technology like Docker [50], the restrictions imposed by
PaaS are now appeased. Container technologies allow developers to customize
platforms by easily describing their software components and their associated
dependencies and tools with support to package them into container images.
A container image built in this manner has everything it needs to launch the
software on any Cloud-based instance (server). Container-oriented services on the Cloud are now called Container-as-a-Service (CaaS), which can be seen as a special form of PaaS [133]. CaaS allows users to deploy and manage their packaged software containers using standard managed Cloud-based VMs. A virtual machine can run one or multiple containers by using operating-system-level (container) virtualization provided by the container daemon. The container daemon offers isolated user space (process-level isolation) to each guest container.

Unlike hardware-level virtualization used for creating VMs, operating-system-level virtualization is more lightweight. Containers on the same host share the same system kernel from the host in comparison to each VM running in its own system kernel, which usually requires more host memory. From the viewpoint of Cloud usage, CaaS can be used for maximizing the usage of Cloud VMs, and hence it is more cost-effective than paying for a VM for software deployment. However, security issues [115] and resource competition issues [100] (due to sharing OS kernel) need to be taken into consideration when considering use of CaaS. Examples of CaaS providers include Amazon ECS [7], Google GKE [73], Azure AKS [121] and Pivotal PKS [142].

As the lowest Cloud level, IaaS can be seen as a fundamental building block for all Cloud services. It is the most flexible Cloud computing model and allows for automated deployment of servers, processing power, storage, and networking on demand. IaaS clients have more control over their infrastructure than users of PaaS or SaaS services. They can manage their Cloud resources via targeted dashboards or APIs, e.g. through OpenStack interfaces to core components. The primary use of IaaS includes the actual development and deployment of PaaS and SaaS platforms. Many Cloud providers now offer IaaS, including Amazon EC2, Windows Azure, Rackspace and Google Compute Engine [104].

As discussed, Cloud computing aims to provide a cost-effective way to build virtual data centers offering data processing and storage capabilities, without users having to take care of the physical maintenance and management of the
underpinning (physical) resources. When we link Big Data to Cloud computing, a new term of Cloud business models emerges — Big Data-as-a-Service (BDaaS).

BDaaS is a term typically used to refer to services offering storage and management of very large data sets and using data processing capabilities of external providers. A given instance of a BDaaS platform provides a combination of a targeted software stack and associated Cloud infrastructure. In many cases, a distributed compute and storage technology (e.g. Hadoop) at the platform level forms the core of the data part of BDaaS. A BDaaS solution typically includes a PaaS layer and potentially a SaaS and/or IaaS layer. This leaves four possible combinations/types for BDaaS as shown in Figure 2.1:

- **PaaS only / Core BDaaS**: provides minimal deployment comprising the BDaaS core and basic tooling. One manifestation of this is Hadoop with YARN [177] and HDFS and one or more popular services like Hive [171]. Amazon Web Service’s Elastic MapReduce (EMR) is one of the most prominent examples of this model.

- **IaaS and PaaS / Performance-oriented BDaaS**: provides an extension of the BDaaS core to include an optimized infrastructure. This allows to reduce some overheads of virtualization and specifically build hardware servers and networks that cater to Hadoop’s performance needs. Altiscale [3] is an example of this type of BDaaS. Clients can outsource their infrastructure/platform needs and management around Hadoop to Altiscale and focus on supporting a flexible SaaS platform.

- **PaaS and SaaS / Feature-oriented BDaaS**: provide an extension of BDaaS core to include features beyond the common Hadoop ecosystem offerings. Qubole [143] is one example of a Feature-oriented BDaaS, providing web and programming interfaces as well as database adapters and technologies as part of an associated SaaS layer. Both Core BDaaS and Feature BDaaS are
independent of the IaaS provider thereby making them very flexible and hence suited for platform migration and adaptation.

- **IaaS and PaaS and SaaS / Integrated BDaaS:** provide a fully vertically integrated BDaaS that combines the performance and feature benefits of the previous two BDaaS. At present no publicly accessible Integrated BDaaS have been made available.

![Figure 2.1: Taxonomy of Big Data as a Service (BDaaS) offerings in the Cloud](image)

### 2.2.3 Challenges and Open Research Issues

With the constantly growing need for enterprise analytics using Big Data and the seemingly, never-ending data generated by e-Commerce and IoTs, and others, many new challenges and open research issues arise. These can include challenges related to storage, capture, processing, filtering, analysis, curation, search, sharing, visualization, querying and privacy of such large volumes of data. These issues are categorized and elaborated in [39]. They include:
• **Data storage and management:** since big data depend on extensive storage capacity and the fact that data volumes are growing exponentially, current data management systems cannot continually be assumed to satisfy the needs of big data. Furthermore, existing algorithms are often not able to process and store data effectively — in part due to the continuous expansion and heterogeneity of data.

• **Data transmission:** transferring large sizes of data is a challenge to overcome on the Cloud. Reducing the size of data before transmission can aid this process somewhat, but the data sizes are often still prohibitive when data transfer is required. As a result, for many scenarios the computational processing takes place where the data is created and subsequently stored [132].

• **Data processing and analysis:** query response times represent a significant issue for many Big Data scenarios especially for real-time applications. Scalable services are required to reduce the response time, e.g. extend the platform with more services as/when need demands. However, [39] also identifies that the learning curve of big data programming for data analysis is still a challenge for many people. More flexible and interactive analytic engines could be helpful to simplify the development and support of big data applications.

• **Data privacy and security:** since data and associated analytics are hosted and undertaken remotely by third-party services, security issues are typically major considerations for Cloud computing and big data application domains. The current technologies used in data security are mainly focused on static data-oriented scenarios. Big data analytics can involve dynamic changes in data (on-the-fly), which gives rise to security and privacy concerns. For example, certain data can be re-identified by linking with other data sets [167] or data can burst from private to public Clouds, but the
policies for what data can burst from private to public Clouds remains a challenge [198]. Privacy-preservation when data mining without exposing sensitive personal information is challenging and a research field actively being investigated [2].

Figure 2.2: Available Cloud services on the AWS Dashboard
2.2.4 Feature-oriented BDaaS for Urban Traffic Analytics

Although many public Cloud providers have offered a variety of BDaaS services, there is no integrated Feature-oriented BDaaS that meets all the needs of Urban Traffic Analytics using geospatial Big Data. Figure 2.2 shows a variety of available services on the AWS dashboard with many offering BDaaS capabilities. Several popular Big Data solutions are now offered as standard service. For example, Amazon EMR provides Apache Hadoop and its ecosystem (including Apache Spark, HBase, Presto, and Flink) as a service. They are generic PaaS solutions for analyzing Big Data, however they often cannot be used to solve the specific problems associated with urban traffic analytics. There are many reasons for this. Some examples include:

- **Lack of support for geospatial Big Data.** Thus spatial indexing software is required for managing urban traffic-related data, however existing BDaaS are not specialized for storing geospatial data. [196] provides an example of using EMR for analyzing large amounts of spatial data. The spatial information is stored in plain text and leverages the MapReduce engine to apply geospatial operations at run time. This process is much less efficient compared to analyzing the geospatial data in a local GIS platform. Additional software/technologies are needed to provide GIS like services, however the pre-defined PaaS makes it hard to add specialized software on demand.

- **Difficulty in customizing the environment:** BDaaS like Amazon EMR involves many software components. The Cloud provider is often responsible for solving the conflicts e.g. dependency conflicts and resource competition between those components, which is one of the major advantages of BDaaS. However there can still be conflicts between the user application and the underpinning software from the BDaaS platform, which forces the developer to follow rules of the BDaaS. This is somewhat similar to the limitations of traditional PaaS where developers are not free to choose the technologies for
the pre-defined platforms. When conflicts exist between third-party tools, e.g. GIS tools used for traffic data analytics, and the BDaaS platform, then this can be difficult (impossible) to resolve.

- **Difficulty with migration**: urban traffic analytics involves real-time data processing which in turn often requires the Cloud facility to be physically close to the data source to reduce the impact of network delays. Moreover, there can be sensitive data which need to be held in private Clouds and public Clouds used for processing of non-sensitive data (Hybrid Cloud Computing). These requirements are difficult to support with existing BDaaS platforms.

Due to the above limitations, we propose a new Feature-oriented BDaaS platform to analyze urban-traffic-related Big Data in this thesis. By using Container technologies and a dedicated (targeted) software stack, we show how this platform can be deployed/scaled on any private/public/hybrid Cloud infrastructure on demand and provide capabilities needed for traffic data and its associated real-time spatial characteristics.
2.3 Social Media data Analysis and Related Work

One of the research focus areas of this thesis is to explore whether social media data can be used for urban traffic analysis/monitoring by leveraging the spatio-temporal information available in such data. This section reviews some of the key work on social-media analysis and especially those in the transport domain.

With the uptake of social media and the ability to capture location-based information, it is not surprising that studies have witnessed a ‘spatial turn’ during the past few years [124]. Social media, led by MySpace, Facebook, Twitter, LinkedIn, Flickr amongst many other offerings, are based on Web 2.0 technologies. Increasingly such systems provide location-based information that has moved social media from cyberspace to real place [165]. The volume, velocity, variety, and veracity of social media data also mean that it possesses many of the typical characteristics of big data [116] and is driving a range of research agendas in data science [82]. The tools to analyze and understand big data are increasingly demanded, especially tools that allow real-time patterns to be extracted from noisy and sometimes-contradictory data.

A range of works on social media data analysis has been undertaken. Sentiment analysis is one of the most extensively researched areas with over 7000 related articles in the last few years [61]. The most popular approaches for sentiment analysis are: subjective lexicon where a list of words labeled as positive, negative or neutral is given; the N-gram model where a group of N words is given as a training data set, and machine learning-based approaches where classification is performed using a set of features extracted from the text [94]. [99] investigate micro-blogging and its use for classification of Twitter data. They used supervised machine learning systems and included three different corpora for training data sets: emoticons, hashtags, and manually labeled data. Since microblogging data is often terse and abbreviated, the features and techniques used in other natural language processing (NLP) approaches are often not applicable
and/or accurate.

Twitter has been used to explore a range of scenarios and application domains. [128] explored movie rankings; [29, 126] explored disaster and emergency response systems using Twitter; [43] explored political sentiment analysis and the use of Twitter for prediction of elections; [211] explored the use of Twitter for sport and use of spectators for tracking and capturing moments from sporting events; [160] explored the use of social media data to better understand and track information related to pandemics and emerging infectious diseases; [69] explored the use of Twitter data for prediction of crime in US cities, while [197] explored eating habits of individuals based on photographic evidence of food/meals posted on Twitter. The majority of these systems are based upon extracting information from the contents of tweets (the tweet text) and applying language processing techniques for sentiment classification. Other users have explored real-time global trending information [119] and identification of relationships between individuals and organizations based around the follower relationships [33, 35].

With regards to transport and traffic-related research based on social media, various studies have followed similar approaches. [98] use natural language processing to identify targeted keywords in tweets and correlate them with transport events, e.g. accidents that may have taken place. They also analyze the confidence level of traffic information. An algorithm was proposed to estimate the event confidence level based on the timeliness of the information and how many people tweet about the same event in a similar time period.

Similar to the above study, [186] extracts traffic information from tweets using syntactic analysis and then classifies the results into two categories: point and line. Point data refers to a single accident/event on the road, e.g. a car crash, while line data refers to a traffic status along certain streets including a start point and end point on a map over a given time period. This can be used to identify congestion along roads for example. The most interesting part of their work that is related to the work described in this thesis is the line data classification. They
use a classification algorithm to build connections/relationships among traffic events identified from tweets and attempt to provide an explanation of what kind of real traffic events can be represented by those relationships. However, their work depends upon large collections of often non-relevant tweets. Ideally far more targeted harvesting of important and related tweets would be beneficial since natural language processing is imperfect and the resultant analyses will almost always include erroneous tweets, e.g. Tweets containing the terms “car accident” could relate to historical incidents, incidents in other locations, or indeed statements from films/books.

[90] built a system providing transportation information during a disaster in Japan (an earthquake). They mainly focus on tweets created by people during disasters and show how they can be used to help victims find evacuation routes through machine learning approaches. Their work on prediction services for events/accidents is relevant to the work described in this thesis.

For monitoring urban traffic status by analyzing the spatio-temporal distribution of social media data, as discussed in Chapter 1, clusters of social media are important to consider. If clusters of social media data are found to have a connection with urban traffic, real-time cluster detection can provide a proxy for the far more expensive mechanisms currently used to capture data such as SCATS. The following section reviews some of the more relevant (real-time) clustering algorithms.

### 2.4 Related Work on Clustering Algorithms

Clustering algorithms can be categorized into five main types [80]: *Partition-based*, *Hierarchical*, *Grid-based*, *Model-based* and *Density-based*. K-means [114] is a typical example of the *Partition-based* clustering category. It divides data into a fixed number of partitions according to the mean value of the data objects. *Hierarchical* clustering such as CURE [74] utilize hierarchical trees as data structures for
managing nested nodes of clusters. Similar to Hierarchical clustering, Grid-based cluster approaches like STING [183] and WaveCluster [154] utilize multiple-level grids as structures for data partitioning. Model-based approaches such as COBWEB [63] apply cluster modules to the dataset and identify best matches. Density-based clustering algorithms like DBSCAN [56] are based on the idea that data objects which form denser regions should be grouped into data clusters. DBSCAN has the ability to find arbitrarily shaped clusters with less predefined parameters compared to other clustering algorithms, hence is used and extended in this thesis. It is also noted that DBSCAN is widely adopted and a mature clustering solution.

There are a variety of extensions to DBSCAN. They can be classified into two main kinds: performance-optimized DBSCAN and application-optimized DBSCAN. The former aims at reducing the execution time for data clustering [81, 137, 179], whilst the latter focuses on adapting DBSCAN to different high-dimensional data structures required for specific application scenarios [55, 164, 190].

There have been many performance-oriented extensions of DBSCAN. For example, l-DBSCAN [179] proposes a method to reduce the size of datasets before running DBSCAN. It employs a graph-based hybrid clustering algorithm [34] to pre-generate a few candidates (approximate) clusters. Only the points in those clusters are then input into DBSCAN for final clustering. However, this two-phased clustering method has several major limitations. Firstly, two critical parameters are added for hybrid clustering. As the authors point out, unsuitable selection of these parameters can lead to inconsistent clustering results. Secondly, the pre-clustering phase is used for filtering out noise data, e.g., outliers in the data. If a highly skewed dataset is input, this phase can become useless and consume unnecessary computing resources. This extension also does not meet any of the real-time clustering requirements, but the idea of reducing or sampling all of the data in DBSCAN is meaningful and has been incorporated into RT-DBSCAN as
will be described in Chapter 6.

MR-DBSCAN [81] provides a parallel version of DBSCAN in a MapReduce manner [47]. The major contribution of this extension is that it provides a method to divide a large dataset into several partitions based on the data dimensions. Localized DBSCANS can subsequently be applied to each partition in parallel during a map phase. The results of each partition are then merged during a final reduce phase. For the overall cost, a partition-division phase is added into DBSCAN. A division method called Cost Balanced Partition is used to generate partitions with equal workloads. This parallel extension meets the requirements of scalable execution for handling large scale data sets, and the MapReduce approach makes it suitable for many popular big data analytics platforms like Hadoop MapReduce [132] and Apache Spark [77]. However, this extension does not meet many key requirements for real-time clustering as arise when dealing with traffic data. For example, it needs to traverse the whole dataset for parallel clustering which means that its execution time is still dependent on the size of the dataset. Thus, while MR-DBSCAN has good performance for batch-oriented data scenarios, it is not suitable for high velocity datasets that can be bursty in nature, which is a key Big Data challenge.

PDSDBSCAN [137] is another version of parallel DBSCAN. It employs disjoint-set data structures to break the sequential nature of DBSCAN so that parallel processing can be performed. Compared to MR-DBSCAN which performs in a master-slave mode, PDSDBSCAN is based on a decentralized architecture. The advantage of this method is that it offers a fully parallel version of DBSCAN clustering. One limitation is that this method involves a lot of inner data transmissions among the computation nodes. As such, the I/O performance is a critical factor to its performance. Similar to MR-DBSCAN, PDSDBSCAN is also impacted by data size and data velocity on the overall performance.

For application-oriented extensions to DBSCAN, we consider two examples that are closely related to our approach.
Stream data is often spatio-temporal in nature and comprised of time-stamped, geographic location information [55]. This can be, for instance, social media data, trajectory data, Internet of Things data amongst other possibilities. This raises a requirement for clustering data in time and space according to their spatio-temporal characteristics. [22] present a method for handling this requirement. They provide an example of clustering spatio-temporal data according to its non-spatial, spatial and associated temporal values. In addition, they propose the notion of a density factor for each cluster which is helpful to identify the density of clusters.

Incremental DBSCAN [57] is another extension of DBSCAN suitable for data mining as might be applied in data warehouses. It supports incremental updates when clustering by inserting new data and deleting old data. It provides controls over the size of the data being used. Old data are excluded from clustering processes based on a time-based threshold which can be specified by the user. This method meets the real-time requirements of tackling ever-increasing volumes of data. However, it only works for time-based clusters. The definition of old data is a critical factor in this algorithm. Essential information can be lost by dropping data if inappropriate thresholds are set. Although this method is designed for daily batch-oriented tasks, the idea of dropping old irrelevant data and inserting new data into existing clusters is essential when designing real-time, high velocity clustering solutions.

Apart from DBSCAN, there are many other density-based clustering algorithms such as OPTICS [11], DENCLUE [84] and CURD [113]. However, few of them focus on real-time clustering. There are several k-means based clustering algorithms designed for tacking real-time data streams such as [12] and [15]. They use a time-based moving window and support k-means clustering within each window. However, k-means clustering needs to pre-identify the number of clusters which is a major limitation when tackling real-time and potentially bursty data.

D-Stream [40] is a density-based clustering approach for handling real-time
streams. It maps input data into a grid, computes the density of each grid, and clusters the grids using a density-based algorithm. There are also several limitations to this method. It cannot handle data arriving in arbitrary order since it requires a sequence of data that is time-stamped and ordered chronologically. It does not provide a solution for parallel execution of real-time high velocity data clusters. It also cannot handle ever-increasing sizes of data. Clustering in separate grids offers advantages to reduce the workload in data traversal for each new input, however eventually grid cells and grids would become full of data and ultimately slow down the processing speed.

In summary, despite the many approaches that have been taken to support different clustering algorithms by different researchers, none of them meet the needs of real time clustering that typifies the needs and requirements for traffic-based analysis. In this thesis, we present a new DBSCAN-based clustering approach. This overcomes many of the issues and limitations related to both DBSCAN and the above mentioned systems when dealing with high volume and especially high velocity, streamed data where new clusters are formed and where older clusters evolve based on the points and the temporal aspects of the point-based information.

2.5 Conclusions

This chapter presents many of the related research works or technologies of Big Data, analytics of traffic data and social media. We introduce the idea of using real-time clustering to analyze the social media data and point out the 'blank spot' of existing density-based clustering algorithms for real-time requirements. These form the context and ultimately the motivation for this work.

We identified that existing BDaaS do not fully meet all requirements for analyzing traffic-related data. We are interested in whether geo-clusters of social media data can be used as proxies for detecting traffic issues. This is explored
in Chapter 4 where we explore whether it is possible to collect (harvest) and identify spatio-temporal clusters of social media just from the road network.

In Chapter 5, we present a novel architecture (SMASH) that addresses many Big Data issues associated with traffic data directly, that overcomes many of the issues of related technologies identified in this chapter.

Furthermore, given the limitations of density-based clustering algorithms in supporting parallel processing suited to real-time Big Data streams, in Chapter 6 we present RT-DBSCAN clustering. We describe the implementation of RT-DBSCAN and how it has been benchmarked on the SMASH platform to meet the requirements for clustering real-time social media streams.

Finally, we explore the correlation between official traffic data and social media in Chapter 7 to answer the hypothesis as to whether social media data use can be used as a proxy for more official traffic data.
This chapter provides a general overview of the data used in this thesis including Twitter data, SCATS Traffic Volume data and PSMA Road Network data. The specific features of the data sets including the data size and spatio-temporal aspects are introduced. The data collection (harvesting) systems are also introduced. This chapter thus provides the data context that is built upon throughout the rest of the thesis.
Social Media Data

Social media data is a widely used data source for urban research as discussed in Section 2.3. Although social media data is well known for its accessibility and *free* access, it is noted that social media data is not completely free when considering hardware/software requirements. Indeed there are data access fees for commercial or larger scale research usage.

For many scenarios, social media data consumers require different aspects of data, e.g., users connection graph, the time or location of posts, timeline activities of target social media users. Capturing all posts can result in voluminous amounts of data. Many social media data providers including Twitter, Instagram and Flickr have built their own Application Programming Interfaces (API) to provide standard data access services over the Internet to allow customers or the public to collect targeted social media data on demand. There are many software systems that now exist that can be used to collect (harvest) data from these APIs. Once harvested, databases are needed to index the collected data sets and underlying servers used to host the data collection and data storage systems. These represent hidden costs of purportedly *free* social media. Compared to many other urban research data sets which can be collected by sensor devices/networks, e.g., environment and transport data, social media data is still for many a much cheaper resource for urban research. Thus there is no need for sensors to be deployed and managed to collect data for example.

Twitter data is a key source of social media data used in this thesis. The following section covers a basic description of Twitter data (a tweet) including the systems used to collect it.

3.1.1 What is Twitter and Twitter data?

Twitter is an online news and social networking service where users can post short messages called *tweets*. The term tweeting is used to describe posting
tweets. This can be done by the public, with the hope that their messages are useful and interesting to other users who follow their account on Twitter, or by official agencies, e.g. weather warning systems or traffic updates. Such kinds of applications are also called *micro-blogging*. People can discover other people and organizations online and subscribe to their message channels by following the accounts on Twitter.

Twitter has a message size restriction with an upper limit of 280 characters. As such, tweets are brief and can include abbreviations and terse sentence structures. Users can effortlessly track hundreds or thousands of twitter accounts and read their post at a glance. There are no rules on structure or syntax of tweets (other than the upper limit on their size) and hence the message content can include ad hoc information with mis-spellings and erroneous content. It has wide adoption with 126 million daily users reported in Feb 2019 [152].

Twitter is free to sign-up to and easy to use as either a broadcaster (tweeter) or a receiver. It has both a web client and smart phone clients (i.e., Android Apps and iOS Apps) which allows users to post and read tweets anywhere with an internet connection. Figure 3.1 illustrates two screenshots of the Twitter app on a smart phone. The screenshot on the left side is the home page of the Twitter account which displays all the recent tweets from the followed Twitter accounts. All tweets are typically, publicly accessible/searchable - even without necessarily following other users. This is unlike Facebook which requires user to accept friend requests for example. The 'Follow' action is used to get targeted tweets from specific users. The screenshot on the right side shows the graphic user interface (GUI) used for tweeting. Users can post short messages with images, and importantly for this work, with their associated geo-location information using the location based service (GPS) on their phone.

Figure 3.2 shows the data structure and the meta information of a typical tweet. It structure is based on Javascript Object Notation (JSON) and tweets are JSON Objects. This is the specified data structure used by the official Twitter
Figure 3.1: Screenshots of the Twitter App — Home page and tweeting page

APIs. The most important attributes of a Tweet Object include:

- **created_at**: UTC time (date string) when the tweet was created;
- **id_str**: the string representation of the unique identifier for the tweet;
- **text**: the actual UTF-8 text of the message with an upper limit length of 280 characters;
- **entities**: the entities which can be parsed from the text of the tweet;

---

1https://developer.twitter.com/en/docs
• *user*: the owner/user object of the tweet, i.e. the tweeter, which includes the basic public information of the creator account;

• *geo-coordinates and place*: these are attributes used for presenting the location where the tweet was created. The geo-attribute is now deprecated and largely been replaced by a coordinates field. The coordinates field represents the geographic location of the Tweet as reported by the user or client application. The coordinates array is formatted with geoJSON² (longitude first, then latitude). When the place field is present, it indicates that the Tweet is associated to a known place, like a restaurant, a shopping center, etc.

²https://geojson.org/
Figure 3.2: An Example of a Typical Tweet in JSON format

```
{
    "created_at": "Sun Aug 25 04:40:10 +0000 2019",
    "id_str": "1165483976088892864",
    "text": "Night practice... https://t.co/LnLugv9rE",
    "entities": {
        "urls": [
            {
                "url": "https://t.co/LnLugv9rE",
                "expanded_url": "https://twitter.com/likat/status/1165483976088892864",
                "display_url": "t.co/LnLugv9rE",
                "index": 11
            }
        ],
        "hashtags": [
            {
                "text": "likat",
                "index": 12
            }
        ],
        "user_mentions": [
            {
                "screen_name": "likat",
                "index": 19
            }
        ],
        "urls": []
    },
    "id": 1165483976088892864,
    "user": {
        "id": 1165472159683209728,
        "name": "likat",
        "screen_name": "likat",
        "location": "Melbourne, Victoria",
        "created_at": "Sun Aug 25 01:51:17 +0000 2019"
    },
    "geo": null,
    "coordinates": null,
    "place": {
        "id": "0fc2a2d98d141000",
        "url": "https://api.twitter.com/1.1/geo/id/0fc2a2d98d141000.json",
        "place_type": "city",
        "name": "Melbourne Park",
        "full_name": "Melbourne Park",
        "country_code": "AU",
        "bounding_box": [
            [144.9035956764448, -37.82184745279805],
            [144.9010156764448, -37.82184745279805],
            [144.9035956764448, -37.82184745279805],
            [144.9010156764448, -37.82184745279805]
        ],
        "attributes": {}
    },
    "lang": "en"
}
```
In this thesis, the geo-location information is an essential feature that is used for analyzing urban traffic issues, however it is noted that not all tweets are geo-tagged. According to recent studies [117, 155], approximately 0.9% of tweets are now geo-tagged. About 0.29% of tweets have the exact pinpoint geo-location (i.e., the coordinates field is present) and about 0.68% of tweets are associated with known places (i.e., the place attribute field is present). Despite these low numbers, there are millions of tweets that occur on a daily basis and hence a significant number of tweets can be collected from the Twitter APIs to provide a rich resource for research. In this work we consider tweets that are made especially on the street network. The accuracy of the geo-location is thus an important factor to consider.

3.1.2 Discussion on the Geo-location Accuracy of Twitter Data

The geo-location tags of social media data are generated by user devices (i.e., smartphones, tablets). Therefore, the geo-location accuracy of social media data is based on the location service provided by such portable devices. Global Positioning System (GPS) location services are a well-known technology packaged into smartphones and tablets. It utilizes a constellation of 24 satellites orbiting the earth at an altitude of 12,000 miles [14]. GPS devices compute a device position by determining the distance between the GPS receiver using a minimum of four satellites. The distance is calculated according to the transmitted radio signals from the satellites to the GPS receivers. The real location is exposed by the intersection point of the suspected areas based on these distances. The process of this acquiring this positioning is also called Location Fix [176]. Hybrid location systems are also employed by modern smartphones and used Assisted GPS (A-GPS), which is a WiFi positioning and Cellular network positioning system [202, 204]. These technologies are typically combined to provide the location service. Users are also able to deactivate their cellular and/or wifi positioning without turning
off the location service. A-GPS is the most accurate of the three, whilst cellular positioning is the least accurate. Figure 3.3 illustrates an example of the accuracy of each of these location services, the blue discs in the figure indicate the estimated positional errors.

![Figure 3.3: Positioning modes of the 3G iPhone: (a) GPS position; (b) WiFi position; and (c) cellular position [207]. The blue disc indicates the estimated positional error.](image)

There are several device-independent factors that can affect the accuracy of GPS. The distance measured by the radio signals assumes the ideal case of radio transmission. However GPS signals can encounter differing conditions when travelling through the atmosphere which can affect the signal and therefore the GPS accuracy. The various GPS satellites can also affect the GPS accuracy. The more satellites that are used, the more accurate the GPS service will be. In addition to the number of used satellites, urban canyons can also cause multi-path effects, where the GPS signal bounces off buildings or other objects thereby reducing the accuracy [207]. Satellite signals are easily interfered with by non-line-of-sight waves and climatic conditions [203]. A-GPS is an improved version of GPS that can reduce the response time of GPS location establishment. It is assisted by data provided from the cellular network related to GPS satellites. This assistance can reduce the response time of GPS location establishment from minutes to
seconds [175]. High GPS accuracy needs a clear view of the sky to receive signals from as many satellites as possible and thereby reduce the influence on signal transmission, however many of the people in urban areas stay indoors and/or may have high-rise buildings nearby that block the satellites. As such, GPS alone cannot achieve high accuracy in such places and this is why support from WiFi and cellular network positioning is necessary [209]. These two technologies are widely used as indoor positioning systems [111]. WiFi positioning uses the geo-location of WiFi access points (WAPs) as lighthouses and locates devices by measuring the distance to multiple WAPs via the strength of WiFi signal [70, 83]. Similarly, the cellular network positioning uses the location of cell phone towers as lighthouses to locate mobile devices. The details of this technology is discussed in [18]. Although the accuracy of Wifi and cellular positioning are usually lower than the GPS positioning, a combination usage of these three independent radio-signal based systems allow to locate the device as accurately as possible in both outdoor and indoor environments. According to recent researches [207, 208]:

- The accuracy of A-GPS positioning in modern mobile phones can achieve an accuracy of ±8 meters (median horizontal error).
- Wifi positioning can achieve an accuracy of ±74 meters.
- Cellular network positioning can achieve an accuracy of ±600 meters.

From the statistical results above, the accuracy of A-GPS positioning can drop considerably in indoor environments, or even be impossible. Although the WiFi and cellular network positioning have lower accuracy (from ten to a hundred orders of magnitude), their signals are not restricted by the view of sky and urban canyons. For indoor environments especially, the signals from WAPs and cellular towers are highly available in urban areas [13] and hence be used to overcome the shortage of A-GPS in indoor environments.

From the Twitter perspective, when attaching geo-location information into a post via a smartphone or a tablet, the Twitter app calls the positioning API (i.e.
the hybrid location system) at the device system level for the location information. This system will attempt to use the A-GPS positioning first. If a position fix is not available from the A-GPS (e.g., it is in an indoor environment), the hybrid system switch to use Wifi positioning as a backup solution. The indoor availability rate of Wifi positioning is about 87.7% according to [207]. If the Wifi positioning cannot achieve a location fix (e.g., not enough geo-identified WAPs nearby), the cellular network positioning is used as a subsequent backup plan. The indoor availability rate of cellular network positioning is reported to be 98.5%.

Thus in conclusion, the A-GPS positioning can achieve an ideal high accuracy when it is available in outdoor environments and a few indoor environments. The WiFi and cellular positioning are able to obtain a position fix for most indoor environments where A-GPS is not possible although in this case their errors in accuracy can be significant. There are also many researchers that focus on combinations of these three technologies to achieve more reliable positioning accuracy. For example, using selective weighting schemes [203]; by using indoor/outdoor awareness methods [146], and by improving the hardware devices themselves [41]. In addition, there have been substantial efforts in recent years to improve indoor WiFi positioning [76, 180] and cellular positioning [204]. Understanding indoor human mobility is also being actively explored [209].

For urban traffic analysis in general, high accuracy of geo-location information is desirable. For many of the real traffic data like vehicle GPS tracks, the accuracy of this data can be ensured since they are collected in outdoor environments. In this thesis, the proposed idea of urban traffic analysis using social media data is based on the spatio-temporal clustering and using aggregated information as identifiers for potential traffic issues. As such, highly accurate positioning accuracy of any single data item (tweet) is not required since positioning information is aggregated along with selected parameters, e.g., the distance threshold to form a cluster. In the case study of Chapter 4, we choose 1,000 meters as the spatial threshold for clustering geo-tagged Twitter data and \( \pm 100 \text{meters} \) as the acceptable
error rate, which is far in excess of the geo-location accuracy achievable using A-GPS ±8 meters.

3.1.3 Collecting Twitter Data

Twitter provides a variety of API endpoints for querying and collecting tweets. The returned tweets data have the same structure as the JSON object shown in Figure 3.2. Standard APIs are free to access and use, however there are also enterprise APIs which charge for data access. These provide much more data than the free to access APIs. Before using these API services, it is necessary to apply for a developer account on Twitter through the Twitter Developer web portal 3. A plan or description of the data usage is required to process a given application together with some form of identity verification, e.g. email and mobile phone number. In addition, it is necessary to follow Twitter developer policies and agreements on access to and acceptable use of the Twitter data 4.

Once approved, a credential is provided that can subsequently be used for accessing the APIs. In this work we only consider the standard (free) API endpoints for collection of tweets. The complete documentation of the Twitter APIs is available on the official Twitter developer website 5.

Twitter offer two mains forms of API.

- REST API: this kind of API can be used for collecting historical tweet data. Two of the widely used endpoints are described here.

  - The Search API 6: This API endpoint is typically used to return a collection of relevant tweets matching a specified query. Such queries can be a combination of different types of restrictions, e.g. based

\[3\text{https://developer.twitter.com/en/apply-for-access}\]
\[5\text{https://developer.twitter.com/en/docs/api-reference-index}\]
on time, space, users, language uses or the topics in the text. One limitation of this API is that it restricts access to older (historic) data. The policy currently only allows tweets made in the last week.

- The Timeline API: This endpoint can be used to return a collection of the most recent Tweets posted by the target user. Importantly, this API can be used to access historic tweets from each user (with the currently policy allowing up to 3200 tweets). For many individuals this represents the complete number of tweets ever made.

- The Streaming (real-time) API: This API endpoint allows the developer to access real-time streams of tweets through long-lived HTTP connections. Public, real-time tweets that match one or more filter predicates are obtained through this API. The filters can be applied to many aspects including the user identity, keywords and location of tweets, e.g. only tweets sent in Melbourne.

In this thesis, the software solution built for collecting Twitter data are described in more detail Chapter 4. Developers can use different strategies to collect tweets via these APIs as long as they fit the query rate limitation imposed by Twitter. There are many existing software libraries to ease the development of Twitter data collection systems. Example of these include: Temboo, Twitter4J and Tweepy. Each of these software systems has extensive documentation and user guides, e.g. how to avoid rate limiting and hence being blocked by Twitter.

---

7https://developer.twitter.com/en/docs/tweets/timelines
10https://temboo.com/library/Library/Twitter/
12https://github.com/tweepy/tweepy
3.2 SCATS Traffic Volume Data

In this thesis, the SCATS traffic volume data is used as a trusted data source to present the real urban traffic status of Melbourne, Australia. The high-level background knowledge of this data resource is described in Section 5.5.1. This data set can be downloaded from the Victorian Government open data platform — DataVic. The name of this data set on the DataVic website is Traffic Signal Volume Data. Figure 3.4 shows the webpage of this dataset on DataVic. It is noted that the raw CSV file of the SCATS Traffic Volume data filtered by different years can be directly downloaded from this page.

![Figure 3.4: Web page of the Traffic Signal Volume Data on the DataVic Portal](image)

It is noted that this Traffic Signal Volume Data does not directly contain the geo-location information of the SCATS sensors which are installed with the traffic

---

signals. Rather, they only have a unique ID that is linked to the sensors. Therefore, it is necessary to get another data set that links this ID to the location where the traffic volumes is actually measured. Figure 3.5 shows the web page of this reference data (Traffic Signal data set) set on the DataVic website.

![Figure 3.5: Web page of the Traffic Signal dataset on the DataVic Portal](image)

After downloading these data sets, they need to be cleaned and merged before they can be used for subsequent data analytics. This pre-processing operation is discussed in more detail in Section 5.5.2.
3.3 PSMA Street Network Data

The PSMA Street Network data is the official geographic data of the street network in Australia. The background information of this data set is discussed in more detail in Section 4.1. This data set can be downloaded from the web platform of the Australian Urban Research Infrastructure Network\(^\text{14}\) (AURIN).

\[\text{Figure 3.6: Overview of the AURIN web interface}\]

Figure 3.6 shows an overview of the AURIN web interface. It has a large map viewer with several menus and tools on the right hand side, e.g. for selecting areas of interest and the data sets related to those areas. All of the data sets on the AURIN platform have some of spatial information, e.g. a point, a polygon, a graph or indeed as a data cube. There are at present over 5,000 data sets accessible

\(^{14}\text{https://aurin.org.au/}\)
from over 100 (definitive) data organisations crossing government, industry and academia. These providers cover many critical urban and regional data sets in Australia, e.g. crime data from VicPolice, census data from the Australian Bureau of Statistics (www.abs.gov.au). AURIN provides access to and use of the PSMA Street Network Data. All of the data is freely available in AURIN to any academic. The procedures for downloading the PSMA Street Network Data from AURIN are briefly described here:

- Select an area as the bounding box which covers the area of interest. The polygon in orange in Figure 3.6 is an example showing the bounding box for the Inner Melbourne area.

- Search and shop for the target data set through the search engine as demonstrated in Figure 3.7. This uses metadata provided by the data agency.

- Download the shopped data set by right clicking the data set in the Data panel. The data can be downloaded as a CSV file, JSON file or if it includes geometry as a Shapefile. For data that does not directly include geometry directly, AURIN provides tools to spatialise the dataset. Once spatialised, the data can be downloaded as a Shapefile.
Figure 3.7: Example of Shop PSMA Street Network Data on AURIN
This chapter covers explore the use of social media data for urban traffic analytics. As previously discussed in Chapter 1, social media data sent from mobile devices typically includes geospatial information. This offers possibilities to show spatial patterns or phenomenon across the population and society. As discussed in Section 2.3, we found the reliability of social media contents, e.g. the tweet text, is often unreliable. However, while the content of individual social media

---

cannot always be guaranteed, clusters of social media in space and time from independent users can increase the credibility and accuracy of information. In particular, in this Chapter we explore how social media can be used to identify spatio-temporal correlations with specific focus on use of such data as a real-time indicator of potential transport issues, e.g. congestion?

This assumption/question is driven by the fact that tweets can be sent on the road and street networks. It is thus reasonable to infer that congestion occurs when clusters of tweets in the same time and space are occurring (from different users), since the driver is presumably stationary at that time. There are some potential issues with this assumption, e.g. it may be a passenger in a moving vehicle tweeting, or it may be people on buses/trams, however statistically it is the case that many people travel on their own in cars and the vast majority of vehicles on the roads are cars, hence clusters of tweets (in space and time) are likely to be through car drivers. Whilst it is possible for drivers to make/receive phone calls without using their hands, e.g. using Bluetooth connections and using voice recognition in the cars, it is still primarily the case that social media use requires people to use their hands.

To explore the extent that social media data can be used as a proxy for traffic congestion, requires a method for harvesting social media data such as Twitter data from the street network. This Chapter presents a novel solution for collecting (harvesting) social media data just from the street network of Australia using official road network data from the Public Sector Mapping Agency (PSMA - https://www.psma.com.au). A case study examining the clustering of social media data on the street network is presented in Section 4.3 together with the big data infrastructure required for data aggregation and associated analytics. Section 4.4.2 describes the deployment of the data harvesting systems and the clustering processes on the NeCTAR Research Cloud (https://www.nectar.org.au).
4.1 Public Sector Mapping Agency Data

For harvesting social media data created on the street/road network, an official geographic dataset is needed to accurately describe the road network. The PSMA data represents the most accurate national geographic dataset for Australia. This data is regularly updated when new roads are built or indeed any urban features established. The PSMA offers a wide range of data products: housing and land-use information; features of interest (e.g. railways, hospitals, schools); a national Gazetteer (address geocoding system) amongst many others. The PSMA dataset used in this chapter is the PSMA road data. This contains the road network topology for Australia. This data is license protected but available for research use from the Australian Urban Research Infrastructure Network (AURIN) as discussed in Chapter 1. Figure 4.1 illustrates a typical tuple of the PSMA Road data displayed in the AURIN web client. This PSMA data entity in Figure 4.1 can be visualized as a segment/polyline of a street localized by a serial of endpoints coordinates. The data fields of this entity contain the specification of the road/street characteristics including its name, the number of lanes, width, speed, height, and weight limits, as well as the geographic topology of the street itself. The PSMA dataset for the Melbourne City area is around 14Mb in size.

4.2 Harvesting Tweets from the Road Network

To explore the extent that social media can be used as a proxy for traffic flows and/or congestion, it is necessary to collect significant amounts of data sent from individuals on the road network itself. Thus requires a novel data harvester to be established that serves the specific needs of social media data collection when used for traffic analytics.

Twitter supports two programmatic interfaces to access their data. These are realised as Representational State Transfer (REST)-based services. The services
Figure 4.1: PSMA Road Data example from AURIN

comprise a Streaming API and a Search API. The former is used for Tweets that are pushed to the end user clients while the latter is used for requesting specific tweets. Twitter supports the specification of geospatial coordinate systems when harvesting Twitter. As noted, these are typically given as bounding boxes or circles with a given center/radius for Twitter harvesting. These can be tweets that include particular text, from a particular user (e.g. @Yikai) or containing a particular hashtag (e.g. #TrafficCongestion). Twitter harvesting is commonly done at an aggregated level and returns include tweets from within a given area, e.g. a bounding box around the borders of Melbourne. Such an approach would collect large amounts of irrelevant data for traffic analytics, hence a novel solution is required. Specifically a targeted tweet harvester is required to harvest tweets made “only” along streets by utilizing the PSMA Road data. Figure 4.2 illustrates how this harvester works.

A PSMA pre-processor is used to retrieve relevant data from the PSMA Road dataset (in Javascript Object Notation format) that is then passed to the targeted twitter harvester. This harvester calculates a range of centroids used as the
basis/range for the harvesting locations, i.e. these ranges are based on the diameter of the road and the center of the road as indicated in the black area in Figure 4.2. Each of these centroids is used for querying the Twitter Search API for tweets in that region, i.e. that particular centroid.

Each street segment in PSMA contains coordinates that include a start point and an end point. To cover the complete street network, each street segment can be split into several sub-cells with equal distances based upon the diameter of the road (as obtained from PSMA). The measurement of distance is the key to divide a line segment into a series of points. The Haversine formula [178], given in Equation 4.1, is used to calculate the distance (e.g. in meters) between two geographic points based on their latitude/longitude value. Algorithm 1 describes the exact procedure to query Twitter for tweets made along the road network. Because of the frequency limit in querying the Twitter API, the speed of querying tweets made on the streets should be controlled. The behavior when harvesting tweets for many thousands of centroids is like driving and collecting data at a specified speed along the entire street network. The primary constraint of this
approach is that each query can only contain a single centroid. Given this, it is inefficient to use a single harvester for collecting on-street tweets over a large-scale area due to the frequency/speed constraint and the limited area covered by each query. However, the use of Cloud infrastructure makes it possible to use a large and scalable number of harvesters in a coordinated manner, e.g. to collect more data at rush hour.

\[
\begin{align*}
Point_1 &= (lat_1, lon_1) \\
Point_2 &= (lat_2, lon_2) \\
\delta_{lat} &= \text{radian}(lat_1) - \text{radian}(lat_2) \\
\delta_{lon} &= \text{radian}(lon_1) - \text{radian}(lon_2) \\
\phi &= 2 \times \arcsin \sqrt{\sin^2 \left( \frac{\delta_{lat}}{2} \right) + \cos(\text{radian}(lat_1)) \times \cos(\text{radian}(lat_2)) \times \sin^2 \left( \frac{\delta_{lon}}{2} \right)} \\
R &= \text{Average Earth Radius in the area of Point}_1 \text{ and Point}_2 \\
\text{Distance (Point}_1 \text{ to Point}_2) &= \phi \times R
\end{align*}
\]

Each collected tweet has a label added by the targeted Twitter harvester that includes metadata regarding the Tweet such as the query details and the street name. Figure 4.3 shows an example label of one of the collected tweets when included into CouchDB. In this example, the label shows that this tweet was harvested on the Monash Freeway (one of the motorways around Melbourne) and provides the associated bounding box details. Such metadata provides rich information that can be used for analysis.
4.2 Harvesting Tweets from the Road Network

Algorithm 1 Harvesting Tweets along a Street Segment

Input 1: \( Q = \{(\text{lat}_1, \text{lon}_1), (\text{lat}_2, \text{lon}_2), \ldots\} \) — An ordered list of coordinates that represent the geographic information of an input street segment (polyline).

Input 2: \( \delta \) — An interval distance for splitting the street polyline.

Output: \( U = \{(\text{lat}_1, \text{lon}_1), (\text{lat}_2, \text{lon}_2), \ldots\} \) — A list of coordinates for the centroids.

1: \textbf{function} getCentroids(\( Q, \delta \)) \hfill \triangleright \text{Call this function for each street polyline} \( Q \)
2: \( U \leftarrow \{\} \)
3: \textbf{for} \( i \leftarrow 0 \) to \( Q.size - 1 \) \textbf{do}
4: \( P_1 \leftarrow Q.get(i) \)
5: \( P_2 \leftarrow Q.get(i+1) \)
6: \( \text{Distance} \leftarrow \text{getDistance}(P_1, P_2) \) \hfill \triangleright \text{use Equation 4.1}
7: \( n \leftarrow \text{roundUp}(\text{Distance}/\delta)+1 \) \hfill \triangleright n \text{ is the number of required centroids}
8: \( \Delta_{\text{lat}} \leftarrow |\text{getLat}(P_1) - \text{getLat}(P_2)|/n \)
9: \( \Delta_{\text{lon}} \leftarrow |\text{getLon}(P_1) - \text{getLon}(P_2)|/n \)
10: \textbf{for} \( i \leftarrow 0 \) to \( n - 1 \) \textbf{do}
11: \( \text{lat} \leftarrow \text{getLat}(P_1) + \Delta_{\text{lat}} \times j \)
12: \( \text{lon} \leftarrow \text{getLon}(P_1) + \Delta_{\text{lon}} \times j \)
13: \( c \leftarrow (\text{lat}, \text{lon}) \)
14: \( U.insert(c) \)
15: \textbf{return} \( U \)

Figure 4.3: Harvester label added to a given Tweet when stored in the database
4.3 Clustering Tweets using DBSCAN

As noted, a given stretch of road can be considered congested when tweets sent in the same spatial/time window are clustering (on that particular stretch of road). This implies that the temporal and spatial extent of the tweet is captured. Hence the created time and the latitude and longitude of the tweet need to be extracted from tweets to support the clustering process.

Several spatio-temporal clustering approaches have been developed that can be used to detect congestion. [97] surveyed several methods to cluster spatio-temporal data. In relation to social media data, centre-based clustering methods (e.g. K-means [114]) are not suitable because they assume that it is possible to determine in advance how many clusters there would be in a set of data. Density-based spatial clustering of applications with noise (DBSCAN) is a more feasible approach and was adopted for clustering social media data in this thesis. Discussions about the clustering algorithms are revisited in Section 2.4. There are two parameters that need to be pre-set for DBSCAN: a distance threshold $\epsilon$ and a minimum number of points $minPts$. A cluster is formed when a set of data points are close to each other (within $\epsilon$) and when the total number of points is greater than or equal to $minPts$. DBSCAN can identify tweets which are close to each other in time and space. Other tweets can be discarded as random entries (noise), e.g. those that are sparsely distributed in the feature space.

In the case study of this section, we use spatio-temporal tweet clustering as the basis for identifying traffic congestion. In the implementation of the case study, some assumptions are made: if there are four or more tweets sent in a 1 kilometer stretch within 15 minutes, it is considered as possible transport congestion, and the tweets should be marked as part of a congestion cluster. The thresholds underlying this assumption are configurable in the platform. The number of tweets here (four) corresponds to the parameter $minPts$ in DBSCAN. It is noted that there is no general way in choosing $minPts$ for DBSCAN, hence a
heuristic introduced in [147] is utilized: \( \text{minPts} \geq D + 1 \), where \( D \) is the number of dimensions in the dataset. As discussed, the dataset used for clustering has three features: created time, latitude, and longitude. Therefore, the number of dimensions is three, and \( \text{minPts} \) is thus determined as four for this case study. On the other hand, the spatial distance 1 km and the time distance 15 minutes collectively correspond to the parameter \( \epsilon \).

We perform several clustering experiments using the ELKI Data Mining Framework (ELKI — http://elki.dbs.ifi.lmu.de). However, the results were not ideal since the available distance functions for calculating the distance between two points was not universally applicable for both time and space features. As a result, a custom distance function for clustering was defined. Specifically, time and space use different measurements, however, if they are taken to be equally important, then it is possible to give the same weight to the time value and space value. Equation 4.2 presents the initial method for calculating the spatio-temporal distance (\( \text{STD} \)) between two spatio-temporal points (i.e. \( \text{Point}_1 \) and \( \text{Point}_2 \)), where \( TD \) is the gap (in seconds) of the timestamp between \( \text{Point}_1 \) and \( \text{Point}_2 \), and \( SD \) is the spatial distance (in meters) between two latitude/longitude points calculated through Equation 4.1. An improved version of spatio-temporal distance calculation is introduced by Equation 6.1 in Chapter 6 — see page 126.

\[
\begin{align*}
\text{Point}_1 &= (\text{lat}_1, \text{lon}_1, \text{time}_1), \quad \text{Point}_2 = (\text{lat}_2, \text{lon}_2, \text{time}_2) \\
TD &= |\text{time}_1 - \text{time}_2| \\
SD &= \text{getDistance}(\text{lat}_1, \text{lon}_1, \text{lat}_2, \text{lon}_2) \\
\text{STD} &= \frac{TD}{2} + \frac{SD}{2}
\end{align*}
\]  

(4.2)

As noted, we assume 1,000 meters as the value for spatial distance and 1,000 seconds for the temporal distance (about 15 minutes) and these are used to define the \( \epsilon \) threshold in the case study. Specifically, the value of \( \epsilon \) is given as 1,000 (1000
from space $\times 0.5 + 1000$ from time $\times 0.5$.

### 4.4 Implementation and Deployment

For the initial case study, we collected Tweets posted in the Sydney Central Business District (CBD) in May 2015 using a cluster of targeted Twitter harvesters deployed on the NeCTAR Research Cloud. These collected tweets data were processed and stored in a centralized NoSQL database — CouchDB. After the data collection, tweets stored in the database were clustered by a single-process program using the DBSCAN algorithm with specialized settings described in Section 4.3, i.e. Equation 4.2 for calculating the spatio-temporal distance, using the pre-set $\epsilon$ and $\text{minPts}$ for defining/describing the clusters. A web client was created to visualize the results of this case study.

#### 4.4.1 Apache CouchDB — NoSQL Database for Storing Semi-structured Tweets Data

Twitter data represents a form of semi-structured data as introduced in Section 2.2. As such, it is not well suited to traditional relational database systems, e.g. SQL-based systems. To store the collected tweets data, Apache CouchDB [9] was used. CouchDB is a document-oriented NoSQL database with a RESTful HTTP/JSON API used for storing and accessing data. It also supports MapReduce which is a cornerstone of Big Data technologies as introduced in Section 2.2.1. With the MapReduce capability, CouchDB can group and aggregate data to summarize/visualize clustering results, i.e. group or reduce the tweets data by cluster ID which is generated/attached by the clustering program. The tweets themselves are in JSON [46] format and retrieved from the Twitter APIs. These can be directly stored in CouchDB without any additional processing.
4.4 Implementation and Deployment

4.4.2 Software Structure

Figure 4.4 shows the architecture of the software system underpinning this harvesting and analysis applications. A series of Twitter harvesters act as feeds into the system. A group of services/processors are used for the associated data analytics and subsequent visualization. In addition to the clustering process, a real-time service is deployed for accessing and visualizing the real-time tweet streams. It is important to note that this system has been designed to be extensible so that many harvesters can be dynamically deployed across the Cloud. This allows for scalability and flexibility, e.g. supporting targeted harvesters for multiple cities in Australia or more harvesters at rush hour.

Figure 4.4: Software Architecture for Harvesting and Clustering Tweets

It is noted that this system was designed and developed before the appearance of the SMASH platform proposed in Chapter 5. As such, it used a separate cluster and data management solution of the NeCTAR Research Cloud. The capability of the SMASH platform including support for MapReduce by using in-memory computation with spatio-temporal indexing support, provided a simpler and more efficient mechanism to identify the social media data which have spatial correlations with other geographic datasets, e.g. near the streets.
4.5 Results and Analysis of Clustering

A web client was developed for data/analysis and visualization of social media data used as a proxy for traffic data. A variety of statistical review pages were realised for charting tweets on a particular date or a particular day of the week, as well as identifying rush hour and/or potential incidents that may take place around the city of Australia based on the number, size and locations of tweet clusters.

Figure 4.5: Clustering of Tweets in Sydney on May 2015 (small triangles are potential congestion (tweet clusters))

Figure 4.5 illustrates aspects of the functionality in using social media data for understanding urban traffic issues. The system allows users to pick a particular date, a period (morning rush, evening rush or off-peak), a day of the week or a street to view the mapping of tweets in the City of Sydney. Those standalone tweets are marked as points on the map and those groups of tweets that are close to each other in time and space (i.e. they form a cluster) are marked as triangles.

The clusters in Figure 4.5 are identified as potential traffic issues (i.e. congestion) based on the previous assumptions. However, this needs to be validated/verified. This requirement raises a challenge for the work in this thesis.
Chapter 7 presents approaches exploring the possibility of using the spatio-temporal information (clusters) of social media data for identifying urban traffic issues and their actual correlation with real traffic data. Further case-studies are designed to explore the spatio-temporal correlation between the distribution of social media data and the variation of official urban traffic (volume) data.

![Image of social media analytics](image_url)

**Figure 4.6:** Example of real-time social-media analytics

Figure 4.6 illustrates an example of a real-time social media monitoring service. Once a user in a selected area posts a new tweet with a GPS tag, it can be captured and marked on a map. A customized filter is used for highlighting tweets based on the keywords of the captured tweets. Whilst not designed for data analysis, such real-time information gives insights into urban traffic issues.

### 4.6 A Review of Existing Social Media Applications for Urban Traffic Analytics

There are several social media (community) based applications that provide real-time traffic navigation services. Waze[^2] is one such example. It is one of the

[^2]: https://www.waze.com/
most popular social media platforms for getting and reporting real-time urban traffic information with over 50 million users in 2017 [60]. Waze collects map data, travel times and traffic information from users and transmits it to the Waze server [193]. Waze users (Wazers) who use smart phones or GPS supported devices are encouraged to report a variety of traffic-related incidents such as congestion, road closures, car crashes, etc to the Waze server. Such reported information are used to help other Wazers either by alerting them of the conditions ahead and potentially re-routing the user to avoid affected areas. Waze also collects anonymous data like the users real-time locations and speeds to improve its service. Waze is very much a crowd-sourcing application. Crowd-sourcing is a model that obtains goods and services, including ideas and finances, from a large, relatively open and often rapidly-evolving group of internet users [85]. The more people that provide data, the more accurate the extracted information will be [131].

Figure 4.7 shows two screenshots of the Waze App. The screenshot on the left is the road map interface with user reported marks/incidents shown. The screenshot on the right is the reporting page where Wazers can submit their real-time findings on the roads to help others. The report can be attached with user text messages and photos. After submission, such social media reports/messages are made publicly visible (as shown in the left screenshot). Other Wazers can support a published report by clicking the like button or leaving comments much like other social media apps. Waze use the information extracted from these social network behaviors together with the real-time location and speed of active Wazers to evaluate the publish reports and improve the accuracy of their service.

Based on such data collection, Waze is able to provide real-time urban traffic information and routing suggestions. To make convenient to drivers, Waze also provides location-based services, e.g., identifying the cheapest fuel station or parking area near a user or along their route.
Figure 4.7: Screenshots of the Waze App — Road Map Page and Reporting Page

The Cyclers\(^3\) mobile app is another example of a community-based (crowd-sourcing) routing service. It is able to discover and navigate the safest and most pleasant cycling routes including combinations with public transport and bike sharing by leveraging information shared by the user community.

The use case and success of these social-media applications supports the hypothesis of this thesis — social-media data can be used as a cheaper proxy for more official urban traffic systems. However, Waze is a platform-dependent application used to reveal real-time urban traffic information. Unlike mainstream social media platforms like Twitter, it is not ubiquitous (and certainly not in

\(^3\)https://urbancyclers.com/cities
Australia). This brings two challenges in using social-media applications like Waze for real-time urban traffic analysis:

- It requires enough particular platform users (e.g. Wazers) to create the marked maps and continuously update the information to make it useful. Currently, there are only 13 countries that have full base maps. Other countries are incompletely mapped and require users to record roads and edit maps [193].

- The incidents reported by platform users often happen before they are actually observed. Such information can help to monitor existing urban traffic issues but there is a lack of methods to use the data to predict the road network status.

This chapter takes these two challenges into consideration. Firstly, we need a general model to process data from all kinds of social media platforms. Secondly, if certain spatio-temporal relationships exist between social media clusters and the real urban traffic status, we can leverage this relationship to predict the urban traffic issues before they actually happen. This of course needs verification to ensure that the predictions are correct. This is explored in the subsequent chapters.
4.7 Conclusion

One of the most significant advantages of using social media as a data source for urban research is that it is free, widely used by many members of the population, is often spatially represented and can provide near real-time information. In contrast, the cost and lack of real-time analytics support when accessing official urban traffic data remain challenging to many road network organisations such as VicRoads as discussed in Section 1.5.2.

However it is clear that social media will be far more sparse and untrustworthy to other more official traffic data. The trustworthiness of social media requires that any clusters that are identified, i.e. as potential traffic congestion, are measured and compared with official traffic statistics. This in turn requires that systems are established that support such comparisons including real-time comparisons.

The idea and methods presented in this chapter provide two directions to extend our research. Firstly we need to explore the correlation between social media clusters and traffic data/status. This is for verifying our hypothesis whether social media data can indeed be used as a real-time proxy for analyzing urban traffic. Chapter 7 illustrates the case-studies we have undertaken to explore this topic. Secondly, we need a Cloud-based platform to conduct realistic case-studies analyzing (real-time) urban traffic issues. In the following chapter, we introduce the SMASH architecture that has been designed to address this challenge.

This chapter is largely based on the publication:

To tackle the unique challenges associated with traffic data, a dedicated and scalable solution is required. In this chapter we present the SMASH platform. This is realised as a Cloud-based solution designed to meet key big data aspects associated with urban traffic, with specific focus on the volume and velocity of traffic data. SMASH provides a dedicated Big Data as a Service (BDaaS) solution for tackling traffic related spatio-temporal data.

In this chapter, we describe the key aspects of the SMASH solution and its overall architecture. We show how SMASH supports scalable deployment through Docker containers to ease its scaling and management across different Cloud infrastructures (e.g. Hybrid Cloud). These Containers are used to realise three layers of the SMASH: Data Storage Layer, Computing Layer and Interface
5.1 Requirements and Challenges

In Chapter 2, we reviewed many of the common challenges in tackling Big Data. For the urban traffic field in particular, the challenges for analyzing urban Big Data include:

- Continual growth of large volumes of data. Data sources used for urban traffic analytics are usually collected from sensors and can include transport trajectory data and environment data, or reported by individual smart devices (e.g. social media data and metadata [138] communicating with mobile base stations). Data from these sources is continually generated. This requires that scalable data storage is provided for storing and accessing these data in an efficient and durable manner.

- Multiple heterogeneous data stream coming at a high velocity. Multiple sources of data can be streamed at different and variable speeds. Scalable systems and interfaces are thus needed for adapting to such bursty data deluge, e.g. Traffic data generated at rush hour.

- Ability to index and query large-scale geospatial data. Much urban traffic analytics data are geo-coded (e.g. locations of individual cars or traffic trajectories. A range of methods [91, 105, 185, 210] for urban traffic analysis have been proposed with many based on GIS analysis. As such, a GIS supported system is needed to index and query such big geospatial data.

- Ability to apply existing/general processing methods related to urban traffic analytics on the Cloud using generic parallel processing interfaces. In Cloud Computing environments, generic software platforms are required to adapt and support parallel processing of urban traffic data.
• Processing Big Data stream in (near) real-time. Urban traffic analytics are usually time sensitive, e.g. traffic congestion prediction [32]. Therefore, stream processing support is needed to analyze incoming real-time data to produce near real-time results.

• Visualization of data and results of urban traffic analytics. The ability to present spatial and temporal information in analytic results is essential.

Many of the research efforts on big data processing techniques can be traced back to the early 2000s and efforts of Google through the MapReduce programming model, the BigTable DBMS and the Google File System [47, 71]. There is no definitive description of big data, but generally it refers to data-sets of such size (volume) that they cannot be stored on a single machine (server) and the analysis cannot be undertaken on a single machine (server). These issues place numerous demands on the underlying data infrastructure including the need for fetching the data, searching the data, storing the data and visualizing the data.

The SMASH architecture was designed to tackle these issues directly with specific focus on the demands of urban traffic analytics. Firstly, it was recognized that a distributed file store was needed. Large volumes of data that are real-time in nature require that distributed data storage is in place. To tackle this, the SMASH stack utilizes the Hadoop Distributed File System (HDFS) [24]. Secondly, there is a need for performing computation on traffic data, such as aggregation, statistics, and machine learning. Since the computations are complex in nature, Apache Spark was adopted over native Hadoop MapReduce. Traffic data are spatial and temporal in nature, hence it is essential to be able to index them spatio-temporally, and visualize them on maps. As a result GeoMesa and GeoServer were added to the SMASH stack. GeoMesa provides a spatio-temporal indexing strategy on top of the Accumulo BigTable DBMS, while GeoServer is used to serve maps and vector data to spatial clients.
5.2 The Cloud Platform as a Service (PaaS)

The SMASH platform itself can be deployed on any Cloud infrastructure. In this thesis, the SMASH solution was deployed on the federally funded Australia-wide National eResearch Collaboration Tools and Resources (NeCTAR — https://www.nectar.org.au) Research Cloud. The NeCTAR project was established to realize a national Cloud facility for researchers across Australia. NeCTAR utilizes the OpenStack Cloud middleware and has multiple Cloud availability zones across Australia. At present, NeCTAR provides almost 30,000 servers across multiple availability zones in Melbourne, Monash, Brisbane, Canberra, Adelaide and Tasmania. The primary focus of NeCTAR has been to establish an Australia-wide IaaS service. The SMASH platform uses OpenStack and the Docker Engine API to deploy and manage the Feature-oriented SMASH platform with targeted analytics required by the transport domain.

The SMASH platform instance, described in this chapter, utilizes 44 CPUs with 180GB of RAM and 3.9TB of storage. The system has been designed to scale however.

5.3 Software Stack

The overall architecture of SMASH and a comparison with the traditional Hadoop ecosystem [192] is shown in Figure 5.1. Each of these capabilities of the stack is described below in turn and the role they perform within the SMASH architecture.

5.3.1 Hadoop Distributed File System (HDFS)

At the lowest level of the SMASH architecture lies a distributed file system: HDFS. HDFS offers a reliable, scalable and essentially a distributed file system. With the volume and velocity of traffic data, there is an absolute need to distribute the
5.3 Software Stack

Figure 5.1: SMASH Architecture compared to the traditional Hadoop ecosystem

storage and access to the data. A single server cannot cope with large volumes of high velocity data hence spreading the ingestion of data and the subsequent storage across multiple servers is essential. A typical composition of a HDFS system is illustrated in Figure 5.2.

Figure 5.2: Traditional Hadoop Distributed File System Architecture

In HDFS, a namenode process and a set of datanode processes run on a group of machines. A namenode hosts the metadata of data chunks stored in the file system to indicate which datanode hosts which particular data chunk. HDFS chunks
large files into smaller blocks (128 MB by default) then spreads and duplicates them across multiple machines (datanodes). HDFS components communicate with each other via TCP/IP sockets. HDFS also offers robust strategies to cope with single/multiple nodes failure. To tackle datanode failures, HDFS automatically replicates each data chunk to multiple datanodes according to the Hadoop settings. Based on the replication settings of Hadoop, data can be guaranteed to be available should a given datanode fail. To tackle namenode failures, HDFS offers a secondary-namenode, which can be launched on a different machine to the namenode server. This component acts as a live backup of the namenode and can take the place of a given namenode in case of a given namenode failure.

A core service provided by HDFS for data processing is MapReduce. MapReduce provides a common way to handle the processing of large amounts of data based on the idea of divide and conquer. A MapReduce job typically involves splitting an input dataset into independent data chunks, which can then be processed by the map tasks in parallel. This model of distributing the processing activities to separate servers means that it is highly scalable and hence suited to Cloud-like environments where more servers can be added on demand. The MapReduce framework normally involves a master JobTracker and one slave TaskTracker per node. The master is responsible for scheduling and monitoring slave tasks. A typical dataflow of MapReduce is shown in Figure 5.3.

The Hadoop MapReduce framework allows processing large amounts of data in parallel across a cluster of servers. It has been developed and applied as a mature technique for many years. More recently Apache Spark is a new-generation, fast engine for large-scale data processing [153]. It offers improved performance over MapReduce through in-memory data processing. We have included Apache Spark into the SMASH architecture.
5.3 Software Stack

5.3.2 ZooKeeper

In setting up the SMASH software stack, capturing the detailed configuration information for the various software systems and tools is essential. Configuration, especially in distributed environments involving multiple systems and services, can require a lot of work to set up, update and manage. An erroneous configuration can adversely impact upon performance and lead to unreliable systems. ZooKeeper [79] is one of the most commonly used tools for managing configuration information and is used in the SMASH software stack.

ZooKeeper offers replicated synchronization services with eventual consistency offered for tools and their configurations. It provides degrees of robustness by migrating multiple nodes as ensembles where clients can connect to any of them to get their associated configuration information. As long as the majority of nodes are working, the ensemble of ZooKeeper nodes is kept alive. This means that a master node can be dynamically chosen by consensus within the ensemble. If the master node fails, the role of the master can automatically be passed on to another node.
5.3.3 Apache Accumulo/GeoMesa

On top of HDFS is a distributed spatio-temporal database that is needed for storing and querying diverse traffic-related data. The software solution selected to satisfy this requirement was GeoMesa with Apache Accumulo. Apache Accumulo (https://accumulo.apache.org) is a sorted, distributed key/value data store that provides a robust, scalable, high performance service. It was inspired by the BigTable technology from Google, but has improved over time with the introduction of cell-based access control and server-side programming mechanisms that can be used to modify key/value pairs at various points in the data management process. Accumulo is one of the most popular NoSQL databases currently available.

GeoMesa (https://www.geomesa.org) is an open-source, distributed, spatio-temporal database that can be deployed on many Cloud data storage systems, including Accumulo, HBase (https://hbase.apache.org), Cassandra (https://cassandra.apache.org), and Kafka (https://kafka.apache.org). Leveraging a highly parallelized indexing strategy, GeoMesa aims to provide high levels of spatial querying and data manipulation. GeoMesa is capable of ingesting, indexing and querying billions of geometrical features. It is integrated into the GeoTools open-source library and can act as a data source for GeoServer by the use of bespoke plug-ins. GeoMesa implements an indexing strategy using a geo-hashing algorithm (three-dimensional Z-order curve), which combines three dimensions of traffic information (latitude, longitude, and time) into a single dimension lexicographic key space in Accumulo.

5.3.4 Apache Spark

A big data processing engine is required in SMASH for some traffic analytic methods. Hadoop has a native MapReduce data processing engine which is good for simple parallel tasks, e.g. filtering, searching, and counting. However, this native
engine is both difficult and inefficient when used for more complex algorithms, e.g. spatial clustering, image processing, and machine learning approaches. For the SMASH architecture, Apache Spark (https://spark.apache.org) was chosen as the alternative data engine inspired by Hadoop MapReduce but utilizing in-memory processing suitable for more complex calculations. Some of the most important features of Apache Spark that support the SMASH architecture are described below. Firstly, Apache Spark offers Resilient Distributed Dataset (RDD) capabilities. RDD is a fundamental data structure of Spark that abstracts the parallelization of distributed data chunks for developers. This provides a fault-tolerant abstraction for in-memory computing. RDDs are resilient because they have a long lineage. Whenever a failure in the system occurs, they can recompute themselves using prior information available through this lineage. Operations on RDDs are categorized into Transformations or Actions. Transformations are operations that transform an RDD to another RDD, while Actions are operations that get the results of the processing. Importantly, a Spark job can be evaluated and divided (using a directed acyclic graph (DAG)) into tasks and stages that are sent to different executors, allowing decomposition of processing. Similarly, data within the same RDD can be divided into many partitions to control the granularity of in-memory data.

Apache Spark is much faster than the Hadoop MapReduce. There are a variety of factors for this speed-up. Firstly MapReduce stores data back to the disk after a map or reduce action, while Spark leverages in-memory RDDs and “spills” data on disk only when it exceeds memory limitations. Thus, the execution of Spark is faster than MapReduce by reducing the amount of I/O operations. Secondly, RDDs allow recovery of failed nodes by re-computation using the DAG predecessor. In some ways, this supports a similar recovery to MapReduce by way of checkpointing. However, to reduce the dependencies of lineage further, it is possible to combine those two methods together in Spark. As a result if an iterative computation program needs to be executed, e.g. a machine learning algorithm,
Spark has a better performance than MapReduce due to its in-memory processing. One point to make however is that Spark has stricter memory requirements. Thus in a Spark cluster, the memory should be at least as large as the amount of data to process, because the data has to fit into memory. Block sizes are thus important: processing big data sets requires larger memory, but if only smaller data sets are to be processed then they will waste block space (as they will only use a fraction of the block size).

Figure 5.4 shows the typical execution of an Apache Spark application on the SMASH platform.

![Figure 5.4: SPARK workflow on SMASH](image)

Spark has the ability to read/write data to/from both the bottom level HDFS (e.g. rows of data in a CSV file) and the middle level Accumulo/GeoMesa (SimpleFeature objects of GeoTools). GeoMesa/Accumulo have some native supports for Spark including offering abstract interface for loading/writing data and SparkSQL support for manipulating data. Querying spatio-temporal data from the SMASH storage stack to RDDs in Spark is made user friendly by using SQL queries based on the Open Geospatial Consortium (OGC) methods (http://www.opengeospatial.org/standards/sfs/). For example, if Apache is used for analyzing large amounts of spatio-temporal data, additional toolkits like GeoTools
5.4 Container-based Implementation and Management

(http://geotools.org) often need to be used for spatial operations. The use of such tools can be complex and less efficient than leveraging the spatio-temporal index/functions available in GeoMesa/Accumulo.

5.3.5 GeoServer

A further demand of the SMASH architecture is for data visualization. Traffic-related data and the results of analysis are usually presented at geographic aggregation levels and/or in some form of spatial representation, e.g. congestion on roads visualized on a map. To this end, GeoServer is used to provide a standard map service for visualizing many kinds of spatial data.

GeoServer (http://geoserver.org) is an open-source server based on Java technology that allows users to share, process and edit geospatial data. Designed for interoperability, GeoServer allows users to publish data from spatial data sources using open standards — typically based upon the Open Geospatial Consortium (OGC). GeoServer implements a range of web-based administration interfaces and associated Rest-based APIs for easy configuration. When publishing data, e.g. a map layer, a time dimension can be specified so that it is possible to create visualization with temporal dimensions. Such capabilities are important, e.g. to be able to visualize and process traffic flows at different parts of the day.

Through use of the GeoMesa plugin, spatio-temporal data stored in the SMASH platform can be readily added to GeoServer as data layers, including support for tiled web map services to visualize the results of analysis or raw traffic data on a base map such as Google Maps and Open Street Map.

5.4 Container-based Implementation and Management

One of the key requirements in designing the SMASH architecture is scalability. Each of those software components in SMASH is designed to scale. To minimize
the efforts for scaling the SMASH system, e.g. as a cluster, we use the Docker Engine (https://www.docker.com) as a software solution. Docker is a container-based engine, which allows to host applications in separate system environments on a single physical underlying machine. Each software component of SMASH is packaged with its required settings into separate Docker images to provide different service layers. For instance, we create a Docker image for a Spark service as the computation layer, a separate image GeoServer service as the visualization layer and another image for GeoMesa, Accumulo and HDFS services as the data storage layer. The benefit of this implementation is that a given software component can be switched or upgraded without affecting others by simply changing the image. Figure 5.5 illustrates the structure of this Dockerized implementation of SMASH.

**Figure 5.5:** High-level SMASH Software Architecture and its Dockerized Implementation

Existing products such as the Docker-based Kubernetes [21] and Docker Swarm [127] are increasingly available for auto deploying and scaling Docker-based applications. SMASH offers a one-stop solution for managing everything on a SMASH cluster including virtual machine management, the firewall groups/rules configurations and Docker images/container management. It provides a command-line-based client, written in JavaScript with associated NodeJS run-time environment, that can communicate with both the SMASH software stack containers and the underlying Cloud infrastructures. This client is called Clouddity. It utilizes several open source libraries to realize its overall functionality. For instance,
pkgcloud (https://github.com/pkgcloud/pkgcloud/) abstracts away differences between multiple Cloud providers including managing the VMs and firewall rules on the Cloud. Dockerode (https://github.com/apocas/dockerode/) is a remote Docker API used for managing Docker images/containers. Dockprom (https://github.com/stefanprodan/dockprom/) is used to monitor the status and resource usage of each Docker container. Figure 5.6 is the help information of this Clouddity client and it offers various features to control the SMASH platform. Figure 5.7 presents an example of listing the node information of SMASH.
5.5 Benchmarking the SMASH Platform

As mentioned, SMASH has been designed to scale. In this section we present the actual benchmarking experiments performed on a given deployed SMASH platform instance. The experiments focus specifically on data ingestion and processing. A target SMASH Dockerized cluster was deployed on the NeCTAR Research Cloud.

To benchmark the performance, a range sizes of data-sets (from 1Gb through to 70Gb of SCATS data) were used as resource against which different sizes of SMASH clusters (i.e. a master node with multiple slave nodes) were compared. In the experiments, the master node (VM) had four CPUs offering 2600MHz + 512KB cache and 12GB RAM with 80GB attached storage. The slave nodes (VMs) each had two CPUs offering 2600MHz + 512KB cache with 8GB RAM and 60GB storage. All of the VMs were built using Ubuntu 16.04 LTS (Linux Kernel Version: 4.4.0-21-generic). All nodes were connected with a 10Gb backbone network. Docker version 1.11.2 was used and all containers were run in host mode.

The following subsections cover the details of the used data and benchmarking case-studies.
5.5 Benchmarking the SMASH Platform

5.5.1 Data for Benchmarking

The SCATS volume data, introduced in Section 1.5.2 page 20, was used as the data resource to benchmark the SMASH platform. The data was collected through SCATS sensors deployed on the major street network of Australia. A typical example of a SCATS enabled sensor network is shown in Figure 5.8 in which each red dot is a SCATS sensor. This sensor network is for the area around Melbourne Central Business District (CBD). Figure 5.9 shows the whole SCATS sensor network installed in Victoria.

![Figure 5.8: SCATS Sensors Installed in the Melbourne CBD](image)

The SCATS system supports the capture of dynamic timing of signal phases at traffic signals. The system uses sensors, typically at each traffic signal through a strip across the road, to detect vehicle presence in each lane. Information collected from vehicle sensors allows SCATS information (in principle) to be used to calculate and adjust the timing of traffic signals in the network for improving traffic throughput. Figure 5.10 demonstrates how SCATS sensors work at a par-
Table 5.1: Examples of SCATS traffic volume data

<table>
<thead>
<tr>
<th>site_id</th>
<th>street name</th>
<th>HFID</th>
<th>date</th>
<th>count_1</th>
<th>...</th>
<th>count_96</th>
</tr>
</thead>
<tbody>
<tr>
<td>4700</td>
<td>McNaughton Rd</td>
<td>9508</td>
<td>2008-01-02</td>
<td>{vc1}</td>
<td>...</td>
<td>{vc96}</td>
</tr>
<tr>
<td>487</td>
<td>Forester Rd S</td>
<td>15088</td>
<td>2008-01-02</td>
<td>{vc1}</td>
<td>...</td>
<td>{vc96}</td>
</tr>
<tr>
<td>4905</td>
<td>Park St E</td>
<td>19928</td>
<td>2008-01-02</td>
<td>{vc1}</td>
<td>...</td>
<td>{vc96}</td>
</tr>
</tbody>
</table>

Figure 5.9: SCATS Sensors Installed around Victoria

SCATS sensors are realised as inductive loops installed within the road pavement. These sensors are installed in each lane to detect and count the number of passing vehicles. Table 5.1 lists three examples of SCATS data. Each SCATS data tuple represents the data coming from a single sensor in a single day and each day is divided into 96 time slots. Thus, the number of vehicles every 15 minutes can be directly read from this data structure. Data from vehicles that traverse these road sections is automatically captured and reported to a central database from the Department of Transport, e.g. VicRoads in Victoria. This table shows a small example of the kinds of data that are captured through
SCATS sensors. There are more fields in the metadata (e.g. traffic flow direction, sensor id, traffic flow id) of SCATS volume data. The HFID in Table 5.1 stands for Heterogeneous Traffic Flow ID which can be used as the foreign key to join the SCATS supportive geometry data set as shown in Table 5.2. The ‘point.geometry’ represents the geographic information of the SCATS sensor devices and the ‘line.geometry’ is the geographic information of the traffic flow (directional street lines) where SCATS sensors are installed.
Table 5.2: Example of the SCATS supportive geometry data

<table>
<thead>
<tr>
<th>HFID</th>
<th>line_geometry</th>
<th>point_geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>9508</td>
<td>LineString (lon1 lat1, lon2 lat2)</td>
<td>Point (longitude latitude)</td>
</tr>
<tr>
<td>15088</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19928</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

5.5.2 Benchmarking Data Ingestion

In this case study, we explored the use of SMASH to ingest SCATS data into GeoMesa/Accumulo. Raw SCATS data was stored as two tables in two CSV files. A volume-table dataset containing the vehicle volumes from every sensor over given time periods together with the sensor ID, HFID and date was used. The sensor-table dataset (supportive geometry data) contained the HFID with its geo-information of point (sensor location) and line (traffic flow on street). The sensor-table acts as a reference to the volume-table. The size of the volume-table (CSV file) can be huge and depends on the historic amount of SCATS data that has been collected. When importing SCATS data into SMASH, the ingesting work involves joining two major tables (using the HFID as the joining key) in plan text file and importing the result-table into GeoMesa/Accumulo as indexed geospatial data. The size of the used sensor-table (CSV file) for Victoria was 5.3GB.

For benchmarking, different sizes of volume-tables need to be joined with the same sensor-table. In this experiment, we varied the number of slave nodes in SMASH during this activity. Each test was conducted three times, and it was noted that the results were very similar to one another. Figure 5.11 presents the averaged results of this benchmarking work. From the chart, we see that it takes more time to ingest larger data sets into the Cloud-based SMASH cluster with the same number of slave nodes.

The increase in execution time is stable however, i.e. we do not find a sharp performance drop when increasing the size of the volume data. For the same
5.5 Benchmarking the SMASH Platform

Size of data, a larger cluster with more slave nodes does not always have the best performance in our benchmarking test. For example when importing 15GB volume-data as shown in the bar chart, there is a significant performance increase from 2-slaves to 4-slaves; however, the performances are similar between 6, 8 and 10 slave clusters and 8-slave clusters perform even better than 10-slaves cluster in ingesting 15GB SCATS volume data. The same scenario can be found in ingesting 10GB, 20GB and 30GB datasets as well. This is because the data shifting among nodes (for joining tables) can become a performance bottleneck in such scenarios.

The procedure of ingesting the SCATS data can be divided into two parts. The first part is to join two large tables via MapReduce [47] which involves transmitting mapped data into corresponding nodes. The second part is to write data into the local Accumulo/GeoMesa instance at each node. Having more nodes means there are more writers to write joint data entities simultaneously to the disk volume. However, the workload of data transmission can significantly increase if we add many nodes into the cluster especially when dealing with large volume data. In case of the 12-slaves with the 60GB dataset, the workload of shuffling such size of data among 12 nodes drags down the performance even this cluster has more executors than the 10-slaves cluster.
Figure 5.11: SCATS Data Ingestion Benchmarking Chart
5.5.3 Benchmarking Data Aggregation

In this case study, we benchmarked the performance of data analytics using Spark with the SCATS data stored in GeoMesa/Accumulo. To support this we consider data aggregation by day of week, e.g., to show the average vehicle-volume change of a particular road on Mondays. The Spark tasks load data into Spark run-time as a Resilient distributed dataset (RDDs) [206]. These data are then used to perform MapReduce and subsequently store the results back to GeoMesa/Accumulo for visualization. The results of the data aggregation benchmarking are shown in Figure 5.12 and examples of the resultant visualization shown in Figure 5.13 and Figure 5.14.

In this case, the change in performance is different to the ingestion benchmarking presented in Section 5.5.2. For example in testing with 12-slaves, aggregating larger datasets does not always take more time for execution especially for large clusters with many nodes. For 2-slave clusters and 4-slave clusters, we observe a more stable time increase when aggregating increasing amounts of data. As was the case when benchmarking data ingestion, we observe that larger clusters do not always out-perform smaller clusters. The 15-40GB groups in Figure 5.12 illustrate this phenomenon. This might be due to the irregular network traffic at the time of benchmarking — noting that the NeCTAR Research Cloud is a shared facility used by many data-intensive research communities.

We emphasize that Figure 5.13 and Figure 5.14 are not simply visualizations that are statically created. Rather, these show the aggregation of SCATS data during a 24-hour period with real-time, high throughput data processing occurring. The images show the processing of large volumes of SCATS traffic data. Figure 5.13 shows typical off-peak traffic flows whilst Figure 5.14 shows typical rush hour flows (on a Monday). The darker lines indicate higher volumes of traffic. This real-time data processing and visualization is only possible through platforms such as SMASH and large-scale Cloud infrastructures.
Figure 5.12: SCATS Data Aggregation Benchmarking Chart
5.5 Benchmarking the SMASH Platform

Figure 5.13: Visualisation of Lower Volumes of Traffic Data Across Melbourne using Real-time SCATS Data Processing

Figure 5.14: Visualisation of Higher Volumes (e.g. rush hour) of Traffic Data Across Melbourne using Real-time SCATS Data Processing
5.6 Conclusion

In this chapter, we presented and benchmarked a big data analytics platform (SMASH) suited for urban traffic researches and this work solved our first hypothesis raised in Section 1.5.3. The data used for the performance benchmark focused on road traffic data, however, it should be noted that the system is generic and can be applied with other large-scale high velocity data sets, e.g. social media data, weather data, pollution data.

After establishing a specialized and scalable platform for traffic analytics, the next step of our research is to consider if new algorithms are needed for analyzing real-time traffic information (by using social media or other spatio-temporal data). In Chapter 4 and Chapter 7, we identified that the spatio-temporal density of social media clusters could be used for identifying traffic issues. However, the question remains whether novel algorithms are needed to identify/analyze such spatio-temporal social media clusters. One way to achieve this is to identify these clusters and the associated traffic abnormalities in real-time. In the following chapter we introduce a new variation of DBSCAN (i.e. RT-DBSCAN) to meet this need.

This chapter is largely based on the publication:

In this chapter, we introduce a real-time density-based clustering algorithm (RT-DBSCAN) that can be used to process social media streams as a proxy for traffic analytics. Clustering Big Data streams such as social media requires algorithms that support real-time scalable analytics capabilities.

6.1 Motivation

Clustering is one of the major data mining methods used for knowledge discovery that has been applied to Big Data [78]. Density-based clustering algorithms like
DBSCAN [56] are in widespread use and numerous extensions are now available for discovering patterns and clusters in large data sets [22, 81, 137, 147]. However, neither DBSCAN nor its extensions support real-time processing or can tackle streamed (high velocity) data [148]. Rather, DBSCAN operates in a batch mode where all the data is acquired and then processed. This feature makes it unsuitable for supporting the ever-growing data from real-time data streams.

There is a strong need for real-time cluster discovery in many diverse application domains beyond urban traffic analytics, e.g. emergency response, network access patterns amongst others. Historically, DBSCAN has been the most widely used clustering algorithm. However, it has limitations for real-time clustering. The demands for real-time clustering of Big Data raise several needs and requirements for improvements and refinements of the DBSCAN algorithm, including the ability to:

- generate a series of up-to-date intermediate result checkpoints when processing real-time (incoming) data;
- support scalable parallel execution capabilities to reduce the response time for generating checkpoints, and
- offer consistent performance in tackling the ever-growing amount of data being generated.

Existing extensions of DBSCAN offer no solution to the combination of these requirements. In this chapter, we present a real-time parallel version of DBSCAN (RT-DBSCAN) to address the above requirements. Compared to the original version of DBSCAN, optional parameters are added to the algorithm for controlling the efficiency and granularity of parallel-workload division. For clustering spatio-temporal data in time and space, a spatio-temporal distance is applied, noting that the definition of distance in this algorithm can be adapted to other kinds of higher dimensional data. We have implemented RT-DBSCAN using Apache
6.2 Challenges

Clustering algorithms like DBSCAN normally need to input the whole dataset into a clustering process (all-in with single-out). The complexity of DBSCAN is $O(n \log n)$. A typical DBSCAN traverses the whole dataset and identifies the neighbors of each point. Each data element can be used/processed multiple times (e.g. as candidates to different clusters). Although incremental-DBSCAN supports updating of clusters by inserting new input data into existing clusters, this algorithm does not cope with ever-growing sizes of historical data due to the data traversal demands of DBSCAN. To deal with this, incremental-DBSCAN [57] reviewed in Section 2.4 drops outdated data to keep a fit size of the dataset, i.e. a fixed and relatively small size of dataset to maintain the clustering efficiency.

A key challenge of real-time DBSCAN is in controlling the size of traversal data needed to cluster ever-growing data volumes. In the DBSCAN algorithm, for each new input data, a group of potential nearby points needs to be identified for cluster detection. For each new input point, if there is an efficient way to identify a full set of essential nearby context points in the historical dataset, only this subset of data is needed for clustering against any new input. Therefore, we can input the data point-by-point into the clustering process and get a series of up-to-date cluster checkpoints. If the performance of this pre-filtering method (i.e. getting the nearby context points) is not sensitive to the size of the dataset, then we can cluster real-time stream data on-the-fly without being challenged by the ever-growing volume of data streamed over potentially extended time periods.

This idea forms the basis for the definition of our real-time clustering method (RT-DBSCAN):
Definition 1 For a new input point $p$ and a group of historical clustered points $cPoints$, $\text{nearbyCtx}(p, cPoints)$ is used to obtain a subset of $cPoints$ which contains essential information of nearby inputs. Checkpoints produced by RT-DBSCAN on this subset must be identical to the result of applying normal DBSCAN on the whole dataset for each input.

6.3 Algorithm

This section covers the algorithmic details of RT-DBSCAN. It starts from a solution to tackle the challenges identified in the previous section to the application of this method in two versions. The first version is for serial processing, i.e. with a single processor. The second version is for parallel processing, i.e. with a cluster of real-time processors.

6.3.1 Identifying a Full Set of Essential Nearby Context Points for Each Input

The original DBSCAN algorithm involves two key parameters: $\epsilon$ and $\text{minPts}$. The distance parameter $\epsilon$ defines how close two points need to be in order to be considered in the same neighbourhood. The border parameter $\text{minPts}$ defines how many neighbourhoods related to a single point there should be for this to be considered as a cluster. In the following examples, we consider a scenario where $\epsilon = 1$ unit and $\text{minPts} = 3$ points.

In Figure 6.1, we highlight three scenarios related to putting a new point $P_i$ into a set of historical clustered points. In scenario A, we consider firstly retrieving all the historical data within $1-\epsilon$ distance to $P_i$, to get 3 non-clustered (noise) points. Since their distances to $P_i$ are smaller than $\epsilon$, $P_i$ now has 3 neighbours, and thus these four points form a new cluster. This seems sensible, but it can be wrong. If there is a point $P_b$ that is less than $1-\epsilon$ away from point $P_a$ but more than $1-\epsilon$ away from $P_i$, as shown in scenario A of the Figure 6.1, $P_b$ will be ignored by this
procedure. Although a new cluster is identified, \( P_b \) is not marked as a member of this cluster — it should be. This result is inconsistent with the assertion made in Definition 1. If we consider extending the range of the nearby context from 1-\( \varepsilon \) to 2-\( \varepsilon \) as shown in scenario B, 4 points are discovered including \( P_b \). These 5 points are grouped into the same cluster. A similar question naturally arises. What if there is a point \( P_c \) which is less than 1-\( \varepsilon \) away from \( P_b \) and more than 2-\( \varepsilon \) away from \( P_i \) as shown in scenario C? This sounds like an endless issue, but it is not. Since \( P_a, P_b \) and \( P_c \) are historical processed points and \( minPts = 3 \), a cluster
must have already been identified when the last of these three points was input in a previous iteration. In scenario C, although \( P_c \) is ignored as a nearby context point, its cluster information is carried by \( P_a \) and \( P_b \) and subsequently passed to \( P_i \). With this information, \( P_i \) and two other noise points can be absorbed into an existing cluster. This result meets the requirements of Definition 1. In this case, where \( \text{minPts} = 3 \), we find that historical points within distance \( 2\varepsilon \) away from the input data contain enough information to establish connections between the input point and the existing historical noise points and existing clusters. The minimum distance (\( \text{minDis} \)) of nearby-contexts, which is \( 2\varepsilon \) in this case, is related to the parameter \( \varepsilon \) and \( \text{minPts} \). \( \text{minDis} \) is given as: \( \text{minDis} = (\text{minPts} - 1) \times \varepsilon \).

In the following section, we mark this procedure as \( \text{nearbyCtx}(p, \text{cPoints}) \) where \( p \) is the input and \( \text{cPoints} \) is the historical dataset. If \( \text{cPoints} \) are indexed in a database, the response time of this procedure needs to be fast and non-sensitive to the growing size of \( \text{cPoints} \).

### 6.3.2 Insertion of Data into Existing Clusters

After identifying the nearby context, the next task is to insert input points into existing clusters according to the information revealed by their nearby context. Inserting input points one-by-one is the most straightforward approach, and this forms the single process version of RT-DBSCAN.

The challenge of this simple approach is how to build connections between each point and its nearby points. Numerous factors and scenarios can have an impact on adding a new point into historical data points and clusters. Specifically for each input:

- IF we do not have enough neighbors and none of the neighbors have enough neighbors, THEN the input is a noise point and has no direct impact on others.
• IF we do not have enough neighbors, but some of the neighbors have enough neighbors to form a cluster, THEN we mark those neighbors as cores of the new clusters AND the input point is a border point for this new cluster or clusters. A core point can only belong to one cluster while a border point can belong to multiple clusters. We merge the clusters if the core point/points connect them.

• IF we have enough neighbors but all neighbors are noise points, THEN the input is a core point of a new cluster. Its neighbors can be core points or border points depending on the number of neighbors.

• IF we have enough neighbors and all non-noise neighbors belong to the same cluster, THEN the input is a core point. The existing clusters are extended by absorbing all associated non-clustered points. The neighbors of input points can be core points or border points depending on their number of neighbors.

• IF we have enough neighbors and those non-noise neighbors belong to multiple different clusters, THEN the input is a core point. We then merge the associated clusters into one cluster and extend the merged cluster by absorbing all associated non-clustered points. The neighbors of an input point can be core points or border points depending on their number of neighborhoods.

These rules drive the design of the RT-DBSCAN, and point-by-point insertion can be repeatedly conducted by following the rules. Additionally, this approach is adapted as several intermediate steps in the parallel version of RT-DBSCAN as described in the following sections.
6.3.3 Converting Point-by-Point Clustering to Tick-by-Tick Clustering for Real-time Parallel Processing

This section covers the idea of parallel RT-DBSCAN. The above algorithm describes an idea for point-by-point iterative/continuous DBSCAN data processing. It avoids traversing the whole historical dataset for every single input so that the number of calculations can be minimized for real-time usage. Executing a single process for RT-DBSCAN by tackling the incoming data stream point-by-point is not an efficient approach and would not meet the requirements of real-time data intensive applications. The challenge is thus to process the incoming data in parallel to increase the throughput of the implementation. However, considering the nature of DBSCAN, it is hard to process data streams in a fully parallel manner since DBSCAN is based on a sequential processing model, e.g. using data in batches or micro-batches. For each input data, a group of nearby data will be queried and involved in the calculation. It is very likely that one input point can be used within the context of another input point. Therefore, these input data cannot be processed without impacting one another when used in a fully parallel manner. For example, assuming $\text{minPts}$ equals 3 and points $P_a, P_b, P_c$ are within $1-\epsilon$ distance to one another. If $P_a, P_b, P_c$ arrive simultaneously and are processed in parallel, their cluster information may not be identified since they are absent in each other’s nearby context. In addition, writing the historical datasets simultaneously can lead to inconsistency of the results. From this we can conclude that: 1) each input point should know the other input data that is being processed in parallel; 2) conflicts between the results of each parallel processor need to be solved before clusters are used/persisted, and previous results need to be persisted into historical datasets before tackling new (incoming) data.

To tackle these challenges for parallel processing, we convert the point-by-point RT-DBSCAN to a (temporal) tick-by-tick RT-DBSCAN where data points from incoming data-streams are divided into separate ticks based on their arriving times. Within each tick, data are processed and clustered in parallel. The results
of each parallel processing step (within one tick) need to be merged before being persisted to solve any/all conflicts in data consistency, and at the end of each tick, the result must be persisted into the historical dataset before a new tick is started.

The updated dataset at the end of each tick represents a checkpoint for the clusters. A series of checkpoints forms the growth history of clusters. The size of a tick is a parameter that needs to be considered in RT-DBSCAN. Generating a new checkpoint for only a few new inputs would likely be a waste of computational resources. However, this may sometimes be needed to guarantee an up-to-date service. To address this, a combination of time-based and volume-based ticks can be used for balancing the performance and output latency.

### 6.3.4 Parallel Processing Tasks For Each Tick

In this section, parallel processing within each tick task is described. To get the nearby context points for a group of inputs in a given tick, the first task of each tick is to get the nearby points context for input point-set (iPs) \( \{P_1, P_2, \ldots, P_j\} \). Depending on how the historical data (cPoints) is persisted, there are different ways to achieve this. The first option is to execute \( \text{nearbyCtx}(p, \text{cPoints}) \) multiple times for each point and merge the returned sets. The second option, as shown in Figure 6.2, is to get the nearby context at one spot by generating a bounding box for the iPs.

If cPoints are stored in a database, the second option is normally preferred since it reduces the number of queries to the database. Firstly, a minimum bounding box is calculated to cover every point in the iPs. Then a new bounding box is generated by extending the previous rectangle with \( \text{minDis} \) for each border. The new bounding box is used for establishing the nearby points context from cPoints. Compared to option 1, the drawback of this method is that it can add unnecessary historical data into the nearby context. In the RT-DBSCAN implementation, a partitioning method to overcome the drawbacks of option 2 is utilized. The
nearby context aggregates historical points together with the iPs passed into the next task for parallel processing (i.e. iPs $\cup$ nearbyCtx(iPs, cPoints)).

The next task is to support location-based data partitioning and parallelisation of RT-DBSCAN. Inspired by MR-DBSCAN [81], each tick of the parallel RT-DBSCAN is designed in a MapReduce manner. In step 1, the data space is geographically divided into many cells. Each cell contains localized input points. In step 2, multiple local DBSCANs are executed on each cell in parallel. In step 3, the results from each cell are merged to recover the border information broken by the space division. For example, single clusters appearing in multiple cells need to be identified and potentially merged. This is achieved in several steps. Firstly, a fast data division for parallel RT-DBSCAN is required. MR-DBSCAN uses cost-balanced (CB) partitioning, as shown in Figure 6.3, to divide the data points in space into cells. It first divides the data space into equal sized small unit cells. Then, a balanced tree is calculated for merging these unit cells into many CB cells where each CB cell contains nearly the same number of points.

Figure 6.2: Example of a Bounding Box for Multiple Inputs and their Nearby Points Context
This method is good at generating balanced workloads for parallel DBSCAN. However, generating a balanced tree can be computationally expensive. MR-DBSCAN is designed for processing a large amount of data in a single batch task, and its CB partition is only applied once at the beginning of the clustering procedure. However, in the RT-DBSCAN realization, partitions are calculated at the beginning of each tick. This approach makes it impossible to reuse the partitions in previous ticks since point-sets within different ticks have different nearby point contexts. To address this, a new partition method, Fast Clustering (FC) partitioning, is designed which is more suited to RT-DBSCAN.

As illustrated in Figure 6.4, the idea of this partitioning method is to iteratively divide a 2D space into four sub-cells until a threshold is reached, e.g., a threshold related to the number of points in a cell. Following this, we drop the cells where the number of contained points is less than \( \text{minPts} \). Finally, we extend each quad-cell by \( 1-\varepsilon \) distance on each border. One benefit of this extension is to find overlapping areas between cells so that a merge phase can be applied at the end of each tick. Another purpose is to get the nearby points context for each cell since some essential contexts can be carried in dropped cells that need to be re-used.

In Figure 6.4, there are two kinds of thresholds used for dividing the space. The space is iteratively divided into 4 cells until either the number of points within the current cell is less than 6 or the minimum border of the current cell is less than \( 2\varepsilon \).

The first condition, called \( \text{maxPts} \), is to prevent a single cell from having...
Figure 6.4: Illustration of the Fast Clustering Partition Method used by RT-DBSCAN

too many points to process. The value 6 for $maxPts$ in Figure 6.4 is only used for demonstration purpose. In real-case, $maxPts$ must be greater or equal to $minPts \times 2^n$, where $n$ is the number of dimensions in FC partitioning. This is to avoid over-partitioning a potential cluster. The second condition, called $minSize$, is used to prevent the partitioned cells from having very small sizes. If the cell size is less than $1-\epsilon$, points that are potentially in the same cluster will likely be divided into separate cells. Although they will be merged and sorted out in the final merging task, this can lead to a large amount of work for merging and thus be very inefficient. These thresholds are free to be customized depending on the specific cases. After the space is divided into a quad-tree, certain cells are dropped if either the number of points within the cell is less than $minPts$ or no new input points fall inside this cell. The propose of this dropping is to reduce the number of parallel tasks/partitions. The first condition helps to drop cells with very limited numbers of points. It is predetermined that points inside this cell cannot form a cluster. Hence these cells can be dropped. However, some of
the dropped historical points can contain essential-nearby-contexts of new input points within other target cells, and therefore, these dropped points need to be recovered when those target cells are extended in the following step.

The second condition is to overcome the flaw of $\text{nearbyCtx}(iPs, cPoints)$ mentioned in the previous section, where unnecessary historical data in these blank cells can be dropped to reduce the workload. If the input spatio-temporal data arrives in arbitrary order, the single bounding box for the nearby context could be huge in size. Although dropping ‘blank’ cells can filter out non-necessary historical data before starting DBSCAN, it can still generate significant workload through the I/O and FC partitioning. Using multiple discrete bounding boxes can be a solution for such cases. In this work, we only use a single bounding box which is more suitable for data that arrives in (time) sequence. After dropping ‘blank’ cells, all remaining cells (the red rectangle in Figure 6.4) now meet both of the following conditions: they have more than one new input data, and they have enough points to form at least one cluster (regardless of whether they are historical or new points).

Finally, the cells can be extended by $1-\epsilon$ distance (the green rectangle in Figure 6.4). As mentioned above, the purpose of this extension is to identify overlapping areas when merging cells and pick up lost contexts during the cell dropping process. Figure 6.5 illustrates some of these scenarios. The pseudo-code of the FC partition approach is given in Algorithm 2. If data are in a high-dimensional space, this method can be adjusted by dividing the space based on multiple dimensions.

Iteratively dividing a space into 4 cells is a naive version of FC partitioning in 2D space. This approach suffers from dividing flat rectangle shaped cells, i.e. partitioning can stop in the first iteration when the smallest border of a flat rectangle reaches the threshold. This problem is solved by dividing each cell into $2^{(n-m)}$ sub-cells. $n$ is the number of total dimensions and $m$ is the number of dimensions which their corresponding borders reach when the size threshold is
met. After this improvement, the partitioning is driven by each dimension and its corresponding border of the target cell.

After the FC partition method, those points in \( iPs \cup nearbyCtx(iPs, cPoints) \) are divided into two groups: \( aPts \) where each point belongs to one or multiple cells and the group \( dPts \) where each point does not belong to any cell. Only \( aPts \) will apply parallel local DBSCAN for each cell. This procedure is marked as \( PCluster(aPts) \). The result of \( PCluster(aPts) \) may contain duplicated points, i.e. points belonging to duplicate cells. This result will be union-ed with \( dPts \) before being merged/cleaned. All points in \( aPts \) and \( dPts \) will be persisted at the end of this tick.

**Local DBSCAN for Each Partition**

Points inside each partition can contain both new input and historical clustered data. An incremental DBSCAN approach is applied to those points for each partition as shown in Algorithm 3. This incremental DBSCAN (derived from point-by-point clustering) can generate duplicated data belonging to different
6.3 Algorithm

**Algorithm 2** Fast Clustering (FC) partitioning in RT-DBSCAN

**Input:**
- RS — The root space to be partitioned;
- points — An array of vectors representing the location of each point in the root space;
- minPts — The minimum points parameter used in DBSCAN;
- minSize — The minimum size for output partitions;
- maxPts — The maximum number of points allowed in each partition.

**Output:**
- partitions — An array of partitioned space.

```plaintext
1: function Partition(RS, points)
2:     partitions ← {}
3:     Divide(RS, points, partitions)
4:     return partitions

1: procedure Divide(cell, cellPts, partitions)
2:     if cellPoints.length < minPts then
3:         return
4:     if all the points in cellPoints are historical data then
5:         return
6:     if cellPts.length > maxPts && cell.size > minSize then
7:         subCells ← equally split cell into 2 or 4 sub-cells
8:         for Cell c ∈ subCells do
9:             pts ← fetch points in c from cellPoints
10:            Divide(c, pts, partitions)
11:     else
12:         extCell ← extend the boundary of cell by 1-ε distance
13:         partitions ← Add extCell into partitions
```

clusters. Since the final merging procedure handles this problem, duplicated points are not fixed in the local partition.

The input parameter `distance(p_i, p_j)` of Algorithm 3 is a customizable function for calculating the distance between two data points in multi-dimensional spaces. A spatio-temporal distance function is created for clustering social-media data in our implementation. This spatio-temporal distance is given in Equation 6.1, where \( P_i \) and \( P_j \) are vectors representing two spatio-temporal data, e.g. Tweets. \( x \) and \( y \) in vector are values of GPS information, e.g. longitude/latitude and \( t \) is the time-stamp value. This equation is based on Euclidean distance. A customized
s is used to convert the temporal value $t$ into a spatio-value, so that all spatio-temporal values, i.e. $x, y, t$ can have the same unit in the distance calculation.

$$P_i = (x_i, y_i, t_i), P_j = (x_j, y_j, t_j)$$

$$D_g = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \triangle t = \lvert t_i - t_j \rvert$$

$$Distance = \sqrt{D_g^2 + (\triangle t \times s)^2}$$
Algorithm 3 Local Incremental DBSCAN for each Partition in RT-DBSCAN

**Input:** points — An array of point data which contain their location information; \( \epsilon \) — The distance parameter used in DBSCAN; minPts — The minimum points parameter used in DBSCAN; distance\((p_i, p_j)\) — A customizable function which returns the distance from \( p_i \) to \( p_j \) in multi-dimensional space.

**Output:** uPoints — An array of updated/clustered point data.

```plaintext
1: function Local-DBSCAN(points, \( \epsilon \), minPts)
2:   uPoints \leftarrow \{\}\n3:   for Point \ p \in\ points do
4:     if p.clusterId is NULL then
5:       ExtendFromSeed(p, \( \epsilon \), minPts, points, uPoints)
6:     else
7:       uPoints \leftarrow \text{Add} \ p \ \text{into} \ uPoints
8:   return uPoints

1: procedure ExtendFromSeed(seed, \( \epsilon \), minPts, points, uPoints)
2:   nbhdQ \leftarrow \text{getNeighbors}(seed, points, \( \epsilon \), distance\((p_i, p_j)\))
3:   if nbhdQ.length > minPts then
4:     newId \leftarrow \text{generateUUID}()
5:     seed.clusterId \leftarrow \text{newId}
6:     seed.label \leftarrow \text{CORE}
7:     uPoints \leftarrow uPoints.add(seed)
8:   for Point \ nb \in\ nbhdQ do
9:     if \ nb.clusterId is NULL then
10:        nb.clusterId \leftarrow \text{newId}
11:        nb.label \leftarrow \text{BORDER}
12:        uPoints \leftarrow uPoints.add(nb)
13:        ExtendFromSeed(nb, \( \epsilon \), minPts, points, uPoints)
14:   else if \ nb.label is CORE then
15:     nbCpy \leftarrow \text{nb.clone}()
16:     nbCpy.clusterId \leftarrow \text{newId}
17:     nbCpy.label \leftarrow \text{BORDER}
18:     uPoints \leftarrow uPoints.add(nbCpy)
19:   else if \ nb.label is BORDER then
20:     nnbhdQ \leftarrow \text{getNeighbors}(nb, points)
21:     nbCpy \leftarrow \text{nb.clone}()
22:     if \ nnbhdQ.length > minPts then
23:        nbCpy.label \leftarrow \text{CORE}
24:     nbCpy.clusterId \leftarrow \text{newId}
25:     uPoints \leftarrow uPoints.add(nbCpy)
26:   else
27:     uPoints \leftarrow uPoints.add(seed)
```

Merging the Results of Parallel Clustering and Ensuring Consistency with Historical Data

After the parallel local DBSCAN finishes, the result sets ($U_r$) are union-ed with $dPts$, i.e. $U_r = PCluster(aPts) \cup dPts$.

The structure of the data in $U_r$ is given as a vector by $<\text{pointId}, \text{clusterId}, \text{clusterLabel}, \text{ptDetails}>$. The pointId is a unique id for each point. The clusterId is a unique id for the cluster that this point belongs to. The clusterLabel is the role of this point in a cluster, which can be either ‘NOISE’, ‘CORE’ or ‘BORDER’. The ptDetails contains the details of this point including its location, timestamp, and other data fields. After applying the previous procedures, there may be duplicated points in $U_r$. For example, in Figure 6.5 point $P$ is in the overlapped area and thus it exists in two partitions. After applying local DBSCAN on each partition, one instance of $P$ is a ‘NOISE’ point and another instance belongs to a cluster. In this final procedure, these kinds of inconsistencies are handled by a merge function: $U_q = \text{merge}(U_r)$. Points inside $U_q$ are ensured to be unique. Two kinds of merging solutions are supported.

Merging Approach A: Solving inconsistencies where a single point belongs to multiple clusters

If a point belongs to multiple clusters, the use of this point is critical. When a point $P$ has duplicates: if all instances of $P$ are ‘NOISE’ then we keep one of them and drop the others. If all non-noise instances of $P$ belong to the same cluster $A$, then we create a singleton $^1$ of $P$, mark it as a member of the cluster $A$ and merge its roles in cluster $A$ using the priority order: CORE > BORDER > NOISE. If non-noise instances belong to multiple clusters, then we create a singleton of $P$ and merge its roles among multiple clusters using the priority order: CORE > BORDER.

$^1$In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one single instance
If the merged role is ‘BORDER’, where a border point can belong to multiple clusters we add \( \textit{clusterIds} \) into a list and attach it to the new singleton.

If the merged role is ‘CORE’ then the point is a solid joining point for multiple clusters. All such clusters must be merged into one cluster. To do this, we create a singleton of \( P \) and mark it as ‘CORE’. We then randomly pick a cluster \( A \) from the non-noise instances and attach it to the singleton and then identify members in other clusters to be merged. We then change their \( \textit{clusterId} \) to cluster \( A \).

Consider the scenario where cluster \( C_1 \) and \( C_2 \) are found in duplicates of point \( P_i \) and \( C_2 \) is picked as the target to merge (i.e. \( C_1 \rightarrow C_2 \)). Later cluster \( C_2 \) and \( C_3 \) are found in duplicates of point \( P_j \) and \( C_3 \) is picked as the target to merge (i.e. \( C_2 \rightarrow C_3 \)). Merging the clusters in this cascading manner is inefficient. A merging task for \( n \) clusters is not simple since it needs to traverse all \((n - 1)\) clusters that are inside. To maximize the efficiency, a global conversion table, as shown in Figure 6.6, needs to be generated before actually starting to merge the clusters.

![Figure 6.6: Procedure for Generating a Global Cluster Conversion Table](image)

**Merging Approach B: Solving Conflicts between Retrieved Historical Data and non-retrieved Historical data**

After getting the global cluster conversion table, points in \( U_r \) are traversed and their \( \textit{clusterId} \) changed according to the conversion table. However, what if some to-be-merged cluster points are not in \( U_r \)? Data in \( U_r \) come from \( iPs \cup \)
nearbyCtx(iPs, cPoints). There is no guarantee that this set covers all the cluster members. Figure 6.7 illustrates this issue.

**Figure 6.7:** Illustration of the Pitfalls in Merging Clusters

In Figure 6.7, two triangle-shaped points are not merged because they are absent in $U_r$. Since the clusterIds of those potential absent points are known, according to the cluster conversion table, all points in the to-be-merged clusters can be retrieved from cPoints. This procedure is given as:

$$U_c = $getClusterPoints(ConvTbl)$$  \hspace{1cm} (6.2)

A union operation can then be applied on $U_r$ and $U_c$. Cluster merging is finally applied on this combined set. The merge function can be rewritten as shown in Equation 6.3.

$$CvtTbl = getConTbl(U_r)$$

$$U_c = getClusterPoints(CvtTbl)$$

$$U_q = merge(U_r)$$

$$= mergeClusters(U_c \cup U_r, CvtTbl)$$

(6.3)
Algorithm 4 presents the whole merging procedure of RT-DBSCAN. Almost all the merging code can be executed in parallel except the procedure for building a global cluster conversion table. Each computation node needs to report their discoveries to a centralized node for generating the global conversion table. After the above procedures (i.e. partitioning; clustering; merging), the output set $U_q$ contains unique points with their up-to-date cluster information. Finally, $U_q$ should be persisted before a next tick starts.
Algorithm 4 Partitions merging in RT-DBSCAN

**Input:** \( U_r \) — A joint array with duplicated points; \( minPts \): DBSCAN parameter; \( cPoints \): The historical persisted dataset

**Output:** \( U_q \) — A merged array with distinct points and their up-to-date cluster information

```
1: function getCvtTbl(U_r) ▷ Functions for getting the conversion table
2:   cluPts ← {}
3:   for Point p ∈ U_r do
4:     if p.clusterId is not NULL then
5:       cluPts ← cluPts.add(p)
6:     ptMap < p1d, Pts > ← cluPts.aggreByPointId()
7:   cluIdSets ← {}
8:   for Entry < p1d, Pts > ∈ ptMap do
9:     cluIdSet ← {}
10:    core ← NULL
11:    for Point p ∈ Pts do
12:       cluIdSet.add(p.clusterId)
13:       if p.label is CORE then
14:          core ← p
15:    cluIdSet ← cluIdSet.distinct()
16:    if core is NULL && cluIdSet.size() > minPts then
17:       core ← Pts.get(0) ▷ If multiple Border roles to different clusters
18:    if core is not NULL then
19:       cluIdSets.add(cluIdSet)
20: CvtTbl < fromId, toId > ← Empty
21: while cluIdSets.size() > 0 do ▷ This while loop needs to be executed in a single process
22:   seedSet ← cluIdSets.remove(0)
23:   toId ← seedSet.get(0)
24:   toRmSets ← {}
25:   for Set candiSet ∈ cluIdSets do
26:     if seedSet intersect candiSet then
27:       seedSet.addAll(candiSet)
28:       toRmSets.add(candiSet)
29:     if Element fromId ∈ seedSet do
30:       CvtTbl.put(fromId, toId)
31:   cluIdSets ← cluIdSets.removeAll(toRmSets)
32: return CvtTbl
```
6.4 RT-DBSCAN Implementation

The RT-DBSCAN algorithm proposed above has been implemented using Apache Spark Streaming leveraging the SMASH platform described in Chapter 5. As described, SMASH is a platform for hosting and processing historical data with Apache Spark capabilities. In deploying the RT-DBSCAN on SMASH, Kafka is used as the streaming source for feeding new data. GeoMesa over Accumulo provides a distributed geo-spatial database. It is used for storing and querying historical (cluster) data. Spark Streaming is a framework for building streaming applications on Apache Spark. This framework treats data streams as ticks of chunks and executes micro-batch tasks for each tick of data. Spark itself has many other interfaces for running MapReduce functions. These functions suit the needs

Algorithm 4 Partitions merging in RT-DBSCAN (continued)

33: function MERGER($U_r$
34:  $CvtTbl \leftarrow \text{GETCvTBL}(U_r)$
35:  $U_c \leftarrow \{\}$
36:  for Key $fromId \in CvtTbl$ do
37:    $Pts \leftarrow \text{QUERYBYCLUSTERID}(cPoints, fromId)$
38:    $U_c \leftarrow U_c.addAll(Pts)$
39:  $U \leftarrow U_c \cup U_r$
40:  $ptMap< pId, Pts > \leftarrow U.aggreByPointId()$
41:  $U_q \leftarrow \{\}$
42:  for Entry $< pId, Pts > \in ptMap$ do
43:    $pt \leftarrow \text{new instance of point}$
44:    $pt.id \leftarrow pId$
45:    $pt.label \leftarrow \text{MERGEROLES}(Pts) \triangleright \text{CORE} \triangleright \text{BORDER} \triangleright \text{NOISE}$
46:    $cId \leftarrow Pts.get(0).clusterId$
47:    for Instance $p \in Pts$ do
48:      $cId \leftarrow CvtTbl.GET(p.clusterId)$
49:      $pt.clusterId \leftarrow cId$
50:      $U_q \leftarrow U_q.add(pt)$
51: return $U_q$
of RT-DBSCAN and save a lot of work in implementing the clustering algorithm.

Twitter data (tweets) are used for the case studies and benchmarking of the platform. Figure 6.8 provides an illustration of how data are tackled tick-by-tick in 3D space. We assume all incoming data are within a static spatial area and this area is used as the base of all bounding boxes for getting the nearby context. This assumption is made to simplify the bounding box in getting the nearby points contexts for this demonstration. In Figure 6.8, the cube with the red border is one example of this bounding box. It has a static base with dynamic height and position. This is a simplified version of $\text{nearbyCtx}(iPs, cPoints)$ in 3D space and used for demonstration purposes. In the real world (applications), the bounding box can have a random size and position. In addition, a 2D FC partition can be adapted into a 3D space version, e.g. divide each cell into (up to) eight sub-cells according to three dimensions.

**Figure 6.8:** Illustration of a Simplified $\text{nearbyCtx}(iPs, cPoints)$ in 3D space

Figure 6.9 illustrates the procedure of realizing RT-DBSCAN using Spark nodes and Spark Streams as the framework for tackling data streams as a series of data chunks provided in given ticks. An inner workflow for each tick is also presented in this figure. At the beginning of each tick, a FC partition is
applied against incoming data chunks on a single master node. Data shuffling (i.e. sending data to the node which holds the cell it belongs to) and local DBSCAN are executed in parallel on each worker node. A cluster merging table is generated on the master node after all local DBSCAN processing completes. Finally, result merging and data persistence are handled in parallel on worker nodes. A new tick procedure then starts after the results of the previous tick have been persisted.

Figure 6.9: Illustration of the RT-DBSCAN Procedure Using Spark Streaming
6.5 Case Study Benchmarking RT-DBSCAN using Spark Streams

This section presents a case study and the benchmarking of real-time social-media clustering using RT-DBSCAN on the SMASH platform.

6.5.1 Cloud Resource Usage

The SMASH platform for this case study was deployed on the NeCTAR research cloud. Fifteen computation nodes (Virtual Machines) from the Melbourne zone were used to form the core infrastructure for the case studies on RT-DBSCAN. The specification of each node was as follows:\(^2\):

- one master node: VCPUs @2.60GHz \(\times\) 4 ; 12GB RAM; 120GB HDD;

- thirteen slave nodes: VCPUs @2.60GHz \(\times\) 2; 8GB RAM; 70GB HDD;

- one interface node: VCPUs @2.60GHz \(\times\) 4; 16GB RAM; 130GB HDD.

The results of I/O benchmarking on the computational resources were as follows:

- Bi-directional network bandwidth: 7.88 ± 1.03 Gbits/sec;

- Cached reads rate: 3036.1 ± 67.1 MB/sec;

- Buffered disk reads rate: 134.6 ± 18.1 MB/sec, and

- Disk write rate: 599.7 ± 39.1 MB/sec.

\(^2\)It is noted that the work was based on a limited amount of resources on the NeCTAR Cloud and this scale (1 master, 1 interface and 13 slaves) was the largest allowed cluster.
6.5.2 Software Stack and Dockerized Deployment

The software stack and dockerized deployment were the same as the original SMASH platform described in Chapter 5. However, there are several specific configurations:

- At the Application layer: GeoServer (v2.9.4) and Apache Kafka (v0.11.0.0) were deployed with default settings. GeoMesa (v1.3.2) data store plugin was installed alongside the GeoServer for fetching spatio-temporal data layer from the Storage layer.

- At the Computation layer: Apache Spark (v2.1.1) cluster with one master node and thirteen slave nodes was deployed. The Spark cluster was run in its Standalone Mode. Each Spark worker (slave) node had 2 cores and 4GB memory.

- At the Data Storage layer: Hadoop Distributed File System (HDFS) (v2.7.4) was installed as the file system for SMASH. Its block replication was set to 2. sync.behind.writes and synconclose were enabled to ensure data was written immediately into disk. Apache Accumulo (v1.8.1) was deployed over HDFS as a key/value data store. GeoMesa (v1.3.2) was deployed on top of the Accumulo cloud data storage.

6.5.3 Stream Data Source for Case-studies

Twitter data (tweets) were used as the data source for the benchmarking cases studies. Not all tweets data contain GPS information as introduced in Section 3.1.1. Those non-GPS tweets were filtered out before being fed into Spark Stream. We collected a fixed size of tweets data which were post by Twitter users in Melbourne within 6 months in 2015. This dataset contained 604,529 tweets ($\approx 220MB$) with GPS and timestamps. These were harvested through the Twitter API (https:
Real-time density-based Clustering on SMASH

//developer.twitter.com) and stored in a file with arbitrary order. An application was built for reading this data-set and generating the actual data stream. This generator controlled the output rate of the stream and pushed it into the Kafka service on the SMASH platform. Kafka accepted this stream and published a topic for this data source after filtering out potential noisy data (e.g. tweets without GPS or time-stamp values). The stream application running on Spark listens for this topic, obtains continuous data streams from Kafka, applies the RT-DBSCAN algorithm on this stream and persists/updates the results on the GeoMesa/Accumulo clusters of the SMASH platform. Through this pipeline, the performance of RT-DBSCAN using Spark Streaming under different data velocity and volume scenarios was explored and benchmarked.

6.5.4 RT-DBSCAN Performance Benchmarking

Figure 6.10 illustrates a series of checkpoints generated by RT-DBSCAN (on-the-fly). The results are visualized by GeoServer on SMASH using OpenStreetMap (https://www.openstreetmap.org) as the base map. Each red point on the maps of Figure 6.10 represents a noise point which does not belong to any clusters. Each larger green point represents an in-cluster point, i.e. a point that is inside a cluster. RT-DBSCAN can start either from a blank or using existing cluster results as the initial starting point (checkpoint). Incremental updates are then conducted at each tick (corresponding to micro-batches in Spark Streaming) at which point they handle new inputs and generate up-to-date checkpoints based on previous checkpoints.

RT-DBSCAN on Spark Streaming is based on a “micro-batching” architecture. A streaming engine is built on the underlying batch engine, where the streaming engine continuously creates jobs for the batch engine from the continuous data stream. There are two important concepts in this architecture. One is the “Batch Interval Time” (BIT) which is a fixed interval value decided by the user of Spark
Figure 6.10: A Series of Visualisations of Cluster Checkpoints Generated by RT-DBSCAN

Streaming. This value controls the intervals used for generating micro-batch tasks for the underlying batch engine, i.e. the tick interval of RT-DBSCAN. Another concept is the “Batch Processing Time” (BPT) which is the execution time for each batch task. Ideally, a Spark Streaming application needs to ensure that a batch task is completed before the next batch is queued. If the processing times of arriving batches continuously takes longer than the batch interval time then a “snowball effect” can take place. This effect can eventually exhaust Spark resources. Depending on the settings, Spark may discharge this pressure by killing the application or by pushing this pressure back to the source of the data — Apache Kafka. Therefore, it is important to ensure that for most of the batches, $BPT < BIT$. 
The scheduling delay $\tau_i$ for each batch $i$ in a given sequence is defined in Equation 6.4, where $\delta_i = BPT_i - BIT_i$.

$$\tau_i = \begin{cases} 0, & \text{if } i = 0 \\ \tau_{i-1} + \delta_i, & \text{else if } \tau_{i-1} + \delta_i > 0 \\ 0, & \text{else} \end{cases} \quad (6.4)$$

The total delay $T_i$ for each batch $i$ in a given sequence is defined as $T_i = \tau_i + BPT_i$. The average processing delay $v_i$ for each arriving input data involved in batch $i$ can be estimated by $v_i = T_i + \frac{BIT_i}{2}$, where $BIT$ is a fixed configurable parameter used by Spark Streaming. When $\tau_i = 0$, the system achieves its optimal performance. If the values of $\tau_i$ and $T_i$ keep increasing, a “snowball effect” occurs and the system is considered unstable against the input stream. On the other hand, if the value of $\tau_i$ and $T_i$ are stable under a given threshold, this system is considered stable. In the following case studies, we benchmark the RT-DBSCAN on Spark Streaming by using different numbers of Spark executors and different input rates for the data streams. $\tau_i$ and $BPT_i$ are monitored and used for evaluating the stability and performance of RT-DBSCAN. The default parameters used in the following RT-DBSCAN benchmarking are set as follows:

- **DBSCAN parameters:**
  - $\epsilon$ — A spatio-temporal distance threshold calculated by Equation 6.1 at page 126 based on the inputs: $D_g = 100m; \triangle t = 600sec; s = 1.667m/sec$. Social media data posted at the same time within 100 meters are considered as neighbors, since our acceptable geo-location error was set to $\pm 100$ meters as discussed in Section 3.1.2. 600 seconds (10 minutes) is an experimental parameter used as the time buffer. 1.667$m/s$ is used to describe the speed of a virtual vehicle traveling at 60 km/h.
\( \text{minPts} = 3. \)

- **non-DBSCAN parameters:**
  - FC partition configuration: \( \text{maxPts} = 100; \text{minSize} = 2 \times e. \)
  - Tick time (\( BIT \)) = 30 seconds.

These default DBSCAN parameters are identical to the case studies presented in Chapter 7. The selected value for the spatial-temporal ratio \( s \) reflects the walking speed of an individual. The source code of the RT-DBSCAN implementation using Spark Streaming is available at our GitHub repository — https://github.com/project-rhd/smash-app/tree/master/smash-stream/.

**Benchmarking the Scalability of RT-DBSCAN**  Figure 6.11 presents the results of benchmarking the scalability of RT-DBSCAN. The data input rate here is set to 1,000 points per second. Performance on different numbers of Spark executors is benchmarked under the same data stream. The two charts in Figure 6.11 show the timeline on the \( x \) axis and \( \tau_i, BPT_i \) on the \( y \) axis. According to the charts, the first batch usually takes a longer time to process. This is because several initiations are conducted at the beginning of the first tick, e.g. database connections. Almost all the \( BPT_i \) of 1 \( \text{node} \times 2 \text{cores} \) are larger than \( BIT \) and thus \( \tau_i \) keeps growing. This pattern means that 1 \( \text{node} \times 2 \text{cores} \) is not stable with the default parameters in tackling this rate of the input data stream. As shown with the different configurations, 3 \( \text{nodes} \times 2 \text{cores} \) is the most stable and efficient scale among the candidates shown in Figure 6.11 since its \( \tau_i \) reaches zero after several batches from the initiation. The delay of the configuration with 6 \( \text{nodes} \times 2 \text{cores} \) and 12 \( \text{nodes} \times 2 \text{cores} \) are even larger than 1 \( \text{node} \times 2 \text{cores} \) at the beginning of processing. However, the trend of their \( \tau_i \) stabilizes and their \( BPT_i \) wavers near the \( BIT \) line. These two scales are considered stable under this data rate but they are not the most optimal option. The overhead of network I/O among multiple nodes is the major bottleneck with this number of nodes. To conclude, scaling the
number of executors of RT-DBSCAN has a positive effect on its performance but having too many nodes can impact the efficiency due to the data transmissions required over the network.
Figure 6.11: Benchmarking Scalability under Input Rates = 1,000 points/sec
Benchmarking with non-DBSCAN Settings  In addition to the number of Spark executors, several non-DBSCAN parameters can impact the efficiency and stability of RT-DBSCAN under high data velocity situations, e.g. FC partition parameters and the BIT (tick time).

Figure 6.12: Performance Benchmarking using Different maxPts Settings

Figure 6.12 shows the results of benchmarking the performance of RT-DBSCAN according to maxPts which is a parameter/threshold used in the FC partition method. A 1 node × 2 cores Spark cluster is used and the input rate is 600 points/sec. maxPts has a direct impact on the number of points in each cell and the number of data partitions needed for parallel computing, i.e. it impacts on the granularity of parallelization. As seen, maxPts = 200 is the optimal setting for the cluster among all candidates. We also find that the value of maxPts does not have a significant impact on the performance (delay) of the system.

Figure 6.13 benchmarks the performances on BIT i.e. the tick time. A 1 node × 2 cores Spark cluster is used and the input rate is 600 points/sec. As seen, BIT
A larger value of BIT can make the system more stable under higher input rates. However, increasing BIT will improve the average delay for processing each input data. The BIT defines how frequent a new checkpoint is going to be generated. Therefore, having an elastic and flexible BIT is a good strategy to balance the stability and output delay of RT-DBSCAN.
6.6 Conclusions

In this chapter, we proposed a new variation of the DBSCAN algorithm (RT-DBSCAN) for clustering spatio-temporal data targeted specifically to real-time data streams. The novelties of this algorithm are that it tackles ever-growing high velocity data streams whilst utilizing density-based data clustering. Furthermore, the algorithm does not assume that spatio-temporal data arrives in a given time-based sequence.

In the benchmarking, we identified and discussed several configurations that were explored for the performance of RT-DBSCAN using Spark Streaming on the SMASH platform. The benchmarking gives confidence that the algorithms can scale to meet real life high-velocity and bursty traffic flows. In one case-study benchmark, RT-DBSCAN was able to tackle data streams at over 10,000 social media posts per second on SMASH.

In summary, RT-DBSCAN solved one of the two challenges identified in Section 4.7. Practical experiments exploring the correlation between social media data and urban traffic issues (i.e. the other challenge mentioned in Section 4.7) form the basis for the following chapter.

This chapter is largely based on the publication:

In the previous chapter, we explored one of the questions raised from the research hypothesis in Section 1.5.3, i.e. new algorithms for real-time urban traffic analytics suitable for real-time spatio-temporal data.

In this chapter, we present case studies utilising the SMASH platform to answer the other question: is it possible to use social media data as a proxy for traffic data analytics, instead of the more expensive official traffic data from sensor networks such as SCATS?
7.1 Data Description

This section covers the details of data used for the case studies utilising SMASH. Whilst some of the data sets like SCATS were introduced in the previous chapters we describe how they are pre-processed, stored and indexed on the SMASH platform before ultimately being analyzed in combination with social media data.

7.1.1 SCATS Traffic Data

The SCATS sensor network is deployed on the main roads in the major cities of Australia. It provides an official source of urban traffic data as discussed in Section 1.5.2. The exact dataset used in this chapter is the SCATS Traffic Volume data set. This was introduced in Section 5.5.1. This data needs to be pre-processed and ingested into the SMASH platform as described in Section 5.5.2. For each sensor deployed on a single lane of a road, there is an entity in the dataset that records the volume (number) of vehicles that pass over this road section every fifteen minutes. An example of the SCATS Traffic Volume data tuples are shown in Table 5.1.

Different to the SCATS data set used in Chapter 5 for platform benchmarking purposes, the SCATS Traffic Volume data set used in this chapter is used for analytical purpose. This data set is about 7GB in size (plain-text format). It covers the urban traffic volumes that are recorded and subsequently collected from the SCATS sensor network. These sensors are primarily deployed on the major roads of Victoria. The data collected throughout 2017 are used here. The size of this dataset is not excessively large considering that it represents a year-long state-coverage data set. The vast majority of data are located in/near the CBD since the density of the deployed SCATS sensors in this area is much higher than in the suburbs as illustrated in Figure 5.9. In this chapter, we choose a defined bounding-box of Melbourne as the study area for data processing. This Melbourne area is defined as a joint bounding-box of 4 geo-hashed areas [194],
which can be presented as strings: ‘r1q’, ‘r1r’, ‘r1n’ and ‘r1p’.

The SCATS Traffic Volume data are stored in the Accumulo/GeoMesa cluster of SMASH. Each stored SCATS entity contains general information of the associated SCATS sensor and the recording of traffic volumes in a 15-minute window (96 such windows per day). The geographic information of each entity has two spatial forms: a point and a line/polyline segment. The point represents the precise location of the SCATS sensor while the line/polyline represents the road segment where the sensor is installed. A combination of the installation/location ID, sensor ID, and the timestamp of each entity are collectively used as the primary key when storing the data in the database cluster. The spatio-temporal information (i.e. point, polyline and timestamp) are indexed by using the Z3/XZ3 index. The Z3 index utilizes a three-dimensional Z-order curve [129] to index latitude, longitude, and time for point data, while the XZ3 index is a three-dimensional implementation of XZ-ordering [25] to index the latitude, longitude, and time for non-point data. These indexing implementations provide efficiency in answering queries with both spatial and temporal components, i.e. high-frequency spatio-temporal queries from the analyzers running on the SMASH platform.

### 7.1.2 Social Media Data

Twitter and Instagram data were selected as the representative geo-spatially coded social media data used this chapter. As discussed in section 1.5.1, both Twitter and Instagram are widely used by people globally and both have or had public APIs for accessing and using their data. It is noted that the Instagram API has since been restricted as a result of the Cambridge Analytica privacy issue that took place with Facebook [44]. Tweets and Instagram posts created in the defined Melbourne area (see Section 7.1.1) between 2017-06-01 to 2017-12-31 were collected. These were then filtered to only include those data with GPS tags before storing them into Accumulo/GeoMesa. There were 132,927 entities
(tweets/posts) that were finally stored on SMASH as the data resource for the case studies. These social media data were converted into point-shaped spatio-temporal data during pre-processing. Similar to the indexing of SCATS data, all the social media data are indexed using the Z3 index in the GeoMesa/Accumulo cluster.

![Figure 7.1: An Overview the Processed Social Media Data Displayed on a Map](image)

### 7.2 Observations on the Social Media Clusters via SMASH

In this section, we stream the social media data described in Section 7.1.2 into the RT-DBSCAN processors running on the SMASH platform. The implementation details of this procedure are the same as the RT-DBSCAN case study presented in Section 6.5. Following this clustering procedure, a new attribute — “cluster id” is
attached to each of the input social media entities. That is, those entities that are spatio-temporally correlated, e.g. close to each other, and hence have the same “cluster id” assigned (i.e. form a cluster). This is a unique identifier created for each identified spatio-temporal cluster. Those noise entities that do not belong to any clusters are not assigned any cluster identifiers hence have a null for their “cluster id”.

The clustered social media data results are stored in the Accumulo/GeoMesa database as a new table. The GeoServer associated with the SMASH platform can directly read and add to this table, e.g. as a geographic data layer within its resource pool. A standard GIS Web Map Service [149] (WMS) is used for visualizing this social media data layer on any map based clients.

Figure 7.1 presents an overview of the processed social media data in the Melbourne area displayed on one exemplar Web client. Here the green points are “in-cluster” entities and the red points are noise entities. By removing the noise entities, Figure 7.2 presents an overview of the cluster positions that were identified in the Melbourne area. As seen, most of the clusters (over 90%) are
located in/near the Melbourne central area. The high population in these busy areas leads to more human mobility data (e.g. GPS tagged social media data) that accordingly give rise to more spatio-temporal data clusters. To compare the spatio-temporal patterns between social media data and more official urban traffic in the following sections, we focus on a small area around the Melbourne CBD and its surroundings. The impact of social media data for traffic modelling in less densely populated areas is discussed further in Chapter 8.

Figure 7.3: A given detected social media cluster in/near Melbourne Cricket Ground on 17/06/2017 at around 2pm (local time). The blue points indicate the location of each cluster member.

It can be the case that many of the spatio-temporal clusters can be linked to known/understandable events or scenarios, especially when the information of data contents and nearby urban landmarks are taken into considerations. Figure 7.3 illustrates a typical detected event based on the information extracted
from the social media data. This social media cluster is located in the boundary of the Melbourne Cricket Ground (MCG) and it contains sixteen entities (i.e. blue points in Figure 7.3) which are created by sixteen distinct social media accounts on 17/06/2017 from 2:52pm to 4:21pm (Melbourne local time). Based on the location of this cluster and the contents from each cluster member, it is possible to ascertain that an Australian football match happened at the MCG at this time. This was confirmed [1] by providing these information to a web search engine [27] (Google).

![Figure 7.4](image-url)  
**Figure 7.4:** A given detected social media cluster near the Victoria State Library on 26/08/2017 at around 5pm (local time). The blue points indicate the location of each cluster member.

In addition to those clusters with the same topic in content, as shown in Figure 7.3, there are many clusters where users talk about different topics. Figure 7.4 illustrates an example of this kind of social media data cluster. This cluster contains twenty four social media entities presented by the blue points (overlapped)
in the figure. These entities were located near Victoria State Library in Melbourne CBD on 26/08/2017 at around 5pm. The majority of these entities were talking about a protest for same-sex marriage, while a few entities were talking about shopping or university studying. The square in front of Victoria State Library is a usual place for protesting in Melbourne. Also 2017 was the year that same-sex marriage became legal in Australia. In addition, there is a large shopping mall (i.e. Melbourne Central Shopping Centre) and a University (i.e. RMIT University) near the State Library. It is reasonable to infer that the appearance of this cluster is mainly caused by a given protest. Such events (e.g. sports matches, citizen protests) are likely to have a given impact on the “usual” urban traffic because they normally involve a large number of irregular population patterns and associated commuting behaviour.

To understand this, we focus on the connections among social media clusters, urban events and urban traffic phenomenon. Importantly, since SMASH and RT-DBSCAN as introduced in Chapter 5 and Chapter 6, allows us to identify these social media data in (near) real time, we hypothesise that it should be possible to identify those events and related traffic issues together.

It is also noted that not all the social media clusters can be directly linked to any describable events. Figure 7.5 illustrates an example of this kind of cluster. It is a cluster with seven social media entities all located in the Melbourne CBD on 08/08/2017 at around 3pm. As discussed in Section 1.5.1, social media users can post random content like a photo with ‘I am here’ or news and advertisements from any official social media accounts as shown in Figure 7.5. Given this, it is hard to extract common knowledge from such content by using tradition natural language processing (NLP)-based methods as described in Section 2.3. However, the spatio-temporal correlation may indicate certain information about crowds and/or events that may take place.
Figure 7.5: A detected social media cluster in Melbourne CBD area on 08/08/2017 at around 3pm (local time). The blue points indicate the location of each cluster member.
7.3 Correlation between Social Media and Urban Traffic Data

To solve the second hypothesis raised in Section 1.5.3, we need to explore the correlation between social media data and more official urban traffic data. The official SCATS Traffic Volume Data is used as the reference for the traffic status.

In Section 7.2 we identified that spatio-temporal clusters of social media data are often linked to certain urban events which usually involve irregular human mobility patterns. This observation gives rise to a key question: can we find certain correlations between social media clusters such as abnormal spatio-temporal distributions and irregular patterns of human mobility such as urban traffic abnormalities? Based on this idea, three sub-questions for solving our hypothesis were identified:

- How to define abnormal patterns of social media data and abnormal patterns of urban traffic volumes?
- For a detected urban traffic abnormality, can we find any corresponding (nearby) social media abnormalities?
- For a detected social media abnormality, can we find any corresponding (nearby) urban traffic abnormalities?

The following subsections introduce three case studies exploring these questions.

7.3.1 Identifying Abnormal Urban Traffic Volumes and Social Media Data

For identifying irregular patterns (i.e. abnormalities) in urban traffic volumes and social media data, we need to ascertain what constitutes regular patterns. We choose localized volumes of social media data and volumes of urban traffic as
the basis for describing regular patterns against potential abnormalities. To do this, we process the SCATS traffic data for Melbourne in 2017 (i.e. the dataset introduced in Section 7.1.2) and generate a baseline for traffic volumes at each sensor spot on each day of the week and at each hour of the day.

![Box plot of traffic volume and nearby (within 1km distance) social media volume on Mondays at Princes Bridge, Melbourne](image)

**Figure 7.6:** Box plot of traffic volume and nearby (within 1km distance) social media volume on Mondays at Princes Bridge, Melbourne

The upper chart in Figure 7.6 illustrates one such baseline data through a box plot. It presents the hourly maximum, minimum and average traffic volume with standard deviations, i.e. the number of vehicles that pass over a given SCATS
sensor on Mondays at a given location - here Princes Bridge in Melbourne. It is noted that Princes Bridge is one of the major road junctions in the Melbourne CBD. In Figure 7.6, rush hour can be clearly observed based on the trends of the average traffic volumes with clear increases between 7-9 am and again from 4-6 pm. These trends can be regarded as regular traffic volume patterns for that location on that day. Those records which are further away from the average value (e.g. the maximum and minimum record) can be regarded as potentially irregular traffic flows, e.g. the lower level may indicate traffic jams (less vehicles crossing the SCATS sensor) and the upper level may indicate an unusual heavy traffic (public transport shutdown and hence an increase in the number of vehicles passing over the sensor). To identify these ranges, standard deviations ($\sigma$) are calculated for the average traffic volumes. Volumes which do not fit within the range $\text{Avg} \pm 2\sigma$ are identified as irregular traffic patterns, i.e. they represent the traffic abnormalities of interest. According to experimental results [6], 95.4% of the data fits within within the range $\text{Avg} \pm 2\sigma$, thus the 4.6% outliers are considered as abnormal.

Similarly, a baseline of social media data is generated based on their distance to the nearby SCATS sensors. Figure 7.6 (bottom) gives an example of the social-media baseline on Mondays at the Prices Bridge junction. This baseline is generated by aggregating (clustering) nearby social media data within 1 km distance to the SCATS sensor. It is noted that this social media data baseline is generated based on the locations of the SCATS data/sensors. When SCATS data are not present, it is reasonable to use the clustering information of social media alone for identifying abnormalities, since both the baseline method and clustering method expose spatio-temporal skews of the social media data distribution.

The two charts in Figure 7.6 tell the story of traffic volumes and nearby social-media volumes related to the same sensor spots and hence at the same location on a given day. There is no obvious relationship between these two volumes that can be gleaned merely by looking at their trends on the graph. Their ups and downs simply infer that people are more active in the daytime at this spot on Mondays.
and social media users are more active between 5 to 6 pm. To explore the deeper connection between traffic volumes and social media, we need to show that when an abnormality of traffic volumes occurs, it can be related to a traffic issue, which can itself be related to some other kind of social public event like a roadshow, a protest, a traffic accident etc. To validate this assumption, we focus on exploring the relationship between the abnormalities of traffic volumes and nearby social-media distributions. We illustrate two case studies in Section 7.3.2 and Section 7.3.3 that explore the spatio-temporal correlation between the presence of urban traffic abnormalities and the presence of nearby social media abnormalities.
7.3.2 Traffic Data-driven Abnormality Analysis

In this section, we introduce a case study to answer the second question raised at the beginning of Section 7.3, namely: for a detected urban traffic abnormality, can we find any corresponding (nearby) social media abnormalities?

To explore this scenario, we undertake several steps:

- **Baseline generation**: To generate the baseline of the regular traffic volume status as introduced in Section 7.3.1, the SCATS traffic volume data in 2017 were aggregated by the day of the week and by each sensor spot. As noted, SCATS recording time slots are based on a 15 minute period with a total of 96 slots per day. Therefore, the generated baseline is a series of 96 records for each sensor location and for each day of week.

- **Traffic volume abnormality detection**: After establishing the baseline data, SCATS records in the same time window, e.g. with the same timestamp and in the same vicinity are aggregated and compared to the baseline. The abnormalities of traffic status are detected when the traffic volume is out of the range of $\text{Avg} \pm 2\sigma$.

- **Looking at nearby social media data**: After processing the SCATS records to identify traffic abnormalities, the analyzer compares the volume of surrounding social-media data to the baseline of surrounding social media data at that location within that day/time window. In this case study, there are two specific conditions used for identifying surrounding social-media data. Social-media data within $1\text{km}$ distance to the location of that SCATS record are considered as surrounding data. Temporally, the timestamp of social-media needs to be within 30 minutes to the timestamp of that SCATS record. The sizes of the physical area and time window are identical to the parameters used in generating the baseline social-media data. In this step, the abnormalities of social-media distribution (against input SCATS
records) are identified when the number of surrounding social-media is out of the range of $\text{Avg} \pm 2\sigma$.

- **Generating report:** After traffic abnormality detection and analyzing the surrounding social media against each input SCATS record, each input can be classified into one of four classifications. They are *abnormal-traffic with abnormal-social-media* (TT), *abnormal-traffic with normal-social-media* (TF), *normal-traffic with abnormal-social-media* (FT) and *normal-traffic with normal-social-media* (FF). The totals of these four classifications are then put into a four-quadrant contingency table. The odds ratio (OR) and risk ratio (RR) [110] are calculated according to this table to determine the correlation between any abnormalities of traffic volumes with abnormalities of nearby social-media distributions.

**Figure 7.7:** Geographic bounding box of the case study and the locations of the SCATS sensors

As discussed in Section 7.2, this case study is based on choosing a busy area (i.e. an area with a considerable amount of data). The bounding box in Figure
7.7 illustrates the specific area that was chosen. This area primarily covers the Melbourne CBD and its surrounding suburbs. The small red points in Figure 7.7 illustrate the positions of the SCATS sensors. SCATS sensors in Melbourne are usually installed at traffic signals at major street intersections. There are 493 sensor locations and 8,312 actual SCATS sensor devices within this chosen area.

Figure 7.8 illustrates the data processing pipeline of the traffic data used in the case study. In the preparation phase, SCATS traffic volume data generated between 2017-01-01 to 2017-12-31 within the chosen area of Figure 7.7 were stored in the Accumulo/Geomesa cluster. A traffic baseline data-set was generated using Spark and stored into this Accumulo/Geomesa cluster. Social-media data generated from 2017-06-01 to 2017-12-31 within the chosen area were also stored in Accumulo/Geomesa in another table. A social-media distribution baseline was then generated according to the location of SCATS sensors by processing.
7.3 Correlation between Social Media and Urban Traffic Data

the SCATS data and social-media data together with Apache Spark. 388,300 SCATS records were input into the analyzer. These represent all SCATS data generated from 2017-12-01 to 2017-12-07 inside the chosen area. The analyzer works in two steps against each input SCATS data/record. In the first step, it compares the traffic volumes \( \text{vol} \) given as input with its corresponding baseline and it tags abnormalities if \( \text{vol} \notin [\text{avg} - 2\sigma, \text{avg} + 2\sigma] \). The \( \text{avg} \) and \( \sigma \) represent the average volume and the standard deviation recorded in the baseline entities.

In the second step, the analyzer gets the number of nearby social-media data by querying the social-media data-set on SMASH and compares the number \( n \) to its corresponding social-media baseline. The abnormalities of social-media distribution are tagged if \( n \notin [\text{avg} - 2\sigma, \text{avg} + 2\sigma] \).

After identifying abnormalities in traffic volumes and nearby social-media distributions, each input is classified into four classifications (i.e. \( TT, TF, FT, FF \)) as described on page 161. The analyzer takes the size of those four classifications and creates a four-quadrant contingency table. Finally, the odds ratio and risk ratio is calculated according to the measure of association between traffic volume abnormalities and social media abnormalities. The equation for the Odds Ratio (OR) calculation is given in Eq 7.1 and the equation for the Risk Ratio (RR) calculation is given in Eq 7.2. We refer to social-media group as \( \text{socmed} \) group in these equations.

\[
\text{OR} = \frac{\text{Odds of traffic abnormality in abnormal socmed group}}{\text{Odds of traffic abnormality in normal socmed group}} \\
= \frac{N_{TT}/N_{TF}}{N_{TF}/N_{FF}} \quad (7.1)
\]

\[
\text{RR} = \frac{\text{Risk of traffic abnormality in abnormal socmed group}}{\text{Risk of traffic abnormality in normal socmed group}} \\
= \frac{N_{TT}/(N_{TT} + N_{FT})}{N_{TF}/(N_{TF} + N_{FF})} \quad (7.2)
\]

In statistics, the odds ratio and risk ratio [45, 125] are predominant ways to quantify how strongly the presence or absence of property \( A \) is associated with
Table 7.1: Results of SCATS data for the week 2017/12/01 to 2017/12/07 in the Melbourne CBD

<table>
<thead>
<tr>
<th>Social-Media</th>
<th>Traffic</th>
<th>Abnormal</th>
<th>Normal</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abnormal</td>
<td>TT</td>
<td>7,098</td>
<td>24,092</td>
<td>31,190</td>
</tr>
<tr>
<td>Normal</td>
<td>TF</td>
<td>69,302</td>
<td>287,808</td>
<td>357,110</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>76,400</td>
<td>311,900</td>
<td>388,300</td>
</tr>
</tbody>
</table>

Odds ratio = 1.224; Risk ratio = 1.172

the presence or absence of property $B$ in a given population. They are widely used in survey research and epidemiology. If the value of the OR is above 1, then it can be inferred that the presence of property $A$ (traffic volume abnormality) can be considered as ‘associated’ with the presence of property $B$ (abnormality in the surrounding social media). The higher the value of OR, the stronger the association. According to Eq 7.2, if the value of RR is above 1, then a social media abnormality has an associated higher risk of having a traffic abnormality than a normal social media distribution.

Table 7.1 shows the result of the case study in exploring the traffic relationship analysis between SCATS and social media. The corresponding odds ratio is 1.224 which is greater than 1. This means that the presence of traffic volume abnormalities has a degree of correlation to the presence of nearby social-media distribution abnormalities although the correlation is somewhat weak. The risk ratio is 1.172 which implies that there is a higher risk in having traffic volume abnormalities when the distribution of nearby social-media is abnormal. This suggests that the abnormalities of social media distribution may have positive effects in predicting traffic abnormalities.

It is noted that there are several fixed parameters applied in this case study. For
example, the factor 2 defines the range \( \text{avg} \pm 2\sigma \) used for identifying abnormalities. A distance \( 1 \text{km} \) and time window of 1 hour are used for getting the surrounding social media data. The impacts of these values to the OR/RR value are slight according to these experiments. That is, the odds ratio and risk ratio are generally above 1 in all of our case studies with the same data sets using this method.

### 7.3.3 Social Media Data-driven Abnormality Analysis

The previous section presented an association analysis based on the detection of urban traffic (volume) abnormalities. We saw a positive connection between traffic volume and social-media distribution abnormalities. In this section, we consider another association analysis driven by social media data detection to answer the last question raised at the beginning of Section 7.3: For detected social media abnormalities, can we find corresponding (nearby) urban traffic abnormalities?

There are two major differences compared to the analyzer in Section 7.3.2. Firstly, it uses a different approach for identifying abnormal distribution patterns of social media data. Here we use the spatio-temporal clusters of social media data as the identifiers for abnormalities in social media data distributions. Secondly, this analyzer is driven by the social-media data used as input. The hope/expectation of this case study is that it is possible to use this data as a proxy for SCATS data, which is one of our key hypotheses.

The same spatial bounding box as introduced in Section 7.3.2 is used as the target area of the case study. This method starts by detecting social media data clusters. The raw social media data are retrieved from the database on SMASH and subsequently pushed into a real-time stream processor. The real-time stream processor comprises a Kafka pipeline that normalizes multiple sources of social media data and a Spark-Streaming processor for clustering the social media data in time and space. Our real-time density-based spatial-temporal clustering
algorithm (RT-DBSCAN) is applied to this data. Two additional attributes are attached to the social media data after being clustered which classify the data if it is in a cluster and returns the unique ID of the associated cluster. The clustered/updated social-media data are then stored in the Geomesa/Accumulo cluster.

In this case study, the parameter used for calculating $\epsilon$ (the distance threshold for neighbours to be in a cluster) in DBSCAN is: $D_g = 200$ meters, $\Delta t = 20$ minutes, $s = 10$ meters/minutes. The exact formula for calculating the spatio-temporal distance (i.e. $\epsilon$) in RT-DBSCAN is given in Equation 6.1 on page 126. The minimum number of points that must exist to form a cluster is given as 3.

Figure 7.9: A detected spatio-temporal social media cluster and its surrounding traffic flow volume status

For each detected social-media cluster, the social media data-driven analyzer checks its surrounding SCATS records for abnormal traffic flow volumes. As with the previous analyzer, a traffic volume abnormality is detected based on its baseline data, i.e. it needs to be within the range of $\text{Avg} \pm 2\sigma$. A spatio-temporal bounding box is calculated for retrieving the surrounding SCATS data for each social-media cluster. The temporal range of this bounding box starts from the earliest timestamp to the last timestamp among the social-media cluster members.
The spatial area of this bounding box is a circular area with a calculated radius. The center of this circle is calculated by Equation 7.3, where \( n \) is the number of social-media cluster members and \( \lambda \) and \( \varphi \) are the longitude and latitude values of each cluster member. This weighted center is considered as the geographic center of the detected social-media cluster. The radius of this circle is calculated using Equation 7.4, where \( \lambda_{\text{min}} \) and \( \varphi_{\text{min}} \) are the minimum values of the longitude and latitude within the cluster members and \( \lambda_{\text{max}} \) and \( \varphi_{\text{max}} \) are the maximum values. The \( \text{dis} \) function is used for calculating the distance (in meters) between two GPS tags using the Haversine formula given by Equation 4.1 on page 74. In this case study, 300 meters was selected as the minimum value of this radius. Therefore, the surrounding traffic volumes to a social-media cluster are those surrounding SCATS records to the cluster center within the time period covered by the cluster.

\[
\text{Center} = \left( \frac{\sum_{i=1}^{n} \lambda_i}{n}, \frac{\sum_{i=1}^{n} \varphi_i}{n} \right) \quad (7.3)
\]

\[
R = \frac{\text{dis}((\lambda_{\text{min}}, \varphi_{\text{min}}), (\lambda_{\text{max}}, \varphi_{\text{max}}))}{2} + 300 \quad (7.4)
\]

Figure 7.9 illustrates an example of a detected social media cluster and its surrounding SCATS traffic volume records. This (detected) cluster had 24 social media data (blue round points in Figure) as its members, and those data were created at/near the Melbourne Cricket Ground during 2017-07-15T12:53:58 to 2017-07-15T15:46:43. After checking the surrounding SCATS traffic volume data using the spatio-temporal information of the cluster, 7 abnormal traffic volume records were identified across 4 surrounding SCATS sensor spots (red square boxes). The other 3 surrounding SCATS sensor spots (yellow square points) have no traffic volume abnormalities detected. One can assume that there was likely to be a sport match at the stadium during that time and abnormal traffic volumes occurred because of the attendees at that event. In general, it is non-trivial to
understand why a social media cluster occurs and what it may have impact on, e.g. what a crowd are doing just using the location and timestamp of the tweet. Social-media users from different backgrounds often talk about different things and can form a lot of spatio-temporal clusters. Irrespective of why, i.e. independent of the tweet text, the approach can be used to identify abnormal social media clusters from the surrounding traffic abnormalities.

In this case study, social media data (i.e. tweets and Instagram data) created between 2017-06-01 to 2017-12-31 were clustered through the RT-DBSCAN processors using the given parameters introduced as described previously. Each social media item was updated by adding a `clusterId` as a new attribute, where a `clusterId` is a unique id for each cluster. Empty values were set to the `clusterId` if the data did not belong to any cluster. After storing the updated social media data in the GeoMesa/Accumulo cluster, the social-media driven analyzer retrieves this dataset and generates the surrounding bounding box for each detected cluster through running MapReduce on Spark. SCATS records within those surrounding bounding boxes are retrieved from the GeoMesa/Accumulo cluster and compared to their baseline leveraging parallel processing capabilities of the Cloud and the SMASH software stack. A final report is subsequently generated which lists the discoveries for each social-media cluster.

In total, 2,261 social media clusters were detected among 132,927 social media data posts and 1,896 (83.9%) clusters were found to have abnormal traffic volumes in their surrounding spatio-temporal areas. This result shows a strong correlation between the presence of social media clusters with the presence of traffic volume abnormalities. It is noted however that there are several preset parameters which have a direct impact on these results. For example, the DBSCAN parameters defines the density threshold for cluster detection and the equations used for calculating the surrounding spatio-temporal area for each cluster impact on the results. Besides the parameter settings for describing social media clusters, the distribution of SCATS sensors also has impacts on these results.
That is, some social media clusters may have more SCATS sensors/records in their surrounding area than others. In the worst case, these may have no SCATS sensors in the surrounding area of a social media cluster, hence no correlations can/will be found.

![Figure 7.10: Surrounding Traffic Abnormality Detection with Different Sizes of Social Media Clusters](image)

Figure 7.10 illustrates the detected traffic volume status in the surrounding areas of social media clusters based on the size of cluster (i.e. the number of cluster members). From the 2,261 detected clusters, we identify that most of them have 5 or 6 members (tweets/posts). An interesting discovery from this figure is that clusters with more members have a higher probability to have abnormal traffic volumes in their surrounding area. One reason is that larger clusters
potentially have a larger surrounding spatio-temporal space which can cover more SCATS sensors/records. However, a larger size of cluster does not always have a larger surrounding area than a smaller cluster. Another reason may be that a cluster with a higher density has a stronger connection to traffic abnormalities. According to such evidence, the density of social media clusters can be used as a factor in calculating the confidence of traffic abnormality prediction when a social media cluster is identified in real-time.

The case studies in Section 7.3.2 and Section 7.3.3 explored the correlation between official urban traffic data and social media data in two directions. The results indicate that there are indeed spatio-temporal correlations between traffic volume abnormalities and social-media distribution abnormalities. These results provide a basis for using social-media as a cheaper proxy for urban traffic congestion detection, with various caveats such as the density of social media and traffic information. This is a reasonable assumption however since traffic issues typically occur in densely populated urban areas.

7.4 Conclusions

In this chapter, we explored whether it is possible to use social media data for traffic analysis as a proxy for the more expensive and difficult to obtain traffic data from official systems such as SCATS. The focus here has been through focusing on traffic abnormalities as recorded by SCATS platforms and by social media platforms such as Twitter and Instagram. The results of the case studies show that where sufficient data exists for SCATS and for social media data, they can indeed be used as a predictor for actual traffic, however some limitations exist:

- Many of the detected clusters are identified with a small number of social media data. The number of cluster members in general depends on the density of social media data and the selected experimental parameters (e.g. the spatio-temporal buffers and thresholds) used by the clustering
algorithm. The variations of these factors can have an impact on the robustness of the results and would thus need further exploration.

- SCATS sensors and social media users are not evenly distributed around Victoria. As a result, the accuracy of the results and indeed the applicability of the method for using social media data depends greatly on the number of social media clusters that are found. Within Melbourne, these tend to be near the CBD.

Further evaluations on a wider range of data and case studies are therefore needed to build a more generic prediction model in the future.

This chapter is largely based on the following publications:


In this chapter, we summarize the major contributions made in this thesis and propose some potential research directions that can be explored in the future.

8.1 Summary of this Thesis

In this thesis, we identified that a specialized platform is needed for conducting many emerging urban traffic analytics, especially those based on use of Big Data. In particular, challenges related to the volume, velocity, variety, veracity, volatility and value (6Vs) of traffic data needs targeted infrastructure support. We presented the SMASH architecture as one approach to address these issues and hence to meet one of the core hypotheses identified in this thesis. In addition, we identified
that official urban traffic data are usually expensive to collect and hard to access and use, especially in real-time. This gave rise to the second research hypothesis — “Can we find certain ‘cheaper’ real-time accessible data sources that can be used as proxies for real-time urban traffic”? We identified that social media data could be used as a proxy for traffic data and this was explored in detail in this thesis. Different to many existing social media urban research methods, we focused on the spatio-temporal information that often exists within social media data rather than the content of social media, e.g. the given text sent in a tweet. From this, we identified the skew of spatio-temporal clusters of social media data and how this information could be used as the key for identifying (surrounding) urban traffic problems. In exploring this, novel methods and cases-studies were proposed and examined with regard to the extent that real-time traffic analytics can be achieved using social media data. The details and discussions of our major contributions in this thesis can be summarized as follows.

- In Chapter 3, we introduced the data structures and data collection methods involved in this thesis. In particular, we undertook a review related to the positioning accuracy of the geo-location information generated by mobile devices. The accuracy of user geo-location recorded in social media data is critical to our hypothesis since it directly affects the results of our traffic analysis and methodology more generally. The geo-location accuracy of devices (within 8metres) is considered acceptable under the experimental parameter settings used in the analysis methods of this thesis. We also outlined how related data sets were acquired including the road network data and SCATS data.

- In Chapter 4, we proposed a novel method for harvesting social media data focusing on social media posts made solely along the street network. This was used for harvesting targeted social media data based on their spatial correlation to the road network through use of the official road
network data for Australia (from PSMA). It is noted that other solutions are also possible, e.g. collecting all data in a given region and subsequently identifying those that occur on the street/road network through batch-based data processing. However, this approach involves the collection and processing of considerable amounts of unnecessary data, e.g. tweets and posts that are not sent from or near to the street network. The harvesting method here is targeted specifically to the needs of traffic data analytics. We identify that the text of social media cannot always be trusted, but the location and temporal aspects of social media posts can be trusted to a much greater extent. A review of existing social media applications for urban traffic analytics was made at the end of this chapter. Both the advantages and limitations of using such data are discussed and considered within the proposed methodology of this thesis.

- In Chapter 5, we proposed and deployed a Cloud-based architecture — SMASH as a generic platform for conducting (real-time) urban traffic analytics. This architecture provides a solid technology/software stack for tackling many of the Big Data challenges identified in relation to urban traffic data and the geospatial data domain in general. The computation layer of the SMASH architecture is flexible and offers both batch data processing for aggregating large datasets, as well as real-time stream-based data processing, e.g. for analyzing large volumes of incoming data on-the-fly. It also provides the capability and scalability to adopt many complex urban traffic analytical methods for dealing with the 6VS of Big Data. The service/interface layer of SMASH offers a one-stop solution for visualizing urban traffic data and its analytic results with or without mapping services, e.g. Google Maps.

SMASH also supports a real-time parallel algorithm for clustering spatio-temporal stream-based data. The SMASH architecture simplifies the imple-
mentation work required when dealing with traffic data including support for spatio-temporal indexing, stream buffering, resource management and for the overall performance when used on Cloud infrastructures. In this thesis, all of the map based visualization/figures were generated using the SMASH platform and displayed in real-time using standard web browsers. These kinds of data visualizations/layers can also be combined with many other map data services for richer data analytic case-studies.

The SMASH platform represents a key contribution of the thesis as a whole. It is noted that since the thesis was submitted, other researchers have used SMASH for a range of real-time big data scenarios. This includes urban health related researchers exploring pollution and its impact on the health of the population.

- In Chapter 6, we proposed a novel extension of the widely-used DBSCAN algorithm — RT-DBSCAN for real-time clustering of Big Data streams with specific focus on the spatio-temporal density of real-time and evolving data. RT-DBSCAN was implemented within the SMASH platform and subsequently used for clustering social media data by both time and space to provide a customized spatio-temporal distance measure. We also identified that RT-DBSCAN could equally well be adapted to many other data fields as long as a customized measurement of distance/density was provided. For example, a similarity distance can be used to identify certain duplicate data patterns in many real-time data domains such as emergency response and network attack detection. In a case-study benchmark provided in this Chapter, it was shown that RT-DBSCAN on SMASH was able to tackle data streams comprising over 10,000 social media posts per second. During the benchmark, we tested the scalability of this method and identified several parameters of RT-DBSCAN that had a significant impact on its performance when running on the Cloud.
8.2 Future Work

In Chapter 7, we explored the extent that social media data could be used as a proxy for official traffic data through key case studies. Specifically, we explored the correlation between abnormalities in social media data and official urban traffic volume data (and vice versa). Based on the results, we demonstrated that spatio-temporal clusters of social-media data could indeed be used as a form of cheaper/free\(^1\) proxy for more official traffic data. We showed how it was possible to identify social media clusters in real-time and use such information for detecting and/or forecasting traffic abnormalities such as congestion.

8.2 Future Work

In this section, we identify several potential research directions that could build on the work presented in this thesis.

8.2.1 Potential Extensions to the Research

This subsection covers several future works that could be attempted to improve or extend the work presented in this thesis.

- Event analytics: one extension of this thesis’ work is to further analyze the detected social media clusters and how they might be connected to other events. This might for example be cultural/sporting events, fire/traffic accidents or more seasonal events such as the mobility of people to coastal regions during the summer period. By analyzing the content of social media (e.g. using hashtags and keywords) within each detected cluster,

\(^1\)Although we note that free here implies there is no direct cost to access the data from Twitter/Instagram. A cost may well exist for the infrastructure used for the processing and storage of the data, e.g. Cloud computing costs. It is also noted that larger scale data access and use from Twitter may also incur a cost, e.g. using the so-called Firehose to access significantly more data.
Conclusions and Future Work

it would be possible to classify the social media clusters into different themes/topics with associated parameters for estimating their impact on the urban environment, e.g. major traffic accident. In cities such as Melbourne, sporting events can attract a huge volume of people both at the beginning and at end of the event. By analyzing the growth of the associated social media clusters, we may able to capture more detailed information related to crowd mobility. For example, a demonstration/protest-related cluster with directional growth along the street network may be used to infer the movement of protesters and their potential area of impact. Spatio-temporal clusters of social media data may also be used for real-time scenarios, e.g. emergency response. By aggregating social media clusters and comparing them with historical data, it may be possible to find regular patterns of clusters which can be related to events. This information can help city planners to allocate appropriate infrastructure resources to afford congestion issues. For example, road or rail enhancements and their impact on the population can be tackled more systematically. As one example of this Wanninayake focused on commuter travel patterns based on social media clusters [187]. This work showed how people commuted to the Melbourne CBD from the inner/outer suburbs and identified the time and commuting transport they used, e.g. clusters of tweets at railway stations or at bus stops. Through identification of tweeters in clusters at remote suburbs and the same tweeters tweeting in the CBD, the minimal commuter time could be established and hence the travel patterns established.

- **Increasing data resources**: another extension to the work described in this thesis is to increase the volume and sources used for real-time spatio-temporal data and the related cluster analytics. In this thesis, data from the social media platforms Twitter and Instagram were used as the non-official traffic data source. One obvious extension is to increase the volume of social media data and use data from the many other social media sources that
provide geospatially related content, e.g. Flickr, Foursquare. However, the major problem in using social media data is that a lot of social media data does not have geospatial information. To increase the volume of geo-tagged data, other spatio-temporal data (e.g. from portable devices and sensor networks) might be used for clustering analytics. For example, the access records from location-based services like Uber, weather apps and many other nearby search services could be applied. To combine the usage of multiple real-time spatio-temporal data sources could be a new challenge to tackle, e.g. ensuring latency and associated data jitter issues are tackled, as well as the issues of harmonising the data itself, e.g. if it is not in standard JSON format as used here.

- **Adopting other traffic analytic methods within SMASH:** many existing urban traffic analytic methods could also be implemented within the SMASH platform. A variety of traffic data can be stored, indexed and visualized on SMASH including data related to traffic trajectories to data from satellite images. Targeted and potentially complex algorithms for real-time analytics could then be executed as part of a scalable computation environment on the Cloud without the need to worry about system and hardware requirements. As one example of this, the SCATS data for Victoria was explored using the SMASH platform. Figure 8.1 and Figure 8.2 illustrate two visualizations of the associated SCATS analytics that was conducted using the SMASH platform. Figure 8.1 presents the change of traffic flow on a single day in Adelaide CBD represented as a heatmap. Figure 8.2 illustrates the entire traffic volume in Victoria from 2008 and 2014 as a heatmap. By comparing the change of such heatmaps, population growth can be seen with the formation of new transport corridors (highlighted in the green ellipses). Such visualisation can shed light and help to understand the development of the city, the urban sprawl and importantly the impact on human mobility. This has many implications on the health and wellbeing of populations at large,
as well as the costs and planning of new infrastructure, e.g. new roads or train stations.

Figure 8.1: Traffic flow heatmaps of the Adelaide CBD on 2012-05-17 at 0:00, 6:00, 12:00 and 18:00 created through the SMASH platform
Figure 8.2: Heatmaps of total aggregated traffic flow in Victoria in 2008 and 2014 created through the SMASH platform
• **Using SMASH for other urban research:** the SMASH architecture can also be used as the platform for many other urban research challenges, especially those dealing with geographic or spatio-temporal data. For example, air pollution problems can be analyzed on SMASH by comparing the pollution data from air pollution stations to their surrounding environment, e.g. traffic congestion hotspots and factories.

• **Improving the social media clustering model for traffic forecasting:** there are several directions to improve the current analytic model used for traffic issue forecasting and analytics.

  – Firstly, the space distance used when calculating the spatio-temporal distance is based on the Euclidean distance between two data points (i.e. longitude and latitude and the associated timestamps). This is often naive. For example, if we consider two geospatial points as representing two humans in the city, then their actual mobility distance is related to their surrounding infrastructure and the associated environment, e.g. the specific road networks including one way streets and streets with no-through roads. One extension to the work is to take such information into the spatio-temporal distance measurement used as part of the cluster detection.

  – Secondly, ideally it should be possible to quantify the confidence in forecasting the surrounding traffic issues through detected social media clusters. Information like the density of the cluster, the spatio-temporal coverage of the cluster, the events that might be related to the cluster, the location of the cluster and indeed the surrounding transportation/infrastructure resources should ideally be taken into consideration to generate a more accurate score for traffic issue forecasting.

  – Thirdly, the contents of social media posts can be used to provide richer
information. At present, the location and time of the social media data are used for the analysis, but it would be relatively straightforward to also identify the content of the posts, e.g. tweets related to a traffic accident or a particular event. This was not considered here as part of the work in this thesis, but was explored in [161].

- This thesis has explored the use of social media for outdoor traffic issues, but the work and the SMASH platform could in principle be used to address indoor issues as well. Identifying the location of individuals and the paths they take in hospitals, for example, is one possible extension - noting that this would likely be based on other forms of geospatial information and not social media, e.g. their GPS-enabled phone. Combining other forms of data such as data collected from wifi access points to understand movement patterns, e.g. across a University is another extension to the work [54].

- It could also be useful to adopt arbitrary criteria for identifying clusters, e.g. by combining spatio-temporal distance with other potential knowledge such as whether the user is indoor/outdoor, what is their mood, and aspects of the nearby environment etc.

• **Applying RT-DBSCAN to other research domains:** as identified, there are many potential scenarios and application domains where RT-DBSCAN might be applied. In addition to Spark-Streaming, RT-DBSCAN can also be implemented on other data processing architectures as long as a form of master-slave mode communication is supported.

• **Other mainstream Cloud deployments:** Deploying SMASH to other mainstream Cloud infrastructures such as AWS, Azure or Google Cloud would also be one potential extension to the work, since right now it is only accessible to Australian-based researchers due to the terms and conditions of the NeCTAR Research Cloud. This could include targeted use of data centres
based on localised data collection and processing needs. This would result in research topics related to dynamic selection of Cloud availability zones based on the current load, cost and actual source of data requirements at a given point in time [4].

- **Combining real-time social media clustering with existing community-based traffic applications such as Waze:** As discussed in Section 4.6, there are several existing applications that can provide real-time traffic information based on user reported data. It would be helpful to link the social media clusters to the traffic issues reported by such applications to improve the overall confidence. Unfortunately in Australia, there is no mainstream (adopted) platform like Waze.

### 8.2.2 Other Areas

There are many other unexplored areas related to the work presented in this thesis. Some of the more important of these are discussed below:

- **Geo-privacy issues and data protection:** privacy issues are increasingly important when processing user generated spatio-temporal data (such as social media data). The location information of individuals can be used to identify their homes and workplaces. The associated trajectory data in space and time can be used to identify their commuting patterns and lifestyle more generally. Clustering spatio-temporal data across populations can be used to infer the potential relationships between individuals. Many of these issues were explored in [181]. Considerably more work is required to ensure that such data related scenarios are not possible and hence data is not misused.

- **Social network analysis:** social network analysis [151] is another aspect of research on social media data, which focuses on the connection between
people and their associated friendship networks. Common methods of gathering the connection network of social media users are based on the public friendship information available in the user profile, e.g. the follower-/followee information available on Twitter. Such friendship connections are obviously based on social media networks and this may not always reflect the actual friendships. The spatio-temporal relationship of two unconnected accounts could potentially be used to analyse such kinds of relationship. For example, if a group of users is commonly involved in the same social-media clusters, they are more likely to know each other or have certain connections. The applications of such richer relationship could be used to augment the traffic related scenarios described in this thesis.

- **Artificial Intelligence (AI) technology with real-time data**: As introduced in Section 1.5.2, the SCATS system is an adaptive urban traffic control system which can adjust traffic signals based on the current traffic volumes at given road intersections. In the future, such a system could be replaced by a much more powerful decision making approach, e.g. based on AI technology running on the Cloud and using a wide range of real-time Big Data streams. An overview of this technology was explored in [156]. The spatio-temporal relationship of data is the key to its potential exploitation. Other works have looked at identification and classification of trucks and trailers on the road network [38]. This can be used for example to ensure that certain trucks cannot travel along certain roads at given time periods.

- **Autonomous vehicles**: autonomous vehicles are another hot research topic related to AI technology [59]. This technology requires that vehicles are able to sense their surrounding environment, typically using radar sensors [20, 189]. As well as being used for automatic (self)-driving, the information collected by these sensors could potentially be used for analyzing the traffic status. This would be a future research area with numerous major research
challenges, e.g. the volume and velocity of data as well as associated privacy considerations.

- **Computational infrastructure requirements for Internet of Things (IoT) and Mobility-as-a-Service (MaaS):** The computational infrastructure requirements for IoT and MaaS are similar in some ways to urban traffic research as they are based on the ability to host, process and visualize large amounts of spatio-temporal data. There are billions of IoT devices and the diversity and integration of this data and its impact on traffic would offer many new research challenges. This could augment the work described in this thesis. One challenge is the access and use of such data and the many frameworks and approaches that are currently available in the IoT/MaaS domains.
Glossary

6Vs  Volume; Velocity; Variety; Veracity; Volatility; Value

A-GPS  Assisted GPS

API  Application Programming Interface

AURIN  The Australian Urban Research Infrastructure Network

BDaaS  Big Data-as-a-Service

CaaS  Container-as-a-Service

FCD  Floating Car Data

GPS  Global Positioning System

HDFS  The Hadoop Distributed File System

HPC  High Performance Computing

IaaS  Infrastructure-as-a-Service

IoTs  Internet of Things

187
**MPI** The Message Passing Interface is a programming paradigm/architecture for communication and sharing data between processes, allowing to take extremely large data sets and process them in parallel typically across HPC resources.

**NLP** Natural Language Processing

**PaaS** Platform-as-a-Service

**PSMA** The Public Sector Mapping Agency — www.psma.com.au

**SaaS** Software-as-a-Service

**SCATS** The Sydney Coordinated Adaptive Traffic System is a fully adaptive urban traffic control system that is used in 27 countries for capturing and (potentially) augmenting the decision-making related to traffic flows.
Bibliography


[28] Bert Brunekreef, Nicole AH Janssen, Jeroen de Hartog, Hendrik Harssema, Mirjam Knape, and Patricia van Vliet. Air pollution from truck traffic


[54] Muawya Habib Sarnoub Eldaw, Mark Levene, and George Roussos. Presence analytics: making sense of human social presence within a learning


[80] Jiawei Han, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011.


[164] Christian Spieth, Felix Streichert, Nora Speer, and Andreas Zell. Clustering-based approach to identify solutions for the inference of regulatory net-


[180] Chin-Liang Wang, Yih-Shyh Chiou, and Sheng-Cheng Yeh. A location algorithm based on radio propagation modeling for indoor wireless local area


Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Gong, Yikai

Title:
A big data infrastructure for real-time traffic analytics on the cloud

Date:
2019

Persistent Link:
http://hdl.handle.net/11343/230905

File Description:
Final thesis file

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.