A Big Data Architecture for Near Real-time Traffic Analytics

Yikai Gong  
School of Computing and Information Systems  
University of Melbourne  
Melbourne, Victoria 3010, Australia  
yikaig@student.unimelb.edu.au

Paul Rimba  
Data61  
Melbourne, Victoria 3010, Australia  
Paul.Rimba@data61.csiro.au

Richard O. Sinnott  
School of Computing and Information Systems  
University of Melbourne  
Melbourne, Victoria 3010, Australia  
rsinnott@unimelb.edu.au

Abstract— Big data is a popular research topic that has brought about a range of new IT challenges and opportunities. The transport domain is one area that has much to benefit from big data platforms. It requires capabilities for processing voluminous amounts of heterogeneous data that is often created in near real time and at high velocity from a multitude of distributed sensors. It can also require the application of performance-oriented spatial data processing of such data. In this paper, we present a platform (SMASH) that tackles many of the specific challenges raised by the transport domain. We present a range of case studies applying SMASH to transport and other data used to understand traffic phenomenon across the State of Victoria, Australia. The novelty of this work is that this Cloud-based platform is not designed for a specific type of data or for a specific form of data processing. Rather it supports a range of data flavours with a range of data processing possibilities. In particular we show how the platform can be used for analyzing social media data used for traffic jam identification through spatial and temporal clustering tweets on the road network and compare the results with official real-time traffic data based on the Sydney Coordinated Adaptive Traffic System (SCATS – www.scats.com.au) that has been rolled out across Victoria.

Keywords—Big Data; Cloud; Traffic Analysis

1. INTRODUCTION

Urban research is increasingly driven by data [1]. A rich variety of data are being continuously generated around us in our cities. Information capturing people’s daily movements and their behaviors is now available in many data resources. These data cover areas such as transport, the environment and social-media interactions. Some of these data are collected passively at every moment by different sensor-networks deployed across cities and urban environments more generally. Due to the popularization of smart phones with GPS capabilities, many other forms of data are actively posted by people through their mobile devices. Such huge volumes of heterogeneous data bring many data challenges to researchers who are trying to analyze these data collections for different research purposes [2]. In this context, big data platforms used to derive knowledge from such diverse data collections are increasingly demanded [3]. Transport and traffic analysis is one typical area that is facing such problems. Tackling these problems requires the ability to store, index, analyze and derive knowledge from ever-growing data sets in a durable and efficient manner. Ideally a generic yet scalable platform is needed. Such a platform needs to support scaling for increasing data volumes; dealing with high velocity ingestion of data; supporting flexible and optimized data analysis and subsequent visualization of results. In this paper, we present a Cloud-based platform offering such capabilities.

Cloud computing is a dominant technology for big data [4]. It provides a scalable base infrastructure. There are many Cloud initiatives that have established a range of cloud computing services and associated resources, e.g. Amazon Elastic Compute Cloud (EC2), Google Cloud Platform and Microsoft Azure. Many of these platforms offer Infrastructure-as-a-Service (IaaS) capabilities that provide access to virtual machines (servers) and associated storage. However, this core infrastructure is insufficient for tackling the many diverse challenges and opportunities that arise in the transport domain. Rather, targeted software solutions should run on top of core IaaS solutions — so called Software-as-a-Service (SaaS) with capabilities specifically demanded by application domain, e.g. support for traffic data analytics.

There are several existing commercial clouds for monitoring traffic status, e.g. Google Traffic [5] and WAZE [6]. However, a generic platform offering a variety of transport analytics is not yet widely available. In this paper, we present a Docker-based (www.docker.com) SaaS solution for transport analytics that utilizes geoServer, geoMesa, Apache Accumulo, Apache Spark and Apache Hadoop (SMASH). We describe the key capabilities of the platform that support the specific needs and demands of traffic data. In the case studies, we show how this solution provides capabilities for applying statistical analytics and machine-learning methods on highly disaggregated traffic data including combinations of otherwise independent data sets. Specifically we consider the correlation between road-based social-media data from Twitter, i.e. tweets made by individuals on the road network, and official SCATS traffic data from the City of Melbourne provided by the Department of Transport in Victoria [7].

The SMASH platform itself is deployed on the federally funded Australia-wide National eResearch Collaboration Tools and Resources (NeCTAR – www.nectar.org.au) Research Cloud. The NeCTAR project has been 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 Docker Engine API to deploy and manage the SaaS-oriented platform with targeted analytics required by the transport domain, although the solution has been developed to be easily deployable on other Cloud resources.

The remainder of the paper is organized as follows. In Section 2, we review related work on big data processing and transport analysis. Section 3 covers the description of the SMASH architecture and the related technologies. Section 4 presents the implementation of the platform and use of Docker to facilitate deployment and scaling across the NeCTAR Research Cloud. Section 5 shows the case studies in analyzing tweets and SCATS data on SMASH. Finally, Section 6 draws conclusions on the work as a whole and identifies future enhancements.

II. RELATED WORK

Urban transport and traffic more generally is an area that many researchers have explored from a variety of perspectives [8] [9] [10]. One common demand is for traffic forecasting [8] [11] [12]. Precise prediction of traffic congestion can help planners make informed decisions on traffic control and thus lead to savings of travel time, fuel and reduce the overall environmental impact. There are many different approaches for traffic forecasting. A typical class of models 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 [13] and Kalman filtering [14]. 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 of actual traffic flows. Another group is knowledge-based intelligent models, which include support for vector-based regression models [15], artificial neural network models [11] and their various extensions [16] [8] [12]. These methods usually involve applying some form of machine learning algorithms 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 towards building forecasting applications, speed of processing is often a key requirement. One challenge in applying these models is the large-scale historical training data that can often be required, since a single machine cannot store and process voluminous amounts of data efficiently. For real-time forecasting, a high throughput solution is needed to tackle the speed of the incoming data whilst continuously updating the model. Performance and scalability are 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, however, there are a variety of types of data. For example, [10] 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 is a challenge in adopting this method for traffic flow forecasting due to the demands for the processing large-scale image data at city-scales in real-time. This problem is exacerbated since high-resolution point clouds are usually huge in size. Applying analysis algorithms on this size of data can thus take considerable time and require significant computation power. In addition, the 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 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 [17]. With the GPS information commonly associated with social-media, [18] manages to cluster tweets data in time and space specifically on the road network. This can then be used to detect traffic accident blackspots [19] or explore the relationship between those clusters to traffic issues, e.g. the impact on urban traffic caused by public events.

In Australia, many transport agencies collect traffic data through the SCATS system that is used to capture individual vehicle location information [7]. SCATS deploys 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 of this data in real time has data bottlenecks.

Processing and monitoring urban traffic data on the Cloud is not a new idea. For example, [20] use Cloud computing to support intensive Floating Car Data (FCD) models [21] 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 [22] are also used for traffic parameter computation, e.g. vehicle speed. The results of such raw data processing can be exported to Advanced Traveller Information System [23] and/or Advanced Traffic Management Systems [9] for subsequent traffic management and control. Their infrastructure 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 analyse such data. Nevertheless their framework provides a workable 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. However 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.

### III. SMASH ARCHITECTURE

The SMASH architecture aims to solve the problems in applying a variety of traffic analysis approaches on different “big” datasets in a scalable and real-time manner. SMASH is a distributed software stack realized as a collection of software components that work together in a distributed environment (especially Clouds) to form a complete framework for big data applications. The SMASH architecture is designed to tackle the issues associated with transport analytics identified in section II. The overall software-layered architecture of SMASH is shown in Figure I [24].

![High-level SMASH Architecture Diagram](image)

**Fig. 1.** High-level SMASH Software Architecture

Early research on big data processing techniques can be traced back to early 2000s and efforts of Google through MapReduce programming model, the BigTable DBMS and the Google File System [25]. There is no definitive description of big data, but generally it is considered for datasets of such size (volume) that they cannot be stored on a single machine (server) nor can the analysis be undertaken on a single machine (server). These issues place numerous demands on the underlying data infrastructure including the need for fetching, searching, storing and visualizing large amounts of data. In designing SMASH, it was recognized that a distributed file system (DFS) was essential. This DFS should be fault-tolerant, easy to scale and efficient in I/O for handling large data sizes demanding rapid import and subsequent analysis. The Hadoop Distributed File System (HDFS) [26] as an open source implementation of Google File System, meets this requirement and is the base layer of SMASH.

Secondly, we need a distributed database, which supports spatio-temporal indexing for storing and querying diverse traffic-related data. The software solution we chose to satisfy this requirement included GeoMesa with Apache Accumulo. GeoMesa is an open source, distributed, spatial-temporal database that can be deployed on many distributed Cloud data storage systems, including Accumulo, HBase, Cassandra, and Kafka. Accumulo is a BigTable data storage system similar to Apache HBase. It is noted that GeoMesa was originally developed against Accumulo (unlike HBase). GeoMesa aims to provide functions akin to PostGIS/PostgreSQL that are compatible to the Open Geospatial Consortium (OGC) standards. Moreover, it has native support for many big data technologies including Apache Spark, Apache Kafka, and stream processing solutions such as STORM and HERON.

The third part of SMASH software stack is the computation engine. It is known that the native Hadoop MapReduce engine is good for simple parallel tasks. However, it is both difficult and inefficient when used for more complex algorithms, e.g. spatial clustering, image processing and data modeling using machine learning approaches. Apache Spark is an alternative data technology inspired by Hadoop MapReduce but utilizing in-memory processing suitable for more complex calculations. SMASH contains native support for querying spatio-temporal data from GeoMesa, Accumulo, HDFS distributed data storage stack to the Resilient Distributed Datasets (RDDs) in Spark – a fundamental data structure of Spark that abstracts the parallelization of distributed data chunks for developers. Geo spatial queries can be directly executed on the interface of GeoMesa, which utilizes the underlying Spark SQL functions. Although Spark is suitable for processing big data, one of its limitations is real-time processing. Spark has a set of APIs (Spark Stream) for real-time applications that process data through iterators in batch mode. In comparison to other emerging big data streaming processing technologies such as Apache Flink [27] and Apache Apex [28], a minimum delay is always present with Spark when handling real-time incoming data streams. Despite this limitation, key reasons for selection of Spark as a computing engine in SMASH are its performance in parallel computing, abundant APIs and direct support for GeoMesa and HDFS.

A further demand of the SMASH architecture is for 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 geospatial data. 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/or Open Street Map. It should also be noted that it is possible to export data from GeoMesa in a variety of formats to potentially feed different traffic management systems, such as Advanced Traffic Management Systems [9].

One of the key requirements in designing the SMASH architecture is scalability. Each of those software components in SMASH is designed for scale. To minimize the efforts for scaling the SMASH cluster, we use Docker Engine (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 preferred settings into separate Docker images to provide different layers of services. For instance, we create a Docker image for 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 2 illustrates the structure of this Dockerized implementation of SMASH.

Existing products such as the Docker-based Kubernetes (www.kubernetes.io) and Docker Swarm (http://docs.docker.com/engine/swarm) 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. This is realized as a toolkit written in JavaScript with associated NodeJS run-time environment. This toolkit utilizes several open source libraries to realize its functionalities. For instance, pkgcloud (http://github.com/pkgcloud) abstracts away differences between multiple Cloud providers including managing the VMs and firewall rules on the Cloud. Dockeroad (http://github.com/apocas/dockeroade) is a remote Docker API used for managing Docker images/containers. Dockprom (http://github.com/stefanprodan/dockprom) is used to monitor the status and resource usage of each Docker container.

![Diagram of SMASH deployment](image)

This toolkit together with the Docker-based solution greatly simplifies the work in scaling the SMASH architecture including support for both manual and/or automatic scaling. Software instances, software status snapshot and data resources are all put into transferable Docker images/volumes to increase the overall flexibility, e.g. to extend the nodes of a SMASH cluster or to recover from failed nodes. Once a SMASH cluster is initiated/deployed, a distributed application can be deployed together with SMASH to monitor the real-time resource usage of the cluster and scale it via the SMASH toolkit based on the required scalability and fault tolerance demands when dealing with bursty traffic data.

IV. CASE STUDIES

In this section, we present several case studies using the SMASH platform for detecting and analyzing urban traffic flows. Specifically, statistical analysis and spatio-temporal algorithms are applied to traffic data (SCATS) and Twitter data – or more precisely for analysis of tweets made on or near the road network.

A. Applying DBSCAN Clustering on SMASH

The goal of this case study is to identify spatio-temporal clusters of tweets made on or close to the road-network. Our expectation of this work is to establish the relationship between those tweet cluster patterns with real traffic phenomena in Melbourne, e.g. identifying the rush hour and junctions/roads with heavy traffic. The clustering algorithm we use is based on the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [29]. DBSCAN is one of the most common and most cited clustering methods available. The benefits of using this algorithm for clustering tweets are that it is not necessary to provide the number of clusters. Rather the algorithm can find clusters and identify the size and spatio-temporal distance between clusters of tweets. To adapt this algorithm to the near real time transport domain on the Cloud, a parallel implementation is needed. [30] proposed a scalable MapReduce-based version of the DBSCAN algorithm (MR-DBSCAN) that utilizes Hadoop MapReduce. The main contribution of their work is a computation cost-based data partition method, which is able to balance the workload of each parallel DBSCAN sub-task to tackle heavily skewed data. Building on this, [31] presented an implementation realizing MR-DBSCAN using Apache Spark. Using SMASH in this case study a multi-dimensional adaptation of MR-DBSCAN is realized.

1) Multi-dimensional extension of MR-DBSCAN for spatio-temporal clustering

Algorithm 1 shows a classic version of the sequential DBSCAN algorithm. The input of this algorithm is a set of data points; a distance parameter \( \varepsilon \), which indicates how far away \( P_i \) to \( P_j \) need to be from each other to be considered as neighbors and a positive integer parameter \( \text{MinPts} \) used for the limits on the number of points in a cluster. The algorithm is flexible since it is possible to define arbitrary distances between points in a multi-dimensional space. When applying this algorithm to GPS point-based data, the definition of the distance \( \varepsilon \) is the two-dimensional Euclidean-distance between points on a map. When we apply this algorithm for clustering GPS point data with timestamps in a spatio-temporal manner, i.e. identifying clusters of tweets that are created physically close to each other in a particular time period, the definition of spatio-temporal distance \( \varepsilon \) becomes less clear. [18] define a spatio-temporal distance \( \varepsilon \) from point \( P_i \) to \( P_j \) in (1), where \( x \) and \( y \) represent the two-dimensional location of a point and \( t \) represents the timestamp value.

\[
P_i = (x_i, y_i, t_i) \quad P_j = (x_j, y_j, t_j) \quad \varepsilon = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (t_i - t_j)^2}
\]
A problem with this equation is that the value of \( x, y \) and \( t \) can be in different scales and have different units of measure. If \( x \) and \( y \) are measured in kilometers and \( t \) is measured in milliseconds, the time difference will have a much bigger impact on the spatio-temporal distance.

To address this, in this case study, we apply a new approach to map the value of \( x, y \) and \( t \) into the same scale. The mapping method we used is based on the use of a reference system. Conceptually, this reference system is like a moving object, e.g. a walking person where their average speed is recorded. We use the speed of this reference system to map timestamp values into distance values. The refined spatio-temporal distance is shown in (2) where \( s \) represents the speed of the mapping reference system:

\[
P_{g} = (x_{1}, y_{1}, t_{1}) \quad \quad P_{s} = (x_{2}, y_{2}, t_{2}) \\
\varepsilon_{x} = \sqrt{(x_{1} - x_{2})^{2} + (y_{1} - y_{2})^{2}} \\
\Delta t = |t_{1} - t_{2}| \\
\varepsilon = \varepsilon_{s} + (\Delta t \times s) \tag{2}
\]

Through this approach, we can adjust the impact between time and space/distances by changing the reference system. Figure 3 illustrates this spatio-temporal distance and how it looks like in a three-dimensional space.

The reference speed in Figure 3 is 6 kilometers per hour. There are two planes, which indicate different timestamp slots. Plane \( T1 \) is the layer for timestamp at 0 hour and Plane \( T2 \) is the layer for timestamp at 0.1 hour. The two points in the figure represents two spatio-temporal data, e.g. timestamped and geo-located tweets. Point A at the coordinate origin is data at zero-zero-zero, while Point B is the data generated 0.1 hours after Point A. The \( z \)-axis value of Point A and B is generated by multiplying the timestamps at Point A and B and the reference speed. The spatio-temporal distance is then calculated as the Euclidean-distance in this transferred 3D space. In Figure 3, the sphere indicates the \( \varepsilon \)-neighbor zone of Point A where \( \varepsilon \) equals 1 kilometer. Here Point B is outside this sphere hence Point B is not a neighbor of Point A according to the DBSCAN algorithm.

2) Applying MR-DBSCAN on SMASH for analyzing social-media data

MR-DBSCAN adapts a sequential DBSCAN algorithm in a divide and conquer manner through MapReduce. This algorithm divides the dataset into subsets, so that parallel DBSCAN execution can be applied on multiple data partitions. The idea of MR-DBSCAN in 2D space is illustrated in Figure 4. Firstly, it maps the point data into small partitions by dividing the 2D space into rectangles, i.e. the rectangles with solid line borders in Figure 4. Since the results of local DBSCAN on each partition will ultimately be merged together, the nearby information needs to be added into each rectangle. Therefore, clusters across multiple partition rectangles can be identified and joined during the final reduce procedure. For this reason, each rectangle is extended by one-\( \varepsilon \) distance, i.e. those rectangles with a dotted border.

The simplest way for dividing a 2D space into rectangle cells is to equally divide the space into cells of the same size. However, if the data is heavily skewed in space, this approach can lead to extremely unbalanced workloads for parallel executions. Dividing the space according to the computation cost is an improved way to handle this problem in MR-DBSCAN. The computational complexity of DBSCAN is \( O(N \times m) \), where \( N \) is the number of points and \( m \) is the cost of establishing the number of neighborhoods. To apply DBSCAN on Spark, balanced-cost partitioning is needed to divide the space into cells that contain nearly the same number of spatial data elements. This partition method can however introduce many additional computing workloads. A trade-off between different partition methods needs to be considered according to the spatio-temporal distribution of the input data.
In this case study, we focus on Cost Balanced Partition (CBP) and utilize SMASH to improve and extend the implementation of MR-DBSCAN on Spark, i.e. to reduce the computation cost in generating partitions. Figure 5 illustrates this cost-based partition method. Initially, it divides space into \( n \times n \) cells of equal size \((n = 1/2\varepsilon)\). Then cells are grouped into partitions with almost equal numbers of data elements. This procedure is repeated until the maximum number of points per partition is exceeded.

[31] implement MR-DBSCAN on Spark by loading the whole dataset from a file system to Spark and subsequently calculate everything using Spark worker nodes. With the SMASH platform, it is possible to reduce many unnecessary calculations on Spark since the data are indexed and queryable in GeoMesa. For example, in building the CBP, we do not need to compare every single point of each cell to get the number of points in those initial cells as illustrated in Figure 5. If we input a large size of data with a small size of initial cells, it will require a lot of time in calculation of the partition strategy. In SMASH, we can query such statistical information from GeoMesa without a MapReduce phase with Spark. In addition, data do not need to be pushed into Spark during the space partition phase. They only need to be sent to the specific worker node for local DBSCAN processing after the partition phase. This can reduce the cost of data shuffling between the Spark nodes.

MR-DBSCAN introduces two additional input parameters to DBSCAN, namely the size of initial cells \( \nu \) and the maximum number of points in each partition \( \text{MaxP} \). The size of the initial cells directly affects the performance of CBP. In the worst case, if the number of points in an initial cell greatly exceeds the \( \text{MaxP} \) parameter, this procedure will not be able to build balanced partitions. Shrinking the size of the initial cells can help to tackle this problem but can lead to considerable computing workload. This scenario is not unusual when processing a large amount of historical Twitter data for example. There can be many points data crowded in a small area along the timeline, e.g. multiple spatio-temporal data are generated each day - often at the same location. To handle this problem, we extend the partition method to a 3D space model.

As a result, the spatio-temporal space is divided into several cube partitions as shown in Figure 6. Although adding an additional dimension into CBP can increase the computation complexity, this only has a slight impact on the SMASH based implementation since we only need to add a timestamp filter when querying the database.

We apply the MR-DBSCAN algorithm on 14 NeCTAR VMs and consider clustering of tweets spatio-temporally, i.e. identifying groups of tweets that are generated close to each other in both time and space. The input tweet dataset is collected from the Twitter REST APIs (http://dev.twitter.com/rest/public). The dataset used includes tweets that were created by random tweeters in Melbourne from 28/12/2016 to 15/01/2017. We filter out those tweets without GPS tags. The parameters we set for MR-DBSCAN are: \( \varepsilon_g=0.1 \) kilometers; \( \Delta t=10 \) minutes; \( s=10 \) meters per minute; \( \text{MinPts}=3 \); \( \nu=1 \) kilometers\( \times1 \) kilometers\( \times14 \) days; \( \text{MaxP}=100 \). After saving the clustering results into GeoMesa, we use GeoServer to generate a tiled map service for visualizing those clusters. Figure 7 illustrates one identified tweet cluster. This cluster is shown by the red points on the map and is located in Melbourne Cricket Ground on 01/01/2017 at 5:30 PM (Melbourne local time). Multiple users are involved in this cluster and many of them were posting contents related to a cricket game. The results show that this method can be used to identify where and when there is a crowd in the city and potentially what happens at that location based on the context of the social media contents or the landmark/location information of the cluster. Whilst not a traffic scenario per se, this demonstrates that SMASH can derive knowledge from arbitrarily identified clusters.
B. Applying DBSCAN Clustering on SMASH

In this case study, we use SMASH to analyze a large dataset from SCATS (nearly 80 GB). SCATS is deployed at a large scale across the roads of Victoria. Figure 7 shows an installation of SCATS sensors in the City of Melbourne. These sensors collect information of vehicles that drive over the SCATS closed loop sensors. The dataset we use here contains the volume of passing vehicles every 15 minutes for each sensor in Victoria. The data used covers data collected from between 2008 to 2014 with 176,226,786 raw records.

Since there is no directly overlapping time-period between this SCATS dataset and the tweets made on the road network in the time window of data collection, it is meaningless to compare them directly. Therefore, we aggregate this volume of data by days-of-week on SMASH to generate a statistical overview of the traffic for each street on each day of week. The results as shown in Figure 9 can be used to identify regular rush hour patterns. In this example, the traffic volume reaches the first peak at around 8:00AM with 563,550 vehicles. It reaches the second peak at around 5:00PM with 617,216 vehicles. A similar aggregation is applied to cluster tweet data and the combined results are shown in Figure 9 as a direct comparison to the SCATS results. Both the traffic volume and the number of tweets in the cluster show a minimum amount of activity around 3-4AM, but the SCATS volume grows much faster than tweet clusters in the morning. The number of tweet clusters reaches its peak at 7:00PM, which is 2 hours after the traffic peak time.

The previous methods used here are simple and can be implemented through Hadoop MapReduce tasks. However approaches like this have limitations in only supporting localized analyses. The correlation between tweet clusters and SCATS traffic is harder to explore from their results as shown in Figure 9 without SMASH capabilities.
C. Comparing the Results of Tweet Clusters and SCATS aggregation.

In above case studies, we apply two analysis methods to different datasets on SMASH. The raw data and processed results are all stored in the data layer of SMASH for further analysis. Those data/results can be visualized together on a map through the GeoServer component of SMASH. Figure 10 provides an example of visualizing these results on the same map. In Figure 7, we discovered a cluster of tweets, which had an id equal to 100318 at 6PM on 01/01/2017 (Melbourne local time). In Figure 10, we retrieve the tweet clusters generated from 4PM to 8PM on that date.

01/01/2017 is a Monday so we retrieve the traffic volume information from aggregated SCATS data on Monday at 6PM. The traffic volume is represented in shapes of lines on the map and we use red color for heavy traffic (specifically traffic volume over 50 vehicles every 15 minutes) and green for light traffic (i.e. less that 50 vehicles every 15 minutes). From Figure 10, we can identify that the traffic flows are usually heavy around Melbourne Cricket Ground on Monday at 6PM. Such similarity measures can also be applied to calculate the correlation between these results.

![Figure 11. Illustration of impact analysis from tweets cluster to SCATS traffic volume](image)

Figure 11 illustrates the combined analysis that we designed for analyzing the relation between tweet clusters and SCATS traffic volumes. First, tweets in each cluster are map-reduced to a single geographic centroid point with an average timestamp and a distance range. These centroid points are used to represent the clusters, i.e. the transparent orange circles in Figure 11. Next, we use these centroids and their timestamps to generate queries to fetch nearby SCATS volume data.

Finally, we compared the retrieved SCATS traffic volumes to the average tweet cluster values, which can be generated via MapReduce on SMASH. If the retrieved nearby SCATS volume is higher than the historic value, it can help to identify the strength of the spatio-temporal correlation between official traffic volumes from SCATS and social media data when used as a proxy for prediction of traffic flows and road congestion.

V. CONCLUSIONS

In this paper, we present a Cloud-based SaaS solution for urban transport analysis. The novelty of this solution is that it is not designed for a specific traffic analysis method, but rather it is designed as a generic platform that can be used for implementing a variety of traffic research approaches over different big datasets in an efficient and reliable manner. We demonstrate how the platform supports MR-DBSCAN [30] and a performance-oriented Spark implementation. In the case studies, we cluster tweet data in space and time through the DBSCAN algorithm. We also illustrate the capability of SMASH for exploring combinations of localized correlations among diverse traffic-related data.

For the future, we intend to extend the use-case of SMASH on urban transport area. For example, adopting image processing methods for recognizing traffic flow on photos taken by the satellite; applying DBSCAN on real-time spatio-temporal data streams as well as applying traffic modeling algorithms. Another aspect of future work is to review the infrastructure and add new functionalities for on-demand scaling, e.g. adding software components for auto-scaling the platform; benchmarking the performance between the Spark streaming engine and comparing this with other solutions. New computing engines could also be added into SMASH for better performance in tackling real-time big data streams.

Acknowledgements

The authors would like to thank the NeCTAR project for the Cloud resources that have underpinned this resource. The authors also acknowledge VicRoads for access to and use of the SCATS data.

REFERENCES


