Automatic threshold computation for traffic incident detection using INRIX

by

Han-Shu Chang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Carl K. Chang, Major Professor
Anuj Sharma
Ying Cai

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

Copyright © Han-Shu Chang, 2019. All rights reserved.

DEDICATION

I would like to dedicate this Creative Component Report to my parents who have always backed me up unconditionally. I would also like to thank my friends and family for their loving guidance and financial assistance during the writing of this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this report. There are two professors who inspire, support and guide me throughout the research. Firstly, Dr. Carl K. Chang for his guidance, patience, and support throughout my degree pursuit and the writing of this Creative Component Report. Secondly, Dr. Anuj Sharma for his inspiration, guidance, and support throughout this research and the writing of this Creative Component Report. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Ying Cai. I would additionally like to thank Pranamesh Chakraborty, Umesh Chinalachi, Lakshay Ahuja and all other friends for their guidance and help during the course of this project.

ABSTRACT

Traffic congestion in freeways poses a major threat to the economic prosperity of the nation. It not only causes loss of productivity of the workforce but also frustration among the drivers. Traffic incidents such as vehicle crashes, overturned trucks, and stalled vehicles contribute to a significant amount of non-recurrent congestion. Automatic Incident Detection (AID) algorithms have been developed for detecting such incidents in real-time and alerting the drivers. Such incident detection algorithms often rely on generating thresholds based on historical traffic patterns. However, large-scale historical traffic data often poses to be a major challenge in processing these data and automatically generate thresholds for incident detection. This project leverages lambda architecture and cloud computing service to develop an automatic threshold computation module. It is based on the Inter-Quartile Distance (IQD) method for threshold computation and runs on a weekly basis using previous 8 weeks of historical data, amounting to more than 250 GB of traffic data. The incidents detected using the real-time traffic data and the automatically generated thresholds can be used by Traffic Management Centers for real-time incident detection and further analysis such as incident validation and performance tuning.

CHAPTER 1. INTRODUCTION

1.1 Motivation

There are over 37,000 people die in road crashes and an additional 2.35 million are injured or disabled each year. Moreover, Road crashes cost the U.S. $230.6 billion per year, which is an average of $820 per person. With the increase of population, managing traffic conditions grows importance in commuters, goods or service delivery and various activities. Traffic congestion has become a major threat to a nation's economic prosperity. It is often that traffic congestion was classified into two categories: recurrent congestion and non-recurrent congestion. (Ozbay and Kachroo (1999); Anbaroglu et al. (2014)) Recurrent congestion is commonly observed during rush hours, which demonstrates a daily pattern for the traffic flow of a region. Whereas non-recurrent congestion is often caused by traffic incidents such as vehicle crashes, overturned trucks, and stalled vehicles. It is these kinds of events that often lead to travel time variability (Noland and Polak (2002)) and often cause frustration and road rage to the commuters. (Tang and Gao (2005))

In order to restore the smooth traffic flow, it is vital to have accurate and early detection of traffic incidents. An Advanced Traffic Management Systems (ATMS) is a software or system that typically adopts integrated approaches, or uses new communication, control, sensing and data-processing technology to solve traffic congestion problems. (Tang and Gao (2005); Srinivasan et al. (2004); Ikeda et al. (1999); Farradyne (2000)) There are existing open source ATMS like IRIS (Intelligent Roadway Information System). But the installation requirements and accessibility of such ATMS are not as friendly as a public website. Additionally, as time goes by, more and more advanced information and storage technologies have emerged. Thus the project, TIMELI (Traffic Incident Management Enabled by Large-data Innovations), is born and it is an ongoing web-based ATMS software built by the Institute of Transportation (InTrans). We will be giving a more detailed background of TIMELI in the next chapter.

Traffic incidents are not only a major contribution of non-recurrent congestion but also it is found that if an incident lasted one more minute, traffic delay time or congestion would amount to $4 \sim 5$ min in a non-rush hour. (Liu and Yang (1998);Wu et al. (2000)) Hence, automatic incident detection (AID) has been established to be a crucial technology for detecting traffic incidents and for the reduction of non-recurrent traffic congestion.

In this project, we develop a threshold computation module that can be used by the traffic incident detection framework in TIMELI. Previously in the project, the incident detection module is developed. However, the threshold data where the real-time traffic data is compared with is still static. Based on the Inter-Quartile Distance method, it is more accurate to use dynamic threshold data over static data. The TIMELI project and the design of the module follow the lambda architecture in order to process large-scale historical traffic data. The traffic data is collected every minute per day for each segment of an interstate road. Therefore, it amounts to 250GB $\sim$ 480 GB of traffic data. This module utilizes the Azure Data Factory, Azure Blob Storage and on-demand Hadoop cluster as the technologies to process the large-scale data. And based on the Inter-Quartile Distance method (IQD) proposed by Chakraborty et al. (2019), the module is generating the threshold data on a weekly basis, using the previous 8 weeks of historical data as the input. The threshold data is then compared with the real-time traffic data stream and then find out the traffic incidents. These incidents detected are stored in cloud storage right away. They were marked on TIMELI notifying the Traffic operator to handle them and were stored for further analysis such as incident validation and performance tuning.

1.2 Report Organization

The rest of this report is organized as follows. Chapter 2 gives an overview of the Automatic Incident Detection(AID) and Traffic Incident Management Enabled by Large-data Innovations(TIMELI) project developed at the Institute of Transportation (InTrans). Chapter 3 provides detail architecture of TIMELI, explains the AID framework implemented and technologies applied

for the project. Chapter 4 shows the results of the tests conducted. Chapter 5 then explains the future scope of continuing the development and the conclusion of this project.

CHAPTER 2. OVERVIEW

Threshold Computation Module is a prerequisite of the Traffic Incident Detection component using INRIX data built by Byna (2019). It is a component of TIMELI project developed by InTrans. In this chapter, the first section provides a brief description of TIMELI, then the second section gives background information about Automatic incident detection (AID) so that it helps with giving an in-depth understanding of what role Threshold Computation Module plays in TIMELI project.

2.1 TIMELI

Inspired by other Advanced Traffic Management System such as IRIS, TIMELI (Traffic Incident Management Enabled by Large-data Innovations) is a software which integrates innovative data analytic techniques in order to monitor traffic conditions in real-time, proactively control risk, quickly detect traffic incidents, identify the location and potential cause of these incidents and suggest traffic control alternatives. This software will be provided to the Department of Transportation (DoT) where the operators will be alerted about any potential incident going in the area. The operators can then manually verify whether it is an incident or not by checking the live video feeds in that area. In order to identify a potential incident, TIMELI uses various data sources which provide the different types of information related to the traffic on road. In the Traffic Incident Detection module using INRIX data, the road network split into multiple smaller segments. Each segment produces a time-series of the traffic state (average speed). These time-series are extensively large-scale, with thousands of data points being recorded daily per segment. We leverage this data and identify potential congestion detection. Once the incident is detected, an alert is shown to the operator on the screen along with the nearest cameras available to that incident. The operator then confirms whether its an incident or not and take steps accordingly.

2.2 Automatic Incident Detection (AID)

In the course of recent decades, there have been many AID algorithms developed to detect the occurrence of incidents by detecting the change of traffic flow parameters measured at upstream and downstream detector stations (Farradyne (2000)). Principally, AID algorithms can be classified into four classes (Huang (2001)) :

1) prediction algorithms; 2) mode identification algorithms; 3) methods based on a traffic flow model or theory; and 4) incident detection using computational intelligence, such as neural networks, fuzzy logic, or image-based processing algorithms. (Tang and Gao (2005))

A Prediction algorithm is typically a statistical procedure to predict or forecast the traffic flow. It first summarizes the traffic flow parameter (constant) from the historical traffic data and then it compares the predicted value with the current data to obtain their differences and determines whether the differences are beyond a predefined threshold. Standard Deviation Algorithm (SND) is one of the major algorithms in the literature. Using the similar concept of SND algorithm, Chakraborty et al. (2019) proposed the IQD method and which proves to be more robust and can be easily parallelized to a larger road network. In this report, we are applying the IQD method for threshold computation and deploy such a module in the cloud computing environment.

CHAPTER 3. THRESHOLD COMPUTATION DESIGN AND DEPLOYMENT

This chapter begins with the detail information about the architecture designed for TIMELI project whom the module proposed in this paper belongs to. Then, we go through the Traffic incident detection algorithm and explain the inter-quartile deviation(IQD) method used for threshold computation. Before going through the logic and deployment of the threshold computation module, we cover the technologies used. Finally, the input and output data is explained in detail and the overall automatic incident detection(AID) framework is concluded.

3.1 TIMELI project Architecture

3.1.1 Lambda architecture

Lambda architecture is a data-processing architecture that features processing data with both batch and stream-processing methods to handle a massive quantity of data. With the continuous incoming data feeding to the system, Lambda architecture has Speed Layer, Batch Layer, and Serving Layer to satisfy the need for a robust, fault-tolerant, large scale data processing system. Both Speed Layer and Batch Layer process the data and generate the result. Speed Layer focus on producing real-time data and stores the quickly generated results in a short time-to-live database. It compensates for the high latency on the processing by the Batch Layer. Batch Layer manages an immutable, append-only master dataset that stores the raw data permanently and pre-compute the batch views. The Serving Layer is responsible for indexing and exposing the views so that they can be queried.(Hausenblas and Bijnens ( ); Jameskinley (2012); Marz and Schuster ( )) A general Lambda architecture is shown in Fig 3.1.

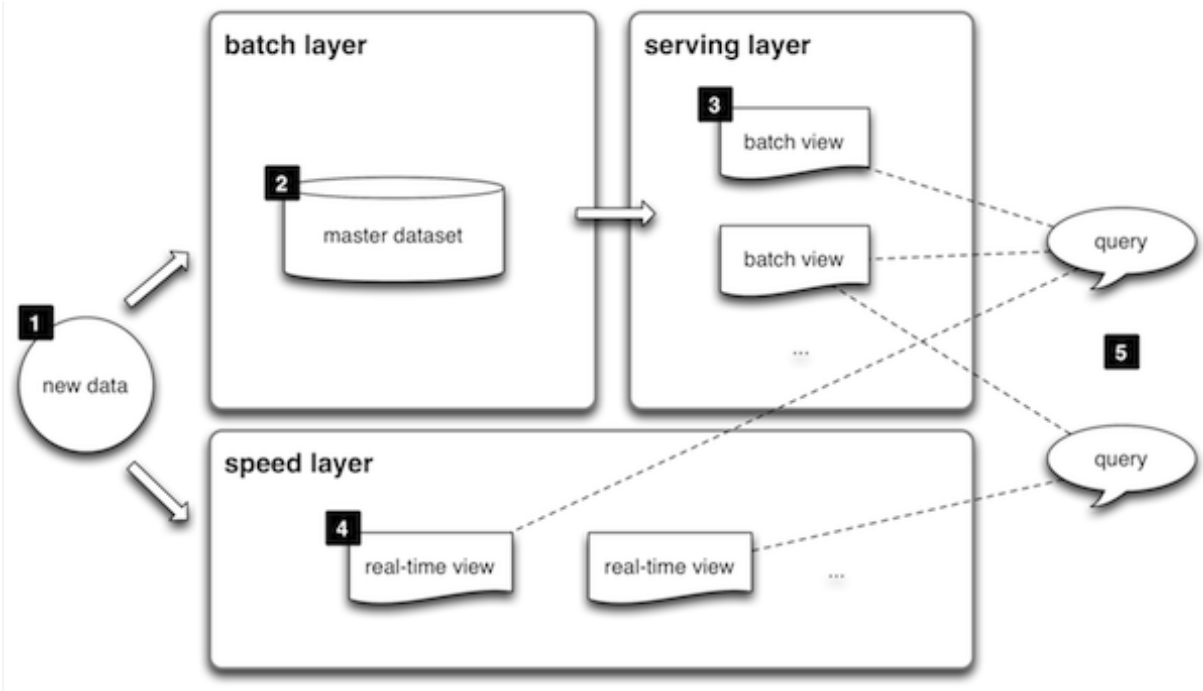

Figure 3.1 Lambda Architecture

3.1.2 Overview of the TIMELI project architecture

The architecture of TIMELI follows Lambda architecture. As shown in Figure 3.2. In the figure, there are a variety of data streams feeding to the system from the left. While in this module we will focus on the incident detection method to the INRIX source. The upper part colored yellow belongs to the Speed Layer, and the lower blue part belongs to the Batch Layer. The component labeled 8 is the TIMELI web application that presents the result to users, which is the Serving Layer of the architecture. The implementation of Lambda architecture expects the Batch Layer and Speed Layer to process and generate the same results. Figure 3.3 is the legends descriptions of the architecture.

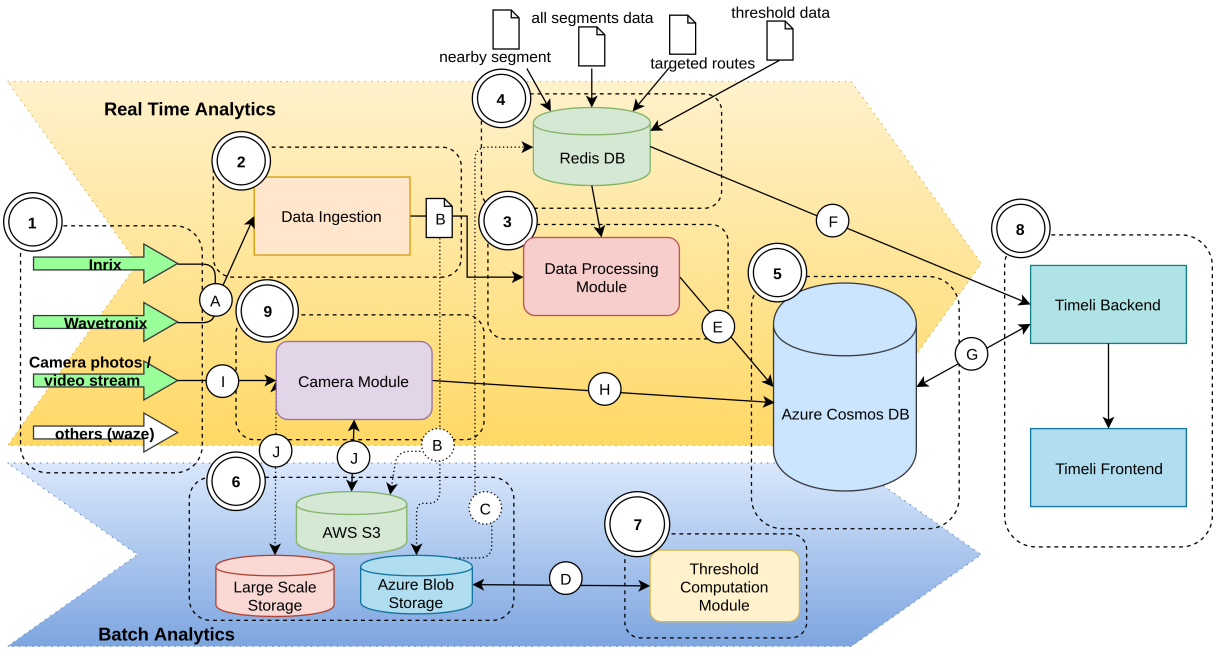

Figure 3.2 TIMELI Architecture

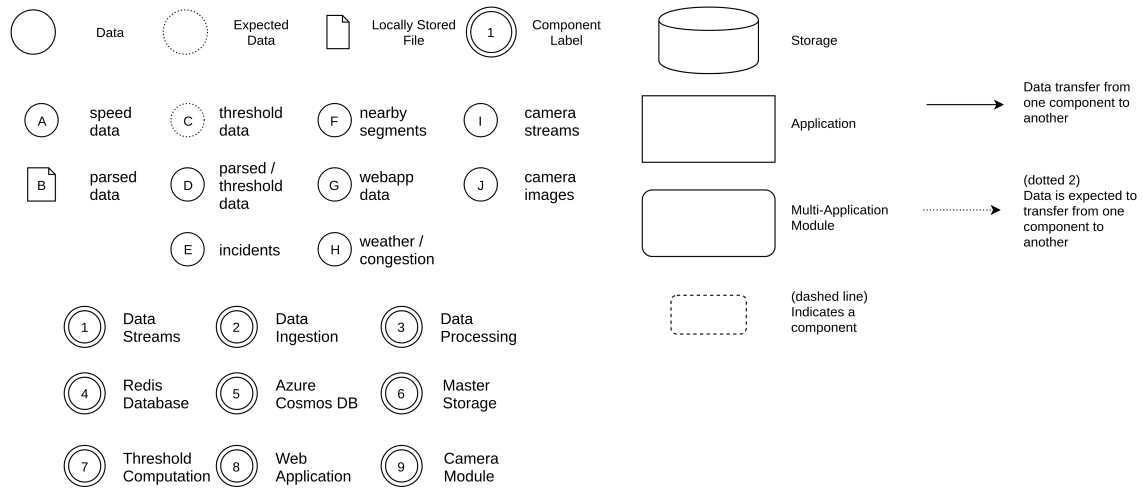

Figure 3.3 TIMELI Architecture legends

3.2 Threshold Computation Module

3.2.1 Traffic incident algorithm framework

The average size of the raw speed data per day is around 4 GB. The threshold computation module takes in historical 8 weeks (2 months) of per day data and summarizes the results. In order to process such large data and update the threshold every week, the Threshold Computation Module needs a master dataset to keep providing input data and stores the output results. As a result, it belongs to the Batch processing Layer. Here is a quick overview of the data flow of this Threshold Computation Module shown in the TIMELI architecture figure 3.2. First, the Data Ingestion module(labeled 2) fetches the INRIX data stream, extracts required information and stores the per day speed data to the master dataset – Azure Blob Storage(data flow labeled C). Finally, at the frequency per week, the Threshold Computation module(labeled 7) computes and updates the weekly threshold data, stores back to the Azure Blob Storage.

Automatic incident detection (AID) has been identified to be a critical technology to reduce non-recurrence traffic congestion.

According to the report by Byna (2019), the core part of traffic incident detection algorithm is as follow:

1. The location where the segment speed $<$ threshold speed will be marked as a *potential incident*.

   (a) If the incoming stream data reflects that the location as *potential incident* for consecutively 3 times, then it is considered as an incident.

   (b) If the location has is already been detected as an incident, and the streaming speed data reflects that the location as *potential incident* for consecutively 3 times, then consider it the same incident detected last time.

2. The location where the segment is considered to have an incident in the previous minute and the segment speed $>$ threshold speed will be marked as a *may be over*.

(a) If the streaming speed data reflects that the location as *may be over* for consecutively 5 times then the incident on that segment is considered to be cleared.

The above traffic incident detection is executed per minute in Data Processing Module(labeled 3) in figure 3.2.

3.2.2 IQD method for threshold computation

After knowing the core incident detection logic we can find that the core of this algorithm is how the threshold speed is determined.

In the paper by Chakraborty et al. (2019), they suggest threshold computation can be done using the following concepts: Normal traffic condition varies depending on the time of the day and the day of the week. Therefore the algorithm involves modeling the univariate statistics of each non-overlapping windows of the time series data as a Laplace distribution with location parameter μ replaced with the mean speed value ($\bar{x}$) and scale parameter ζ replaced with the corresponding standard deviation(σ), which are determined for each day of the week, and 15-min periods of the day for each segment. In other words, threshold speed is a summarizing of the normal traffic flow at each 15 minutes window for each day.

Threshold speed values over 15-min intervals of each day of the week are determined using the previous 8 weeks of traffic data for the same segment for the given day of the week and period of the day, similar to the study of Balke et al. (1996).

Using the popular univariate Standard Deviation(SND) algorithm (Dudek et al. (1974); Balke et al. (1996)), threshold speed ($\tau^{p,d}_{s,snd}$) can be calculated as this:

$$\tau^{p,d}_{s,snd} = \bar{x}^{p,d}_{s,snd} - (c_{snd} * \sigma^{p,d}_{s,snd})$$

where s is the segment, p is a 15-min non-overlapping window, d is the given day of a week, and the optimum constant c is to be determined from the validation set. This threshold speed ($\tau^{p,d}_{s,snd}$) can be compared with real-time speed data to detect anomalies or traffic incidents.

Although SND algorithm is robust, it is known to be susceptible to outliers. Especially the algorithm is found to be affected by occasional low speeds caused due to incidents or traffic constructions. It can lead to misrepresentation of the underlying normal traffic patterns. To alleviate this issue, Hampel provides the maximum absolute deviation and Chakraborty et al. (2019) suggests that the alternative inter-quartile deviation(IQD) and was shown to be an alternate robust summary statistics (learned from the time series tensor itself).

1. MAD algorithm: μ = Median (M), ζ = Maximum Absolute Deviation (MAD),

$$\tau_{s,mad}^{p,d} = M_{s,mad}^{p,d} - (c_{mad} * \sigma_{s,mad}^{p,d})$$

2. IQD algorithm: μ = Median (M), ζ = Inter-Quartile Distance (IQD),

$$\tau_{s,iqd}^{p,d} = M_{s,iqd}^{p,d} - (c_{iqd} * \sigma_{s,iqd}^{p,d})$$

The results have shown that the IQD method can obtain higher incident detection rates without compromising too many false alarms. Please find next chapter or refer to Chakraborty et al. (2019) to learn more information.

3.2.3 Technology Used

3.2.3.1 Apache Hadoop and Apache Pig

In this generation where information and data explode, the vast quantity of data was simply too large to pump through the database bottleneck. Therefore, there comes a MapReduce algorithm that typically chops data into smaller chunks, and when the calculations are done, it brings back and merges them together to generate the resulting dataset. (Dean and Ghemawat (2008)) The open-source project, Hadoop, is developed to allow applications to run with the MapReduce algorithm.

Apache Hadoop is a solution for Big Data that deals with complexities of high volume, velocity, variety of data. It is a whole ecosystem of projects that work together to provide a common set of services. Hadoop enables hardware to provide services to store petabytes of data reliably and allows huge distributed computation across clusters. The core of Apache Hadoop consists of a data storage part, known as **Hadoop Distributed File System (HDFS)**, and a

processing part which is a **MapReduce programming model**. HDFS is a file system that stores all the data in directories for Hadoop. The files don't have strict requirements on the schema, and it scales out to petabytes of storage. The MapReduce programming model manages and executes the jobs assigned, then produce the results to be merged. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the massive data in parallel. Hadoop is reliable for errors and easily scalable for large scale data. Therefore it is suitable for our needs.

To run jobs that require access and process massive files on HDFS, the next paragraph introduces a platform called Apache Pig.

Besides Java, Pig is another language in which the MapReduce program can be written. **Apache Pig** is a high-level scripting platform that runs on Hadoop Cluster, designed to process and analyze large data sets. It consists of two components: Pig Latin script and Runtime Engine. Pig Latin is a language that a programmer can give syntax and commands to handle procedural data flow operations. The runtime engine compiles, validates and optimizes the code, generates and executes jobs and plans, also interacts with the Hadoop system (HDFS and MapReduce) to finish the massive data computation. Pig has the features that make the code extensible, self-optimizing, and easy to program. Users can write the code without knowing Java. Pig supports both structured and unstructured data as an input to perform analytics and uses HDFS to stores the results. [cite Apache Pig] Thus, with Apache Pig, the summary statistics values from raw traffic speed data (mean, median, MAD, and IQD) can be easily parallelized over multiple systems thereby enabling the AID framework to handle massively large dataset over wide traffic network.

3.2.3.2 Azure Data Factory(ADF)

Azure Data Factory is a cloud-based integration service that orchestrates and automates the movement and transformation of data at scale. It offers a code-free UI for intuitive authoring, monitoring, and management. Data-driven workflows (called pipelines) can be created and scheduled to ingest data from various data stores. It supports building complex ETL(Extract, Transform,

Load) processes that transform data visually with data flows or by using compute services such as Azure HDInsight Hadoop, Azure Databricks, and Azure SQL Database. Conclusively, raw data can be formulated into meaningful data stores and data lakes for better business decisions through Azure Data Factory. (Djpmsft ( ))

Azure HDInsight is a managed, full-spectrum, open-source analytics service in the cloud for enterprises. The most popular open-source frameworks or development environments such as Hadoop, Spark, Hive, LLAP, Kafka, Storm, R, and more are available to use. With these frameworks, it offers a wide range of service scenarios such as ETL, data warehousing, machine learning, and IoT. Azure HDInsight has the advantage to spin up big data clusters on-demand, scale them up or down based on your usage needs, and pay only for what you use. HDInsight also integrates seamlessly with other Azure services, including Data Factory and Data Lake Storage, for building comprehensive analytics pipelines. As of security, HDInsight meets industry and government compliance standards and protects your enterprise data assets using an Azure Virtual Network, encryption, and integration with Azure Active Directory. Azure HDInsight makes it easy, fast, and cost-effective to process massive amounts of data. (Hrasheed-Msft ( )) In this project, we create an on-demand Azure HDInsight Hadoop Cluster for computation job in the ADF we created.

3.2.4 Module Architecture

This project is developed on an on-demand Hadoop HDInsight cluster which Azure Data Factory provides. Figure 3.5 shows how the threshold computation module is deployed. We use Azure Blob Storage, Microsoft's object storage solution which is optimized for storing massive amounts of unstructured data or media in the cloud. First, historical traffic speed data is collected in Azure Blob Storage daily with a size of around 4 GB. At the frequency of every week, Azure Data Factory (ADF) creates a pipeline that is scheduled to ingest past 8 weeks of data from Blob storage and launches an on-demand Hadoop HDInsight cluster and generate final threshold data. Inside the Hadoop cluster, we executed the Apache Pig script to perform MapReduce for the IQD algorithm described in section 3.2.2. The results threshold output will pipe back and append in

the Blob Storage. Also, the threshold will be uploaded to the RedisDB (labeled 4) in the TIMELI architecture 3.2 in order for Data Processing Module(labeled 3) to perform incident detection. Figure 3.4 is the UI view of the ADF pipeline.

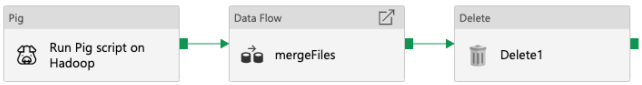

Figure 3.4 ADF pipeline for threshold computation

The threshold computation module involves the storing immutable, constantly growing master dataset(Azure Blob Storage) and batch analyze on those data. Therefore it belongs to the batch layer mentioned in the lambda architecture in section 3.1.1.

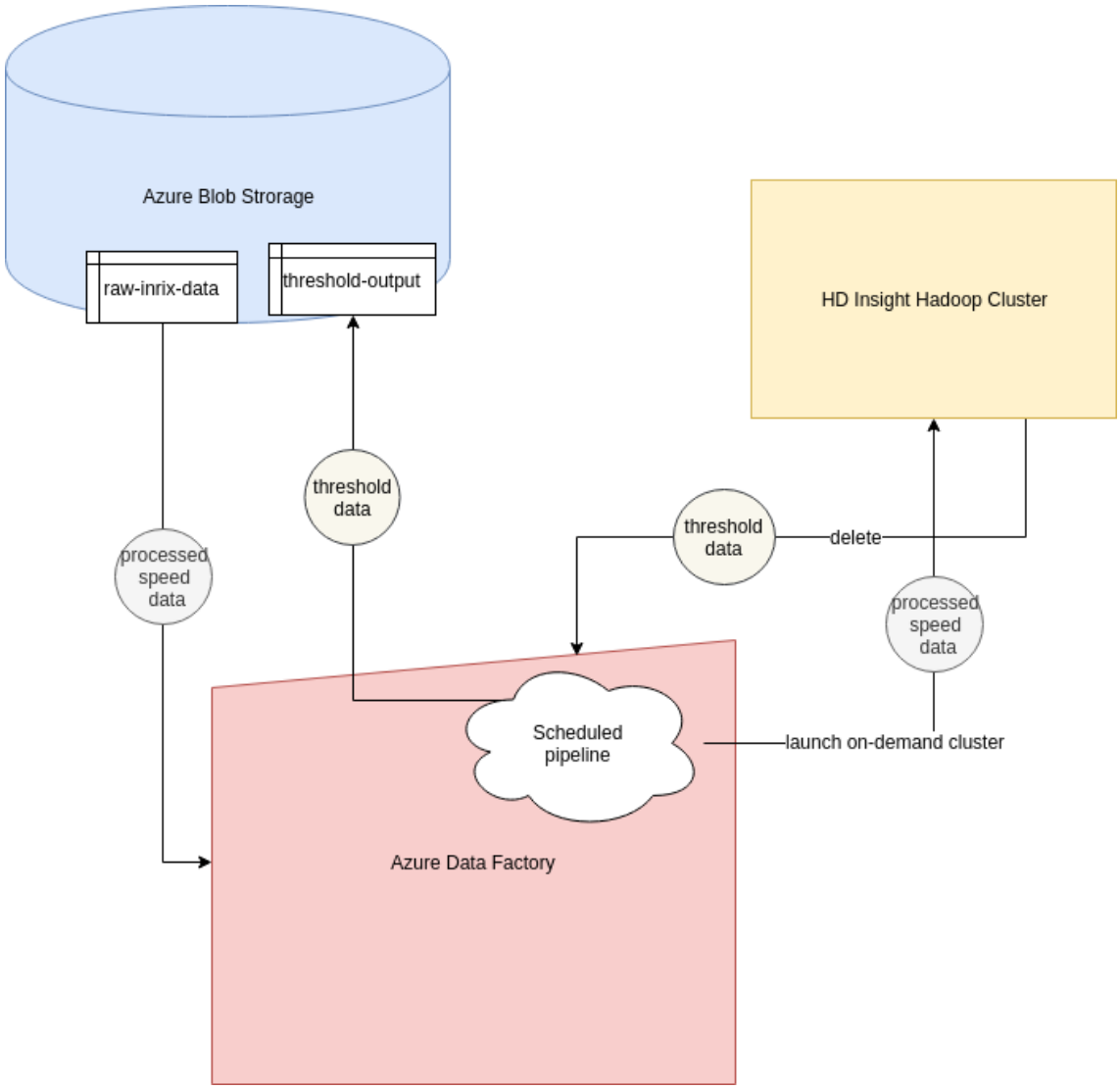

Figure 3.5 Threshold Computation Module Architecture

3.2.5 Input and Outcome Explain

3.2.5.1 Input: INRIX

INRIX is a global SaaS and DaaS private company in Kirkland, Washington that specializes in connected car services and transportation analytics. INRIX provides APIs over HTTP to access trillions of bytes of information about roadway speeds and vehicle counts from 300 million real-time

anonymous mobile phones, connected cars, trucks, delivery vans, and other fleet vehicles equipped with GPS locator devices. For this project, Iowa speed data is retrieved from Inrix, along with location and incident data. Speed data is used to detect congestion, location data is used to plot the incidents on the map and incident data is used to compare the predicted incidents with actual incidents predicted by Inrix. (Inrix ( )) The road network in Iowa is divided into segments. Inrix speed data represents the speed of each of the segments on the road. Each segment is on an average of 0.5 miles. The input data consists of the following 5 columns:

code: unique id of each segment

c-value: confidence value

score: confidence score

speed: speed of the segment

timestamp: the time when this data was retrieved

3.2.5.2 Output: threshold speed

As depicted in the above section, the traffic pattern is disparate at different times of the day and day of the week. Thus, The output threshold speed or the summarization of the historical traffic pattern consists of following fields:

code - segment_id

weekday - the day of the week (Mon, Tues, ..)

hour - hour of the day (0, 1, ..., 23)

period - 15-minute period of each hour (0,1,2,3)

median - median speed

iqd - Inter-quartile distance

3.2.6 Overall AID framework

The flow of the AID algorithm framework is depicted as follows. Historical traffic data are processed weekly in MapReduce for univariate speed thresholds computation of 15-minute windows

for each segment and day-of-week. The speed thresholds can be generated using SND, MAD, or IQD method. Next, the thresholds are matched with real-time speed data and when speed is less than the threshold value for 3 consecutive intervals, an incident alarm is triggered. This persistence test is performed to reduce false alarms generated due to spurious noise in real-time data. While increasing the duration of persistence test will result in further delay in detecting incidents, decreasing the duration can lead to an increase in false alarms due to the noisy real-time speed values. Hence, we used three intervals similar to the previous studies (Li et al. (2013); Ren et al. (2012)).

CHAPTER 4. RESULTS

In this chapter, we first present a brief comparison of the IQD method versus others and then show the processing time of the overall framework.

4.0.1 IQD method performance

The dataset used for the threshold computation algorithm is the traffic speed data and crash data from Interstate Freeways I-80/35 and I-235 of the Des Moines region, in Iowa, USA from April 2017 to October 2017.

The algorithm involves the following 4 performance measurements:

1. Detection Rate (DR) is the ratio of the number of incidents detected by the AID algorithm to the total number of incidents that occurred.

2. False Alarm Rate (FAR) is the ratio of the total number of false alarm cases by the total number of algorithm applications.

3. Mean Time to Detect (MTTD) is the average time elapsed between the actual start of the incident and time when the incident is first detected by the algorithm, which takes into consideration the latency involved in the AID algorithm.

4. Performance Index (PI), formula listed below, brings together all 3 performance measures (DR, FAR, and MTTD) into a single measure to find out the overall performance of the AID algorithm. PI is believed to be one of the best possible measures to reflect the performance of AID during model selection (Ren et al. (2012); Luk et al. (2010); Cheu et al. (2003)). Minimizing PI is the optimization objective used during cross-validation.

$PI = (1.01 - \frac{DR}{100}) \times (\frac{FAR}{100} + 0.001) \times MTTD$

Table 4.1 shows that the resulting IQD method at constant 2.2 can achieve good DR without producing too many false calls. The MTTD loses to the MAD method a bit, but when looking at

the combined performance with all 3 measures depicted by PI, the IQD method is suggested to be the better choice with the least value.

This project's main objective is to implement and deploy the algorithm in the cloud environment. If interested in learning more detail explanations and experiments conducted be sure to refer to Chakraborty et al. (2019).

Table 4.1 Algorithm Results Comparison

Algorithm	**DR**	**# false alarm/day**	**MTTD(mins)**	**PI**	**constant**
IQD	97.1%	4.1	12.4	0.0018	2.2
	>SND	≃SND	<SND	<SND	
	≃MAD	<MAD	>MAD	<MAD	

4.0.2 Processing results

The module takes in 3 months data, filter to remain 8 weeks data, then compute to summarize the threshold data on a weekly basis. The processing results of the module are shown in table 4.2 and Figure 4.1 is the processing statistic shown on ADF User Interface. Figure 4.2 shows a sample of resulting threshold speed heatmap.

Table 4.2 Processing results

Node	**Input Size**	**Time**	**Output Size**
4	02:05:00	368GB	48MB

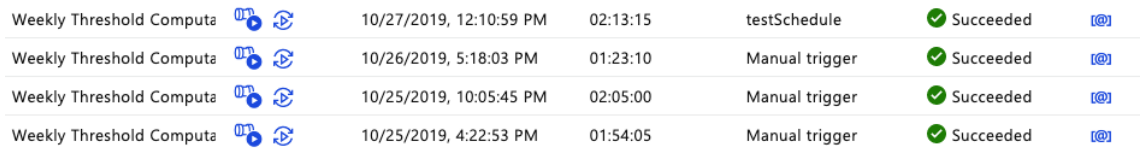

Figure 4.1 Module processing statistics

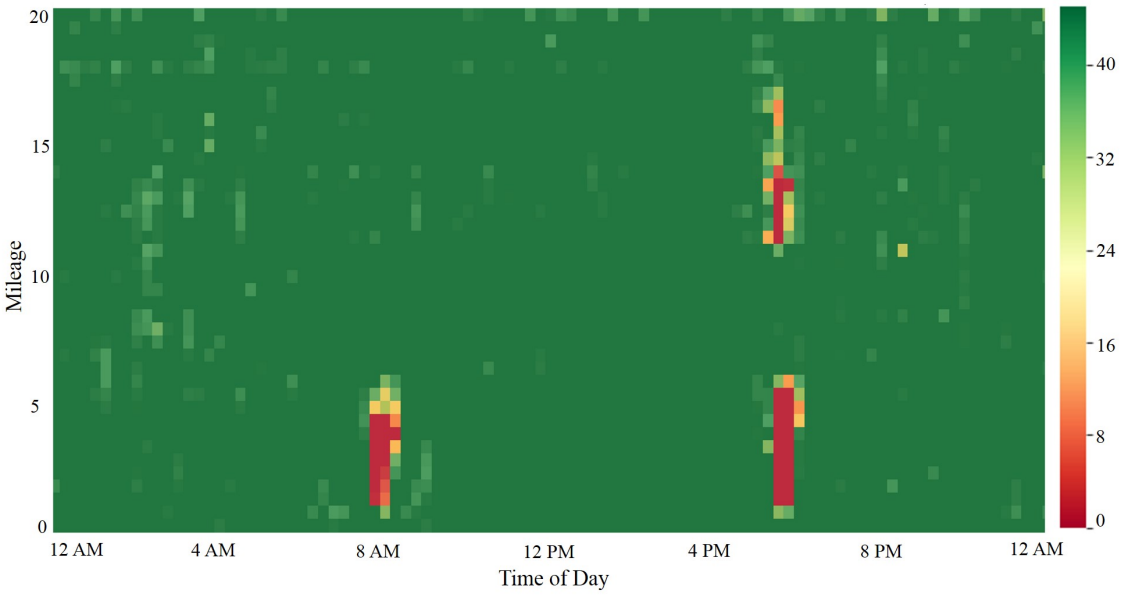

Figure 4.2 Sample speed threshold heatmap (mph)

CHAPTER 5. FUTURE SCOPE AND CONCLUSION

This chapter suggests some directions to the future development of the TIMELI project and concludes this threshold computation module.

5.0.1 Future scope

5.0.1.1 More of threshold computation

Currently, the Univariate speed threshold determination method is used on INRIX data. In the paper by Pranamesh (Chakraborty et al. (2019)), there is multivariate Spatio-Temporal Threshold denoising that further enhance the threshold data. Additionally, the sensor data provided by Wavetronix can be used to compute its threshold data, so that both sources can provide their detection on the road to strike a better accuracy.

5.0.1.2 Lambda archtecture

To follow the lambda architecture, the functionality adheres to each layer need to be further designed. For example, the temporary storage used for the speed layer is CosmosDB has mixed functionality to store historical incident data and speed data. There is a cleanup module and a communication module needed that bridge the Azure Blob Storage, which is the role of an append-only master dataset in the system. Apart from that, the parallelism of batch processing and real-time processing need to be considered its use case in order to be implemented. Currently, the incident detection algorithm is processing in real-time, and the threshold computation is in a batch process. The necessity for having both layer process the same detection is worth consideration and implementation.

5.0.1.3 Road Traffic Dashboard

With all appended historical incident data, it will be nice to generate a monthly or annually dashboard displaying the statistics about the traffic pattern, congestion time, cost, detection analysis for different sources. This can give a better insight into the traffic flow observed on the road.

5.0.1.4 Monitoring

There have been cases when the modules stop running because of no space left on device, exceptions in the code and there are no monitoring mechanisms for it. Use Azure Alert monitoring or Nagios to monitor the running processes on these machines.

5.0.2 Conclusion

Incident detection is required to detect congestion and stalled vehicles on the roads. The impact of an incident is severe and it is required to leverage as many sources as possible to detect them with accuracy, speed, and efficiency.

In this work, we have developed a module that summarizes the traffic pattern by analyzing the past traffic data. With the outcoming threshold data, we detect traffic incidents or anomaly in real-time. Even though it is a part of TIMELI project, the deployment of the cloud services makes it ideal to extend or separate it as an independent project which gives us the flexibility to integrate it with other projects.

As discussed in the last section, there are plenty of thoughts on the improvement. The continuation of the project requires more effort and thorough use cases inspections and user feedback.

BIBLIOGRAPHY

Anbaroglu, B., Heydecker, B., and Cheng, T. (2014). Spatio-temporal clustering for non-recurrent traffic congestion detection on urban road networks. *Transportation Research Part C: Emerging Technologies*, 48:47–65.

Balke, K., Dudek, C. L., and Mountain, C. E. (1996). Using probe-measured travel times to detect major freeway incidents in Houston, Texas. *Transportation Research Record*, 1554(1):213–220.

Byna, R. R. (2019). Traffic incident detection using inrix data. –, 147.

Chakraborty, P., Hegde, C., and Sharma, A. (2019). Data-driven parallelizable traffic incident detection using spatio-temporally denoised robust thresholds. *Transportation Research Part C: Emerging Technologies*, 105:81–99.

Cheu, R. L., Srinivasan, D., and Teh, E. T. (2003). Support vector machine models for freeway incident detection. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, volume 1, pages 238–243. IEEE.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Djpmsft (–). Introduction to azure data factory - azure data factory.

Dudek, C. L., Messer, C. J., and Nuckles, N. B. (1974). Incident detection on urban freeways. *Transportation Research Record*, 495:12–24.

Farradyne, P. (2000). Traffic incident management handbook. *Prepared for Federal Highway Administration, Office of Travel Management*.

Hausenblas, M. and Bijnens, N. (–). Lambda architecture.

Hrasheed-Msft (–). What are the apache hadoop and apache spark technology stack? - azure hdinsight.

Huang, H. (2001). *The Investigation of traffic incident detection algorithm*. PhD thesis, Ph. D. dissertation, China.

Ikeda, H., Kaneko, Y., Matsuo, T., and Tsuji, K. (1999). Abnormal incident detection system employing image processing technology. In *Proceedings 199 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems (Cat. No. 99TH8383)*, pages 748–752. IEEE.

Inrix (–). About.

Jameskinley (2012). The lambda architecture: principles for architecting realtime big data systems.

Li, X., Lam, W. H., and Tam, M. L. (2013). New automatic incident detection algorithm based on traffic data collected for journey time estimation. *Journal of transportation engineering*, 139(8):840–847.

Liu, W. and Yang, Z. (1998). Control methods for high freeway systems. *Beijing, China: Renmin Jiaotong*.

Luk, J., Han, C., and Chin, D. A. (2010). Freeway incident detection: technologies and techniques. –.

Marz, N. and Schuster, W. (–). Nathan marz on storm, immutability in the lambda architecture, clojure.

Noland, R. B. and Polak, J. W. (2002). Travel time variability: a review of theoretical and empirical issues. *Transport reviews*, 22(1):39–54.

Ozbay, K. and Kachroo, P. (1999). Incident management in intelligent transportation systems. –.

Ren, J. S., Wang, W., Wang, J., and Liao, S. (2012). An unsupervised feature learning approach to improve automatic incident detection. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 172–177. IEEE.

Srinivasan, D., Jin, X., and Cheu, R. L. (2004). Evaluation of adaptive neural network models for freeway incident detection. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):1–11.

Tang, S. and Gao, H. (2005). Traffic-incident detection-algorithm based on nonparametric regression. *IEEE Transactions on Intelligent Transportation Systems*, 6(1):38–42.

Wu, Y., Wang, Y., and Cao, J. (2000). The investigation on automatic incident detection algorithm of traffic accident. *Manuf. Automat*, 22(12):42–45.