Cluster Computing manuscript No.
(will be inserted by the editor)

Fangzhou Sun · Abhishek Dubey · Jules White · Aniruddha Gokhale

Transit-Hub: A Smart Public Transportation Decision Support System with Multi-timescale Analytical Services

Abstract Public transit is a critical component of a smart and connected community. As such, citizens expect and require accurate information about real-time arrival/departures of transportation assets. As transit agencies enable large-scale integration of real-time sensors and support back-end data-driven decision support systems, the Dynamic Data-Driven Applications Systems (DDDAS) paradigm becomes a promising approach to make the system smarter by providing online model learning and multi-time scale analytics as part of the decision support system that is used in the DDDAS feedback loop. In this paper, we describe a system in use in Nashville and illustrate the analytic methods developed by our team. These methods use both historical as well as real-time streaming data for online bus arrival prediction. The historical data is used to build classifiers that enable us to create expected performance models as well as identify anomalies. These classifiers can be used to provide schedule adjustment feedback to the metro transit authority. We also show how these analytics services can be packaged into modular, distributed and resilient micro-services that can deployed on both cloud back ends as well as edge computing resources.

1 Introduction

Emerging trends and challenges. Public transit ridership in the United States increased by 37% from 1995-2015, which is roughly twice as much as the country's population growth (21%) in the same years [14]. In 2013 alone, there were 10.7 billion trips taken on U.S. public transportation [13]. Meanwhile, people in the U.S. have been reducing the use of personal vehicles [12]. Public transportation has become an essential part of communities and cities.

Bus services, which is one of the most important segments of public transportation, are vulnerable to delays and congestion due to traffic congestion, weather conditions, special events, etc. Travel and arrival time variation was found to have a substantial impact on commuter satisfaction [16]. Moreover, people's tolerance to errors in bus time predictions is quite low [25]. Providing real-time bus schedules reduces this uncertainty and improves passenger experience and increases ridership. A direct benefit of increased ridership on public transport is the reduced use of personal vehicles and hence reduction in both traffic congestion and greenhouse emissions.

Recently, transit agencies have been integrating real-time sensors into public transit systems. A number of technological systems have been developed by academic researchers and commercial companies to utilize this real-time data. For instance, AVLs (Automatic Vehicle Location) and APCs (Automatic Passenger Counter) can provide real-time data such as vehicle travel time, arrival and departure time, and passenger boarding counts. This data can be used for at-stop displays [23], bus time prediction [17; 48; 15], schedule planning optimization [30; 22], real-time control strategies [24; 39], etc.

However, these sensors have some problems. Accurate real-time bus arrival and departure data that many prediction systems use is not always available. In Nashville, for example, only special bus stops called timepoints are equipped with sensor devices that record exact times. There are over 2,700 bus stops all over the city but only 573 timepoints. In addition, the timepoint dataset is not real-time. It is available at the end of each month when the Nashville Metropolitan Transit Authority (MTA) summarizes and analyzes the historical data. APCs can help provide accurate timing of when a bus stops at a transit stop, which can be used in analysis. On the contrary, AVLs do not provide that. Many transit systems, including the city of Nashville, do not have APCs on buses and use automatic vehicle location (AVL) data

Fangzhou Sun, Abhishek Dubey, Jules White, Aniruddha Gokhale
Institute for Software-Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37212, USA
E-mail: {fangzhou.sun, abhishek.dubey, jules.white, a.gokhale}@vanderbilt.edu

to estimate the arrival and departure time at bus stops and use the estimated data for bus delay prediction in real-time. The issue with this approach is that the lack of quality data results in worse predictive analytic performance. Even for systems with APCs, real-time sensor systems can have many problems in the real world [2; 1], due to reasons, such as low networking bandwidth and delays in uploads. As a result, often GPS position data is noisy.

A typical mechanism for handling noise is to normalize the data. However, normalization requires large data sets, often clustered around transit routes. This is helpful because the transit data of preceding buses may be used to create the models for the current trip on that route. However, if a city does not have high frequency operations across its routes, then such data is not available.

Solution Approach and Contributions. To address the lack of quality data for transit data analytics, yet make effective predictions for bus arrivals, we surmise that the Dynamic Data Driven Applications Systems (DDDAS) paradigm [21] holds promise as a solution approach. In DDDAS, both real-time and/or historical data is used to learn the model of the system that must be controlled, and subsequently a decision support system uses these learned models to make informed decisions and control the system in a feedback loop. This is the approach we utilize in this paper. It integrates historical and streaming real-time bus location data from multiple routes for short-term delay prediction as well as long-term delay pattern analytics. We also use the data feedback loop to provide results to city planners and end users.

This paper significantly extends our prior work on Transit-Hub [42; 38] and provides the following contributions to the study of real-time and predictive analytics for public transportation using DDDAS principles:

- We present a better short-term delay prediction model that combines clustering analysis and Kalman filters and uses real-time data from shared route segments.
- We show the efficacy of our short-term delay prediction model. When predicting the travel time delay of segments 15 minutes ahead of scheduled time, our model reduced the root-mean-square deviation (RMSD) by about 30% to 65% compared with a SVM-Kalman model [15]. The SVM-Kalman model that we used for comparison is a dynamic prediction model that combines SVM and Kalman filters, two of the most widely used models in bus delay prediction [45; 17; 47; 19].
- We provide an algorithm that generates shared bus route segment networks from standard General Transit Feed Specification (GTFS) datasets.
- We illustrate how the analytical algorithms can be packaged into independently deployable and self-contained micro-services.
- We describe how the system's data feedback loop works to provide decision support to city planners by assisting Metro Transportation Authority (MTA) in identifying real-time outliers and optimizing bus timetables to improve bus services and availability.

Paper organization. The remainder of this paper is organized as follows: Section 2 compares the enhanced Transit-Hub system with related work, specifically how we differentiate from and improve on a SVM-Kalman model that also used bus data from multiple routes; Section 3 outlines the key challenges faced in realizing a DDDAS-enabled system for accurate prediction of bus schedules; Section 4 describes the integrated data sources and potential feedback mechanism to MTA; Section 5 describes how we construct the bus delay models and integrate real-time bus data to predict arrival delay in real-time; Section 6 presents the system deployment; Section 7 describes the performance evaluation of travel time delay in route segments and arrival time delay at bus stops; and finally Section 8 presents concluding remarks and future work.

2 Related Work

This section compares Transit-Hub with related work on transit data analysis using different models. In the end, we explain the differences between Transit-Hub models and a SVM-Kalman model that also used shared route segment data.

2.1 Statistical Models

The basic average models directly use the average delay from historical data as the estimated delay for future and are often constructed for performance comparison purposes. For example, Jeong et al. [27] developed a basic average model and found that the basic average model was outperformed by regression models and artificial neural network (ANN) models for bus arrival time prediction. The reason is that the basic average models only use historical data and perform simple average analysis, the model does not reflect real-time conditions and is limited by the consistency of route delay patterns.

Many researchers have conducted studies that utilize both historical and real-time bus data. Weigang et al. [44] presented a model to estimate bus arrival time at bus stops using the real-time GTFS data. Their model contains two sub algorithms to determine the bus speed using the historical average speed and the real-time speed information from GPS. Their main algorithm utilizes the calculated real-time speed to predict the arrival time. Sun et al. [40] proposed a prediction algorithm that combines real-time GPS data and average travel speeds of route segments.

Regression models are also used to explain the impact of variables for delay prediction. Since the variables in transit systems are correlated [20], regression models are typically limited to delay prediction. Patnaik et al. [32] presented a set of regression models that predict bus travel times on a route segment. The data they used is real-world data (number of passengers boarding, stops, dwell time and weather) collected by Automatic Passenger Counters (APC) installed on buses. They also found that weather did not have a significant effect on the prediction.

2.2 Kalman Filter Models

Kalman filters have been used widely for bus delay prediction because of their ability to filter noise and continuously estimate and update actual states from observed real-time data. Chien et al. [19] presented a dynamic travel time prediction model that used real-time and historical data collected on the New York State Thruway (NYST). Shalaby et al. [37] proposed a bus delay prediction model based on two Kalman filter algorithms: one for estimating the running time and another for estimating the dwell time at bus stops. Yang et al. [46] developed a discrete-time Kalman filter model to predict travel time using collected real-time Global Positioning System (GPS) data. Bai et al. [15] proposed a dynamic travel time prediction model that employed support vector machines to provide a base time estimate and a Kalman filter to adjust the prediction using the most recent bus trips on multiple routes.

2.3 Machine Learning Models

Artificial Neural Network (ANN) [18; 27; 47] and support vector machine (SVM) [45; 17; 47; 15] are two of the most popularly used machine learning techniques in bus delay prediction. For example, Jeong et al. [26] developed an ANN model for bus arrival time prediction using Automatic Vehicle Location (AVL) data. Mazloumi et al. [29] used real-time traffic flow data to develop ANN models to predict bus travel times. Yu et al [47] proposed a machine learning model that used bus running times of multiple routes for predicting arrival times of each bus route and proposed bus arrival time prediction models that include Support Vector Machine (SVM), Artificial Neural Network (ANN), k-nearest neighbors algorithm (k-NN) and linear regression (LR).

2.4 Comparison with Our Work

Prior work emphasized long-term and short-term transit data analysis and prediction. However, most of them, as mentioned above, focused on a single route and few noticed that many bus routes share segments with other routes. In 2011, Yu et al. [47] recognized that the data from multiple routes can help to improve the delay prediction. In 2015, Bai et al. [15] proposed a dynamic travel time prediction model that combines SVM and Kalman filter using multiple bus routes data. However, when solving the shared-segment prediction problem, they only used the actual travel time of preceding buses and did not consider the scheduled time difference of separate bus routes. Also, their model included the data of all recent preceding buses, which may contain outliers that should be excluded. Transit-Hub extends these concepts, presents a solution to generate shared route segment network (explained later in Section 5.2.1) using standard static GTFS dataset, and provides transit data predictive analysis at multiple timescales. The benefit is that the analysis results can be used to provide schedule adjustment feedback to MTA, and real-time delay prediction to commuters.

3 Building Multi-timescale Analytical Services for Public Transit

In this section, we present the key problems associated with building multi-timescale analytics models for public transportation systems.

3.1 Problem 1: Integrating and Managing Heterogeneous Data from Multiple Sources

Transportation agencies are employing advanced technologies, such as automated vehicle locators (AVL) and automated passenger counters (APC) to monitor and manage bus services to improve service quality. However, the data collected from multiple data sources may require significant effort to be integrated in order to learn a model for the following reasons: (i) Data are collected at different sampling rates: systems such as AVL and APC have different hardware specifications. Data from different sources need to be sampled before being used by the system; (ii) Data may be missing, duplicated or faulty: these issues need to be detected and handled differently before conducting the data analytics.

Furthermore, the scale of the transit system brings its own challenges and requires efficient and reliable data storage management for the following reasons: (i) Data is large-scale: the real-time transit data, for instance, is accumulated at the scale of several gigabytes per day currently. If Nashville MTA expands its services and updates the devices for faster data rates, more data will be generated and may require more sophisticated management; (ii) Data replication is also required since the system is accessible by the public and needs to be fault-tolerant and reliable.

We address these challenges in Section 4 by describing the heterogeneous data sources and how we integrate, store, and prepare the data for use.

3.2 Problem 2: Utilizing Real-time Bus Data for Multiple Routes that Sharing Similar Segments

Delay prediction models rely on training data. The prediction accuracy depends greatly on the quality of training data. However, the data quality varies for the following reasons: (i) The bus timing is vulnerable to various conditions such as accidents, congestion, road constructions, weather conditions, etc. Therefore, the bus travel time of the preceding bus on the same route may just be an outlier and not reflect the future delay trends; (ii) In mid-sized cities there are limited public resources to support public transportation compared to large metropolis. For example, route 3 (one of the busiest bus routes in Nashville) has 37 trips on weekdays in “From Downtown” direction [8], while M15-SBS in New York has 144 trips in one direction [6]. The quantity of data available for historical analysis and future time prediction in mid-sized cities is less than those in their larger counterparts, which makes it harder to learn accurate models of the system; and (iii) Software bugs, hardware malfunctions and wireless communication issues may occur occasionally and result in missing or faulty real-time data. For example, during our experiments, often AVL data was not uploaded in proper sequence and often had repetitions. Curating such data becomes a challenge in itself.

We address these concerns in Section 5.2.1 by discussing how our short-term prediction model improves the data quality by dividing all bus routes in the city into shared route segments and utilizing real-time bus data from segments shared by multiple routes.

3.3 Problem 3: Providing Schedule Adjustment Feedback to Metro Transit Authority

Improving existing bus schedules is a critical task for metro transportation authorities such as Nashville MTA. MTA regularly examines the historical bus operation reports and updates its bus schedules. Recently real-time sensors are being installed on buses and MTA can track the bus operation in real-time. However, it is still difficult for them to be aware of the actual bus status. For example, by combining real-time bus location feed and static bus schedule, it is not difficult to tell if a bus has deviated from its schedule or not. However, the capability to differentiate a delay event from a normal delay that fits historical delay patterns and thereby identify outliers that need to be further investigated is still lacking at present.

We present our solution to this challenge in Section 4.3 to by designing a data feedback loop for metro transportation authority that tracks the real-time status of the bus operating using analytics result from both historical as well real-time prediction models.

3.4 Problem 4: Building and Deploying the System with High Availability and Scalability

Traditional applications are often built in a monolithic style where all logic for handling requests runs in a single service process. Even though the monolithic architecture is easy to develop, deploy and it is also easy to scale if a load balancer is used, when the scale of the application increases, it will become too large and complex for developers to understand, improve and conduct continuous deployment [34]. Also, the reliability of the monolithic application will be a problem because a break down in one component has the potential to impact the entire application [36].

To improve the scalability and availability of the system, we adopted a microservice architecture, which is a modular architectural pattern for building and deployment [31; 43]. The microservice architecture is well-suited for cloud environments and has many advantages over traditional architectures: (1) Smaller modules are easier to develop and therefore improve the productivity of developers, (2) Services can be developed and deployed independently, (3) The source of faults is more apparent.

However, the microservice pattern is not perfect. It has some unique drawbacks including (1) it is not easy to partition an existing large-scale system into microservices, (2) additional inter-microservice communication mechanism is needed, (3) memory consumption may increase especially if the microservices do not share the same environment. In our current implementation, the microservices are deployed in a single environment to avoid this problem.

Section 6 explains how we addressed this challenge by using a microservices architecture to develop and deploy the back-end services.

4 Data Management and Feedback

In this section, we first present the heterogeneous data sources that the system is using and then describe how we integrate and manage the collected data to address the issues raised in Section 3.1.

4.1 Data Sources

We have been collaborating with the Nashville Metropolitan Transit Authority (MTA) for accessing the static

Bus Schedules	
Format	Static GTFS
Source	Nashville MTA
Update	Every public release
Total Size	193 MB (Version: Mar. 9 2016)

Real-time Transit	
Format	Real-time GTFS
Source	Nashville MTA
Update	Every minute
Total Size	278 GB

Time Points	
Format	Excel
Source	Nashville MTA
Update	Every month
Monthly Size	300,000 entries/month

Table 1: Realtime and Static Datasets Collected and Stored in the System.

and real-time transit data all across the Nashville city. The data sources that we are collecting are as follows (Table 1).

– *Static GTFS data sets:* Static bus schedules and associated geographic information in the General Transit Feed Specification (GTFS) [5] are collected. The data sets include routes, trips, stops, stop times and physical layout.
– *Real-time GTFS data feed:* Real-time transit fleet feed in GTFS real-time [4] format that contains three types of data: service alerts, trip updates and vehicle positions. The data source of the feed includes streaming Automatic Vehicle Location (AVL) data on operating buses.
– *Time point data sets:* Time point Datasets are the historical bus data at time points, including route ID, trip ID, drive ID, actual departure and arrival time, etc. This data is not available in real-time and is only made available at the end of the month.
– *Crowd-sourced Data Feed:* Crowd-sourced data feed is collected anonymously from the Transit-Hub mobile app. Anonymous data generated by users is updated to the server when a user uses the app for route planning and navigation. It should be noted that this data set is not being used in the system described in the paper, however, we will exploit the integration of user-supplied data for closing the loop to the users in the future release.

4.2 Data Management

Data Collection. We have to handle data from each source differently as they have different update rates and formats. For example, (i) Bus schedule data (static GTFS) is updated only when MTA modifies its bus routes or schedules; (ii) Historical time point data set is collected by MTA at the end of the month and is then manually transferred and imported into our MongoDB database. On an average, we collect approximately three hundred thousand entries each month; and (iii) For the real-time transit data, our back-end server requests the data from these real-time feeds every minute and stores the responses in the database (see Table 1).

Data Cleaning. Data cleaning is a crucial step for data pre-processing to handle the following issues:

– *Duplicated data.* Detecting and eliminating duplicated data is one of the major tasks for data cleaning. We compare and remove data with the same time stamps and key-value pairs.
– *Data with logistic errors.* This type of data exists mainly in the real-time bus location data. To deal with it, for example, we remove the records where a bus' distance from a stop changes too fast, or if it moves in the wrong direction. This is done using some custom filters created by us.
– *Missing data.* This can happen for various reasons, which are: (a) operational disruptions due to service alerts, (b) hardware failures, or (c) data transmission issues. The missing data is filled in using linear interpolation on the sampled data.

Data Storage. The large scale of the historical and real-time transit data that are accumulated over time requires efficient storage and management methods. Also, the stored data must be accessible to multiple clients in the system at the same time. To meet this scale requirement, we employ JSON as the data structure and MongoDB [7] for data storage. MongoDB is a distributed NoSQL database that can efficiently store and query data on the scale of terabytes.

4.3 Opportunities for Closing the DDDAS Loop

This section shows how data feedback in the transportation Decision Support System can be used to help metro transportation authority (MTA) to identify real-time outliers and perform long-term delay optimization to improve bus services and availability.

Figure 1 illustrates the data feedback cycle. We utilize the multi-source data from Nashville MTA to conduct real-time and long-term data analytics, and the results can be sent back to them as feedback in different ways:

– *Metro Transportation Authority (MTA).* By doing long-term bus data analysis, our models can find the delay patterns that are associated with seasons, day of the week, and time of day. This feedback can be used by MTA to identify bottlenecks within routes and adjust the bus timetable or route layout accordingly. Also, by tracking the real-time bus data and comparing it with the historical delay patterns, we are able to find the outlier trips that deviate from the normal ones, which will be used to inform MTA to investigate and avoid these in the future.

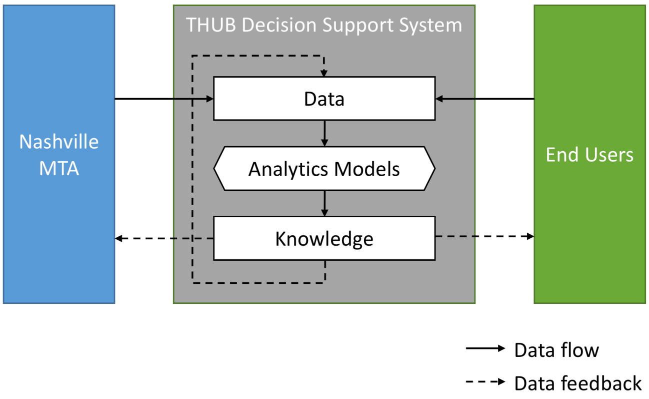

Fig. 1: Proposed DDDAS Loop in Transit-Hub Transportation Decision Support System between MTA, Transit-Hub and end users.

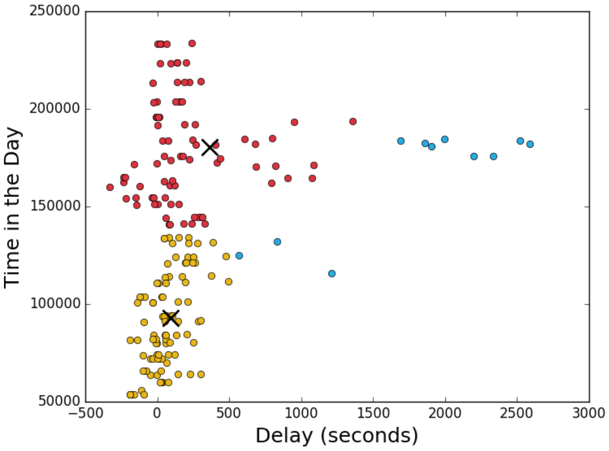

Fig. 2: Cluster historical delay data according to the delay and time in the day at time point “HRWB” on route 3. The figure shows that there are two active delay patterns, one before and one after 2 PM. The blue dots are outliers identified by analysis in Section 5.1.3

– *End Users.* We are collecting anonymous usage and location data from application users. This data can be used to provide an alternative real-time data source for buses. If a user plans to take a bus that is full of people, the system can send notifications to advise him/her to take some other bus or routes. In addition, it can also help to optimize the bus route network and reduce rider walking distances as it shows the origins of users to the bus stops and helps MTA to identify areas with low/high transit service availability.

5 Model Construction

In this section, we present how we construct the long-term delay model, short-term travel-time model and arrival delay prediction model. In particular, we solve Problem 2 described in Section 3.2 by creating a shared route segment network and utilize real-time data from multiple routes.

5.1 Building Model for Analyzing Long-term Delay Patterns

The section describes a long-term analytics model that constructs historical bus delay patterns at time points. In this model, clustering methods are applied to historical arrival delay and travel delay data.

5.1.1 Clustering Analysis

For each weekday, K-means clustering algorithm [28] is used to obtain the cluster of the delay data in accordance with the delay and time of the day by minimizing the within-cluster sum of squares (WCSS).

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{1}$$

where μ_i denotes the mean of all points in the cluster S_i.

Silhouete analysis [35] is an approach to measure how close each point is to others within one cluster.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{2}$$

where for each data point i in the cluster, a_i is the average distance between i and the rest of data points in the same cluster, b_i is the smallest average distance between data point i and every other cluster, and $s(i)$ is the Silhouete score. We calculate the silhouette scores for 2 to 5 clusters derived from K-means algorithm to find the optimal number of clusters with the lowest silhouette score.

The normal distribution of the clustered data helps to identify the typical delay patterns of previous buses, which can be given to users when they want an estimate for a future time, or if there is no real-time data available. The time point data is imported into the database at the end of each month. Then the data is stored according to weekday. We subsequently generate the clusters and normal distributions for all the route segments in each group. Meanwhile, the clustered data and normal distributions are cached and stored in the database. Thus, when we have to query the model, there is no need to run clustering analysis again.

Example Consider a time point 'HRWB' on route 3 in Nashville. The historical bus arrival delay data we select is for Wednesday, outbound direction, between June 1 2016 and June 30 2016 (for a total of 185 points). Figure 2 displays the delay data for a day during that month. In the figure, there are two obvious groups (yellow and red), one is between 5 AM - 2 PM and the other one is between 2 PM and 12 AM. The two groups reveal that there exist two different delay patterns which happen in the morning and in the afternoon separately. This information can be provided to end users to help them plan trips.

5.1.2 Normality Test and Analysis.

The analytics is based on the assumption that historical delay data has a normal distribution. In order to ensure this, we perform normality test on each cluster that we get in the previous step. We can calculate the confidence interval for long-term delay analysis from the distribution curve.

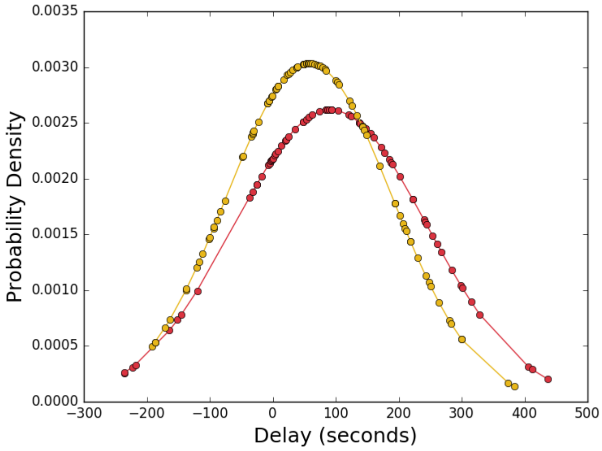

Fig. 3: Normal distribution of the clustered historical delay data at time point "HRWB" on route 3

Example: These are the two normal distributions in Figure 3 that we obtain after performing the normality test on the clusters generated from the data described in the previous example. The cluster for the delay in the afternoon has a higher mean value (92.0 seconds vs. 58.0 seconds) and a wider normal distribution curve, which indicates that buses on route 3 are more likely to be on time in the afternoon. In the afternoon, the 95% confidence interval of delay is between -60.4 seconds to 244.4 seconds while in the morning the 95% confidence interval of delay is between -73.5 seconds to 189.6 seconds (the negative seconds mean the buses are predicted to arrive earlier than scheduled time).

5.1.3 Outlier Analysis.

In order to identify outliers from historical bus data, the first step is to generate the normal distribution for each of the clustered data groups described in the former sections. Since for a normal distribution where μ is the mean value and σ is the standard deviation, 95% of all data is within the confidence interval of [μ-2σ, μ+2σ], we define that the outliers are the historical data with delay greater than μ+2σ or less than μ-2σ in the distribution.

Example: For the dataset mentioned in the previous two examples, there exist some outliers (blue points) in Figure 2. These outliers belong to the two clusters obtained from clustering analysis and are identified by outlier analysis. The outliers mostly emerged during rush hours in the morning and in the evening. One hypothesis is that during rush hours, there are more passengers and more traffic congestion on the route, which will increase the boarding time at stops and travel time on the road. Since our back-end server is monitoring the real-time transit feeds and in the meantime records real-time data, trips that have severe outliers and do not fit in the typical delay pattern can be easily detected and used for further investigation.

5.1.4 Bottleneck Identification

After mean delay patterns of all time points and all route segments are derived, we can then identify the bottlenecks along the routes by using those patterns. This also helps so that actions to optimize the route performance can be taken afterwards.

	Timepoints			
	WE23	WE31	HRWB	WHBG
Morning	116.90	127.71	93.14	443.52
Afternoon	121.03	146.28	114.48	545.49

Table 2: Mean value of the delay data distributions for 4 time points on route 3 in morning and afternoon in June.

There are 4 time points "WE23", "WE31", "HRWB" and "WHBG" on route 3 (traveling away from downtown Nashville). Table 2 shows the findings that the typical arrival delay for "WHBG" is 443.52 seconds in the morning and 545.49 seconds in the afternoon. Considering the fact that the typical arrival delays for "WE23", "WE31" and "HRWB", timepoints before "WHBG", in the morning and afternoon are all below 150 seconds, we can draw the conclusion that the bus stops between "HRWB" and "WHBG" are the bottlenecks for route 3.

5.2 Real-time Data Integration

This section describes a short-term bus arrival delay prediction model that we have developed to address the challenges presented in Section 3. The model integrates real-time bus location data of shared route segments and combines clustering analysis and Kalman filters for delay prediction.

5.2.1 Utilizing Shared Route Segment Data

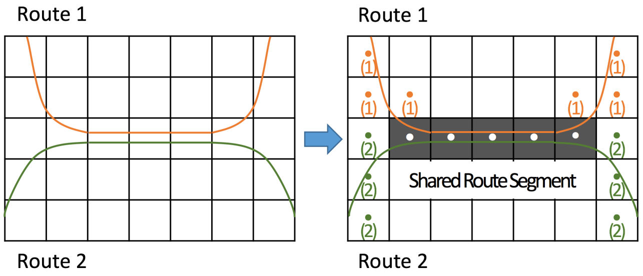

Fig. 4: Finding shared route segments between two bus routes. The segment that contains the three center points is shared by route 1 and route 2.

Problem 2 from Section 3.2 describes the issue that real-time bus data is not always available due to infrequency of buses. To address this challenge, the short-term delay prediction model in Transit-Hub creates a shared bus route segment network, and uses the real-time data from shared route segments for short-term predictive analysis.

Our prior work [42] was based on shared route segments, but at that time we used shared segments that were manually selected and we did not provide a solution to automatically identify shared route segments. In this paper we present an algorithm to create a shared bus route segment network for all the existing routes in the city [41]. Also, the data that the algorithm uses is in standard GTFS format, so the algorithm can be easily applied to other cities that use the same data format.

A Route segment is defined as a maximal part of bus route that is shared by a set of bus routes. In GTFS format, the physical path of bus routes is described using a sequence of coordinate points (in the *shapes.txt* file) on the map. If there are two segments from two bus routes that share the same sequence of coordinate points, then we can assume that the routes share that road segment. The outline of the algorithm to generate the share route segment network is described as below (The key steps are illustrated in Figure 4):

Input: *Static GTFS dataset.* Static bus and associated geographic information are loaded from database.

Output: *Shared route segment network.* Segment layout for each bus route is saved in the database.

Step 1: *Map grid initialization.* The Nashville map is divided into map grids of squares. The length of each square is about 8.97 meters, so each grid cell covers about 80.51 square meters on the map.

Step 2: *Route path re-sampling and smoothing.* The sequences of points in all bus routes are re-sampled to the centers of grid cells if the point is covered by the cell. Also, if the distance between adjacent points in the sequences is larger than the width of a grid cell, points will be interpolated to fill the cells that are missing points. The re-sampled points of each route are cached in the database for determining the shared route segments in the later step. As shown in Figure 4, the paths of route 1 and 2 are re-sampled to the center points of grid cells.

Step 3: *Calculating segments for bus routes.* Each cell is tagged by every route that uses that cell. If a cell contains tags from multiple routes then it becomes part of a new shared segment. For example, the three-point segments in Figure 4 are shared by route 1 and 2, so this segment is marked as a shared route segment. New segments are checked to make sure no duplicated segments are generated.

Step 4: *Segment length limitation.* Any segment that has a length that is greater than 1 mile is divided into smaller segments. because our model is based on the assumption that the travel delay within each segment is equally distributed, and hence the division of larger segments into smaller ones will satisfy this assumption and reduce prediction error.

Using Nashville's static GTFS (version of March 9, 2016), we generated a shared route segment network shown in Figure 5. The 57 bus routes in Nashville city were divided into 5139 segments. The lines in different colors show different route segments. Since the static bus schedules are updated regularly by MTA, the shared route segment network should be updated when new schedules are released.

There are many benefits to using real-time data of shared route segments, such as: (1) Utilizing the real-time data from other routes can greatly increase the volume of data that are available for short-term delay prediction analysis. For example, the route 3 in Nashville from White Bridge to Downtown has a schedule interval of 40 minutes at holiday and weekends. Only using route 3 data means the most recent data is at least 40 minutes old, which is not recent enough to predict the currently delay on route 3. (2) The length of each segment in the network can be controlled by the one-mile limitation mentioned in the last step of the algorithm. Since the delay pattern varies along a bus route, segments with longer length are divided by the algorithm to produce more accurate analytics results. (3) By creating a shared route segment model, the divide and conquer design pattern is used. Individual and self-

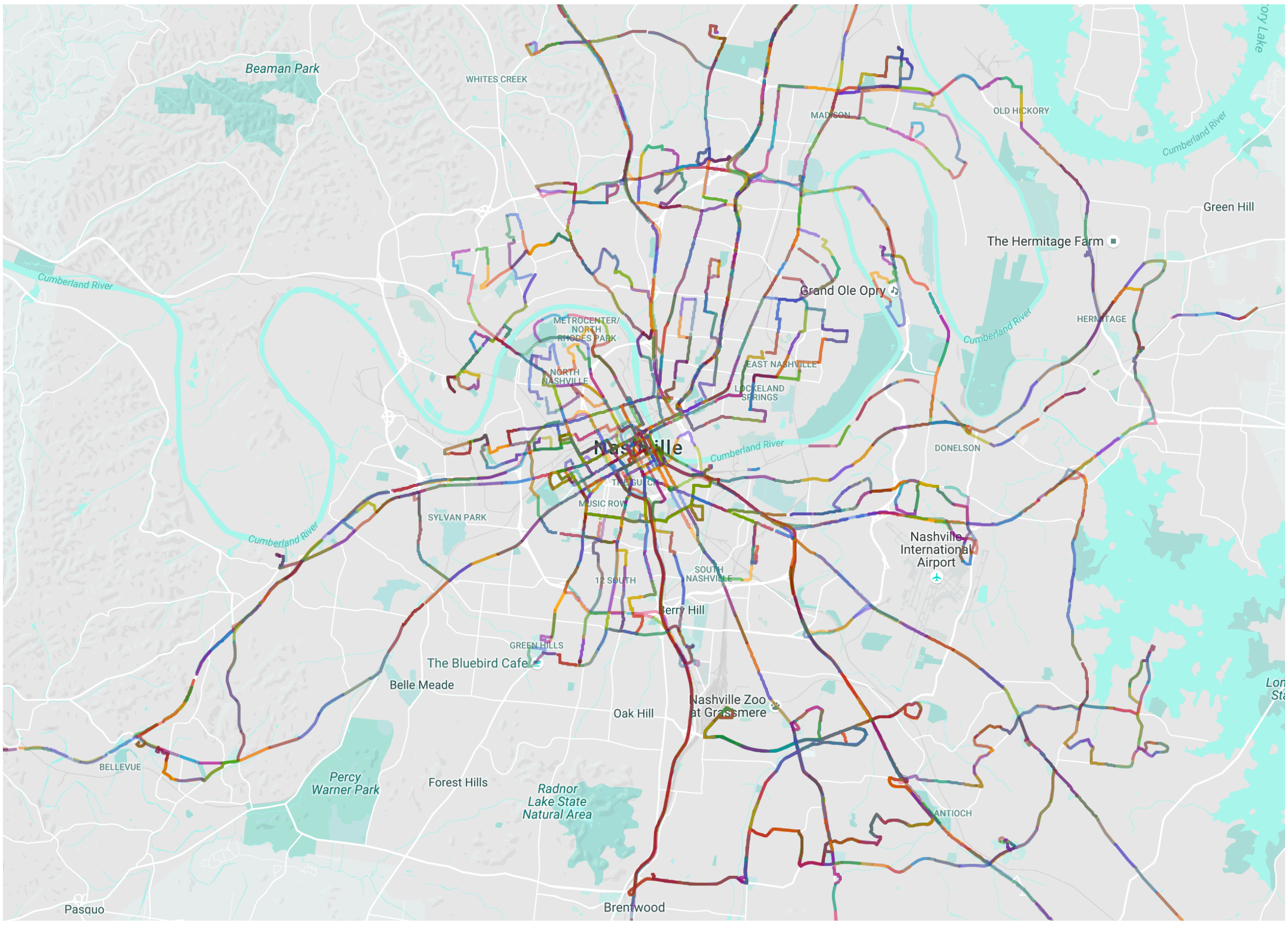

Fig. 5: Generated shared bus route segment network in Nashville. The lines with different colors represent the 5139 shared route segments in all 57 bus routes in the network. The length of the segments are limited to less than 1 mile.

maintained microservice model can then run for each of the segments concurrently.

5.2.2 Estimating the Arrival Time at Bus Stops

Since the actual arrival time at bus stops are not included in the real-time GTFS feed in Nashville, we integrate the real-time bus location data and the static bus stop locations to estimate the arrival time of buses.

From the real-time bus location feed, we can get the bus location and timestamps in the following array format: $[(t_1, d_1), ..., (t_k, d_k), ...]$. Because the update rate of the original data varies from seconds to minutes, we first aggregate the collected data into 1-minute average data using sliding time windows. Then, we assume that bus speed is approximately the average of the two adjacent data points and apply the following equation to calculate the bus arrival time at stops:

$$t_{stop} = t_{k-1} + (t_k - t_{k-1})\frac{d_{stop} - d_{k-1}}{d_k - d_{k-1}} \tag{3}$$

where t_{stop} denotes the estimated arrival time, d_k is the bus's distance from the current location to the first bus stop of the route along the route path at time t_k. Also, $d_{k-1} <= d_{stop} < d_k$.

5.2.3 Updating the Travel Delay Prediction Using K-means Algorithm and Smoothing Filter

Excluding the outliers. If the travel time of a preceding bus differs greatly from other preceding buses, we consider this point an outlier and exclude it from the model computation.

To identify the outliers from the data, we employ K-means algorithm to cluster the preceding bus data according to travel time and time in the day. The Silhouette analysis that was introduced in equation 2 is also used here to find the optimal number of clusters. We choose the cluster whose time of day is closest to the current time. The data points from that cluster are smoothened through the filter described in the next section and used as an estimate for the current travel time on that segment.

Smoothing the preceding bus data. By comparing the travel time of preceding buses and the scheduled travel time within the route segment, we compute the travel delay of the preceding buses in the segment. The travel delay data is then through a filter to eliminate noise and predict the segment's current travel delay. The state transition equation is:

$$x_k = x_{k-1} + \omega_{k-1} \tag{4}$$

where the state variable x_k denotes the time step for which the travel delay needs to be predicted, ω_k denotes the zero mean normal distribution noise with covariance Q_k.

The observation equation used is:

$$z_k = x_k + \nu_k \tag{5}$$

where variable z_k represents the observation of delay at time step k. ν_k represents the zero mean Gaussian distribution observation noise with covariance R_k. ω_k and ν_k are assumed to be independent.

5.2.4 Example

In this section we use an example to explain the workflow of Transit-Hub multi-timescale analysis services. Figure 6 illustrate a common scenario where a bus b_1 is running along a bus route r_1 and the system needs to predict on request, the expected delay for a bus at stop s_i:

1. *Creating shared route segment network.* From the figure we can see that routes r_1 and r_2 are divided into 5 segments: seg_1, seg_2, seg_3, seg_4, seg_5. The segment seg_2 is shared by the routes.
2. *Getting preceding buses using static bus schedules.* From the static bus schedules we find that there are many buses (b_2, b_3, etc.) from route r_1 and r_2 that have passed through segment seg_3
3. *Estimating travel time of the buses in segments.* Preceding buses' travel time can be estimated using the collected real-time bus location data.
4. *Predict travel delay in segments.* The data from recent buses are clustered by travel time and time in the day. The group of data whose mean value (time in the day) is closed to the current time will be smoothed with a Kalman filter.
5. *Getting arrival delay at bus stop.* The sum of the delays for each segment between the current bus position and the target stop s_n is the model's prediction for arrival.

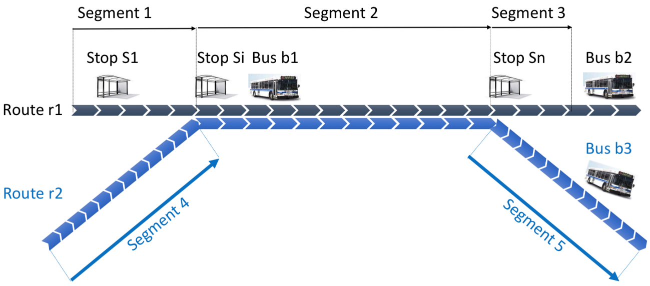

Fig. 6: Use Case: Example of Using Shared Route Segment Data to Predict a Bus's Delay at a Bus Stop

6 Deployed Architecture

In this section, we describe the implementation architecture for the short term online delay prediction service, which addresses problem 4 described in Section 3.4 concerning scalability and availability.

Section 3.4 compared two deployment patterns: traditional monolithic style and microservice style. Microservice deployment is an application architectural pattern where independently deployable and self-contained services can work together, which may be more suitable for complicated web applications [34; 36; 31]. Microservices communicate with each other via lightweight network mechanisms, such as using REpresentational State Transfer (REST) API, message broker, etc. Figure 7 illustrates the overall architecture of the Transit-Hub analytics.

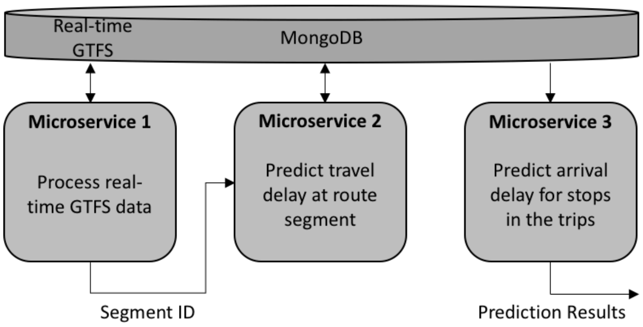
Real-time GTFS
MongoDB
Microservice 1
Process real-time GTFS data
Microservice 2
Predict travel delay at route segment
Microservice 3
Predict arrival delay for stops in the trips
Segment ID
Prediction Results

Fig. 7: Microservice architecture of Transit Hub back-end analytics services

Microservice 1: *Smoothing the real-time GTFS data.* Microservice 1 first cleans the raw real-time GTFS data by removing the duplicate and missing data, and then re-sample it to estimate the bus arrival time on bus routes. This microservice tracks real-time bus location and when a new bus travels through a route segment, it will inform Microservice 2 which updates the travel time delay for this route segment. Microservice 1 is activated by a scheduler every 5 minutes.

Microservice 2: *Predicting arrival delay at segments.* Microservice 2 collects the data processed by Microservice 1 and employs short-term delay prediction (Section 5.2) to update the estimated delay for the route segments. When Microservice 2 receives a prediction update request for a route segment, it wakes up and runs the prediction process to update the travel delay prediction for that route segment.

Microservice 3: *Predicting arrival time at bus stops.* Microservice 3 combines the current delay of all buses and the predicted travel delay for all route segments to produce the arrival delay prediction at all bus stops for all routes. This microservice is activated every minute and stores the prediction results in the database. Note that Microservice 2 runs per route segment whereas service 3 runs to update the arrival time for all routes.

Representational state transfer (REST) API and message broker are two of the popular approaches for providing a communication mechanism between microservices. The REST approach is synchronous by default and uses DNS or a registry for service discovery, and supports load balancing by using software like Ribbon [11]. The message broker is an asynchronous mechanism, which uses queues to manage message queues and can achieve load balancing very easily. Asynchronous message passing is a better choice for microservices because: (1) the individual microservice that sends a message will not be blocked before the other microservice responds; (2) using asynchronous communication can help to reduce unnecessary duplicate computation. For example, in our architecture, microservice 1 is continuously sending the IDs of route segments that need to update prediction to microservice 2. If we find that there are two identical segment IDs in the message queue, then the duplicate can be removed to avoid duplication of work. Based on these considerations, we use RabbitMQ [10], which is a message broker that provides asynchronous messaging.

The microservices are deployed on an OpenStack [9] cloud operating system. We created a *m1.large* nova computing instance for the microservices which has 4 virtual CPUs, $8GB$ RAM and runs Ubuntu 14.04 (LTS). The microservices all together use 10.9% CPU resources and 28% RAM on average. The performance and resource consumption of the microservices will not be affected by user interactions. They run separately and repeatedly in the back end and store analysis results for later use. When an end user sends a prediction request for a route, an independent service in the system will fetch the prediction results from the database and provide the information to the end user.

7 Prediction Performance Evaluation

This section presents experimental results from Transit-Hub's real-time delay prediction model. These results empirically evaluate Transit-Hub's bus travel time delay prediction ability against a SVM-Kalman model [15] using real-time data collected in Nashville. Compared to the SVM-Kalman model, our model takes the scheduled time of preceding buses into consideration, and since we are clustering the data of preceding buses according to time of day and delay, only clusters with an average time of day close to the current time of day will be used. We also evaluate how well our model predicts arrival delay comparing it against real-world data.

7.1 Experiment 1: Evaluating the Travel Time Delay Prediction

The first experiment is designed to evaluate Transit-Hub's ability to predict travel time delay, using its prediction model and comparing against other prediction models using the same real-world data.

Experiment Setup. Routes 3 and 5 are two of the major bus routes in Nashville. As shown in Figure 8, they share the same route segment between time point WES23AWN and time point WES31AWN along West End Avenue. We select this route segment of route 3 and 5 towards WHITE BRIDGE to test our proposed model.

The data used in this experiment is the real-time and static GTFS data for routes 3 and 5 that we collected from Nashville MTA in June 2016. We divide the data into two parts: a training dataset and a validation dataset. The training dataset contains bus data from June 6^{th} to June 12^{nd} and the validation dataset contains data from Jun 13^{rd} to Jun 15^{th}. Our model and the SVM-Kalman are evaluated using the same validation dataset. From our previous paper [42] we learned that only data 120 minutes old or newer is important for real-time delay prediction. Therefore, in this experiment we use the data for buses in the past 2 hours.

Comparing with a SVM-Kalman Model. In order to evaluate the performance of the proposed short-term delay prediction model, we chose and implemented a dynamic SVM-Kalman model that was proposed by Bai, et al. in 2015 [15]. The dynamic model consists of a support vector machines (SVM) model that uses historical data to estimate the current travel time as a baseline prediction, and a Kalman filter model that uses real-time preceding bus data to adjust the base time. The features that they use in the SVM model include: (1) time of the day, (2) road segment ID, (3) weighted average bus travel time of preceding buses, and (4) the travel time of the preceding buses on the same route.

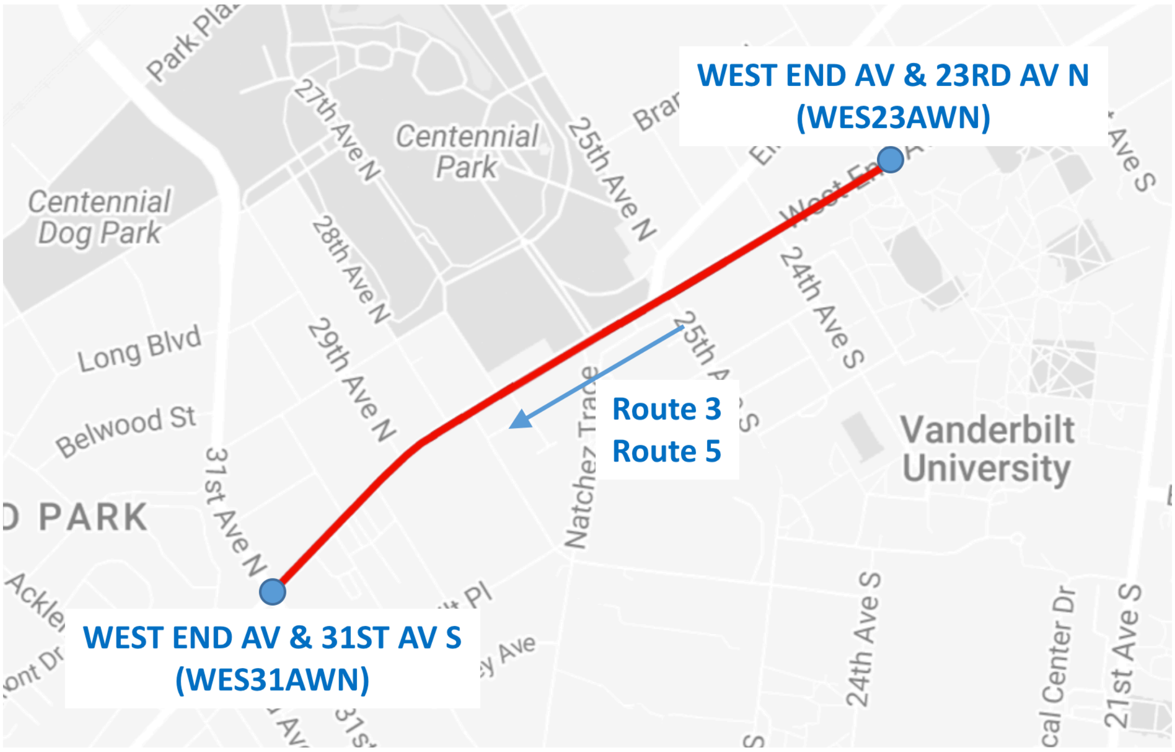

Fig. 8: Studied road segment shared by route 3 and 5

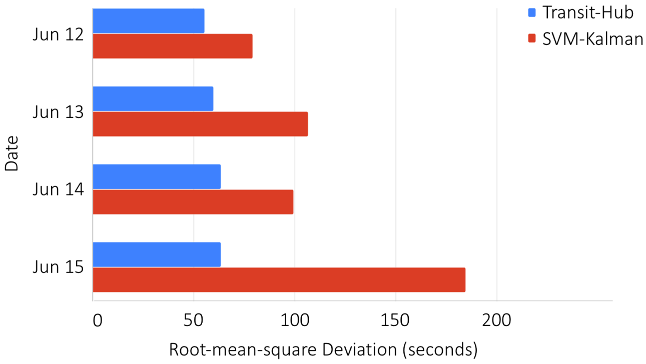

Fig. 9: RMSD of travel time delay prediction for each day when comparing the Transit-Hub model with the SVM Kalman model proposed in 2015. Transit-Hub model outperforms the SVM-Kalman model: (1) RMSD values are smaller (2) it shows less variation on different days.

Results. Figure 9 shows the root-mean-square deviation (RMSD) of the travel time delay prediction results for three days in June. The RMSD of travel time delay is calculated using the following equation:

$$t_{ij}^{act\_tra} = t_j^{act\_arr} - t_i^{act\_dep} \tag{6}$$

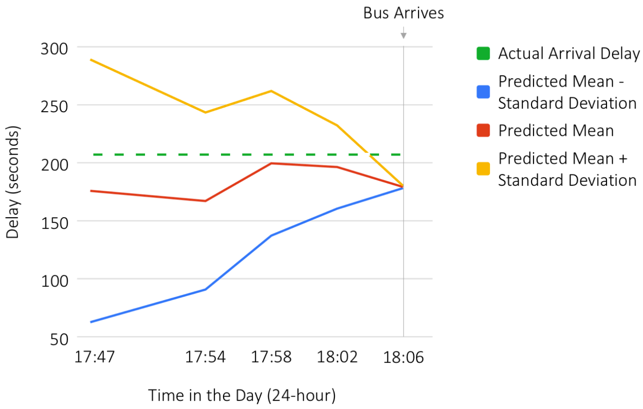

Fig. 10: Arrival time delay prediction for a bus stop of a trip: (1) actual arrival delay, (2) predicted mean value - standard deviation, (3) predicted mean, (4) predicted mean value + standard deviation.

$$RMSD = \sqrt{\frac{\sum_i^n (t_{ij}^{act\_tra} - t_{ij}^{pred\_tra})^2}{n}} \tag{7}$$

where i and j are indexes of the timepoints along the route, and $i < j$. Variable $t_i^{act\_arr}$ and $t_i^{act\_dep}$ represent the actual arrival and departure time at timepoint i, $t_{ij}^{act\_tra}$ and $t_{ij}^{pred\_tra}$ represent the actual and predicted travel time at the segment between timepoint i and j, respectively. n is the number of bus trips in the dataset.

Since the SVM model ignores the differences that exist in the scheduled travel of preceding buses and the model does not exclude outliers, we expect our model to outperform the SVM model from [15]. The experimental results validate our hypothesis. When predicting the travel time delay 15 minutes ahead using collected data from Jun 12 to Jun 15, the RSMD of the our model is about 30% to 65% lower compared to the SVM-Kalman model.

7.2 Experiment 2: Evaluating the Arrival Time Delay Prediction

The second experiment is designed to evaluate the short-term prediction model's performance when the prediction horizon changes. For this experiment, we choose a trip from route 3 on June 14^{th} 2016. The studied segment is shown in Figure 11.

Results. Figure 10 shows the actual delay and the predicted arrival delay with confidence interval as the prediction horizon decreases from 19 minutes to 0 minutes (the time just before the bus arrived).

Since our model integrates the predicted the travel delay in route segments and estimated arrival delay at the most recent bus stop that the bus passed, we expect the predicted confidence interval will become smaller and the error will decrease as the prediction interval reduces, i.e., as we make the prediction closer to the scheduled time of arrival.

This example shows that when predicting 19 minutes before the actual arrival time, the confidence interval is 226.5 seconds and the interval decreases to 2.1 seconds. We notice a 27.8 seconds difference between predicted delay and actual delay when the bus arrives, we attribute this to normal system variance.

8 Conclusion and Future Work

In this paper, we presented research on a DDDAS-enabled smart public transportation decision support system that significantly extends our prior work on Transit-Hub [42] by illustrating and validating the methods developed for long-term and short-term predictive analytics services. Table 3 summarizes the work by presenting the challenges we resolved, the corresponding design principle used, and approaches when developing the system. Our long-term delay analysis service excludes the noise of outliers in the historical dataset and identifies the delay patterns of time points and route segments that are associated with different times of day, day of the week and seasons. The city planners can utilize the feedback data to optimize the bus schedules and improve rider satisfaction. Residents and travelers in cities like Nashville can also benefit from our short-term delay prediction services.

In the future, the work presented in the paper can be extended in the following ways: (1) We want to integrate more data sources into the analysis and prediction models. New data sources, such as traffic flow, weather conditions, special events, can impact public transportation and can be used as new feature vectors to improve the current services. The crowd-sourced data is being collected and the integration of user data will be explored in the future. (2) The services can be deployed further in edge devices using tools like Apache Edgent [3] to reduce the data transmission between edge nodes and a central analytics engine. (3) The system can fit into a smart city platform called Cyber-pHysical Application aRchItecture with Objective-based reconfiguraTion (CHARIOT) [33], which will improve the system's resilience and communication heterogeneity. (4) Storm is a scalable, fast and distributed computation system. In order to scale the Transit-Hub system to serve multiple cities in the future, Storm can be integrated into the system to consume the distributed streaming real-time data feeds, run the multi-timescale analytics and then make the results available to all users.

Acknowledgments

This work is supported by The National Science Foundation under the award numbers CNS-1528799. We acknowledge the support provided by our partners from Nashville Metropolitan Transport Authority.

References

1. (2014) Cota says its real-time bus-tracking system doesn't work. http://www.dispatch.com/content/stories/local/2014/07/23/COTA-says-its-GPS-system-doesnt-work.html, accessed: 2016-09-30
2. (2014) Real-time port authority bus tracking system not always real. http://www.post-gazette.com/news/transportation/2014/10/16/Real-time-Port-Authority-tracking-not-always-real/stories/201410160155, accessed: 2016-09-30
3. (2016) Apache edgent documentation. http://edgent.apache.org/docs/home, accessed: 2016-09-30
4. (2016) General transit feed specification (gtfs) real-time overview. https://developers.google.com/transit/gtfs-realtime/, accessed: 2016-09-18
5. (2016) General transit feed specification (gtfs) static overview. https://developers.google.com/transit/gtfs/, accessed: 2016-09-18
6. (2016) M15 service between east harlem and south ferry. http://web.mta.info/nyct/bus/schedule/manh/m015scur.pdf, accessed: 2016-09-26
7. (2016) The mongodb 3.2 manual. https://docs.mongodb.com/manual/, accessed: 2016-09-25
8. (2016) Nashville mta maps and schedules. http://www.nashvillemta.org/

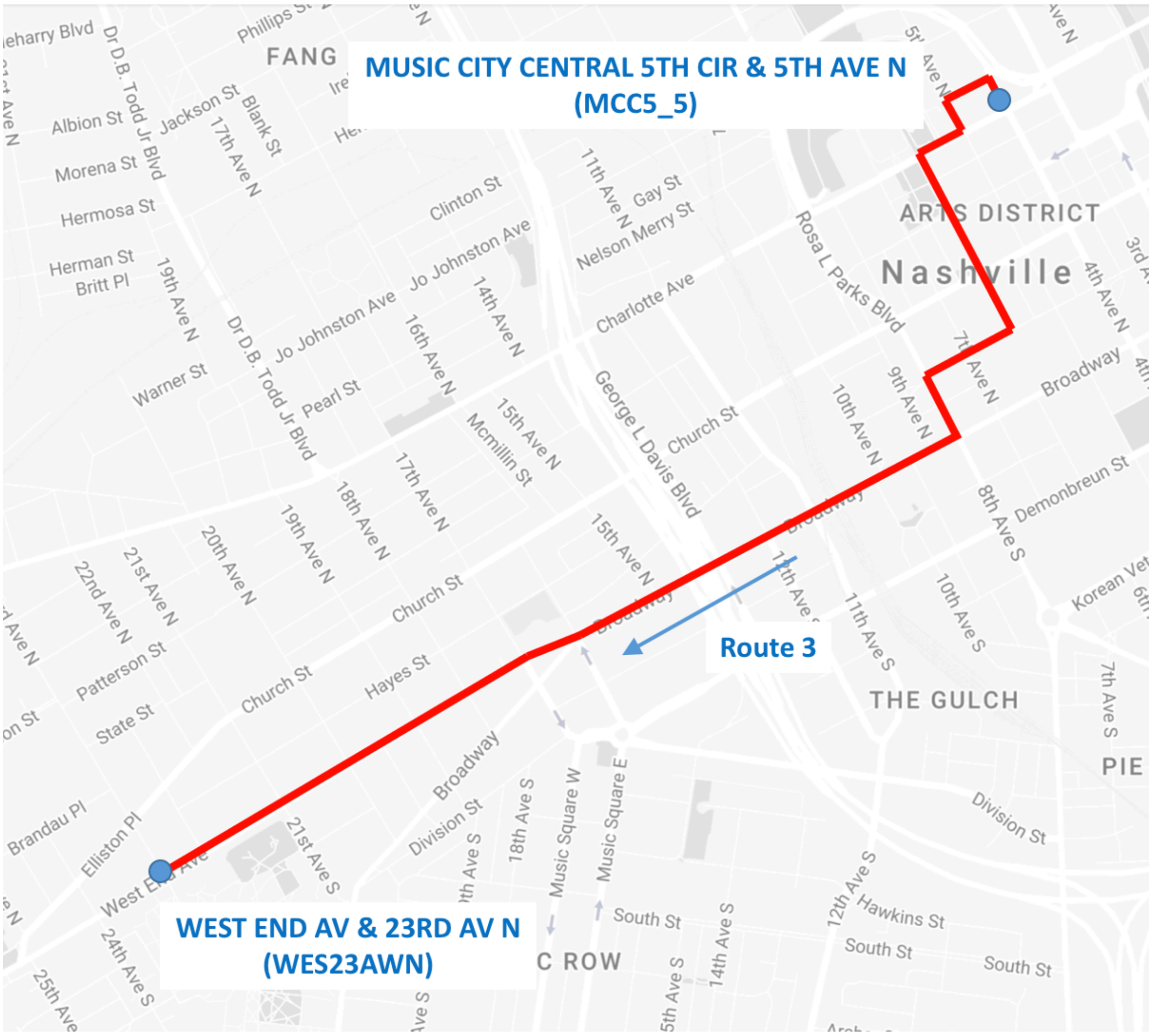

Fig. 11: Studied segment of route 3 that starts from first bus stop (MCC5_5) to the 15^{th} bus stop (WES23AWN)

Nashville-MTA-Maps-and-Schedules.asp, accessed: 2016-09-26

9. (2016) Openstack documentation. http://docs.openstack.org/, accessed: 2016-09-30
10. (2016) Rabbitmq. https://www.rabbitmq.com/, accessed: 2016-09-24
11. (2016) Ribbon, a inter process communication (remote procedure calls) library. https://github.com/Netflix/ribbon, accessed: 2016-09-29
12. Administration FH (2014) Travel monitoring and traffic volume
13. (APTA) APTA (2014) Record 10.7 billion trips taken on u.s. public transportation in 2013
14. (APTA) APTA (2016) Americans took 10.6 billion trips on public transportation in 2015
15. Bai C, Peng ZR, Lu QC, Sun J (2015) Dynamic bus travel time prediction models on road with multiple bus routes. Computational intelligence and neuroscience 2015:63
16. Bates J, Polak J, Jones P, Cook A (2001) The valuation of reliability for personal travel. Transportation Research Part E: Logistics and Transportation Review 37(2):191–229
17. Bin Y, Zhongzhen Y, Baozhen Y (2006) Bus arrival time prediction using support vector machines. Journal of Intelligent Transportation Systems 10(4):151–158
18. Chen M, Liu X, Xia J, Chien SI (2004) A dynamic bus-arrival time prediction model based on apc data. Computer-Aided Civil and Infrastructure Engineering 19(5):364–376
19. Chien SIJ, Kuchipudi CM (2003) Dynamic travel time prediction with real-time and historic data. Journal of transportation engineering 129(6):608–616
20. Chien SIJ, Ding Y, Wei C (2002) Dynamic bus arrival time prediction with artificial neural networks. Journal of Transportation Engineering 128(5):429–438
21. Darema F (2004) Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements. Computational Science-ICCS 2004 pp 662–669
22. Dessouky M, Hall R, Nowroozi A, Mourikas K (1999) Bus dispatching at timed transfer transit stations using bus tracking technology. Transportation Research Part C: Emerging Technologies 7(4):187–208
23. Dziekan K, Kottenhoff K (2007) Dynamic at-stop real-time information displays for public transport: effects on cus-

Challenge	Design Principle	Approach	Section
Learning the historical delay patterns	Long-term analytics model	Clustering analysis, normality test and outlier analysis	Sec 5.1
Accurate bus delay prediction	Better usage of the real-time bus data	Prediction model that combines clustering and Kalman filter	Sec 5.2
Lack of quality real-time data	Integrating shared route segment data	Shared route segment network	Sec 5.2.1
Improve scalability and fault isolation	Separation into independent modules	Microservice architecture	Sec 6
Optimize the bus service	Data feedback loop	Provide the analytics results via feedback loop to MTA	Sec 4.3

Table 3: Summary OF Architectural Decisions in Transit-Hub

tomers. Transportation Research Part A: Policy and Practice 41(6):489–501

24. Fu L, Liu Q, Calamai P (2003) Real-time optimization model for dynamic scheduling of transit operations. Transportation Research Record: Journal of the Transportation Research Board (1857):48–55
25. Gooze A, Watkins K, Borning A (2013) Benefits of real-time transit information and impacts of data accuracy on rider experience. Transportation Research Record: Journal of the Transportation Research Board (2351):95–103
26. Jeong R, Rilett R (2004) Bus arrival time prediction using artificial neural network model. In: Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on, IEEE, pp 988–993
27. Jeong RH (2005) The prediction of bus arrival time using automatic vehicle location systems data. PhD thesis, Texas A&M University
28. Lloyd SP (1982) Least squares quantization in pcm. Information Theory, IEEE Transactions on 28(2):129–137
29. Mazloumi E, Moridpour S, Currie G, Rose G (2011) Exploring the value of traffic flow data in bus travel time prediction. Journal of Transportation Engineering 138(4):436–446
30. Mazloumi E, Mesbah M, Ceder A, Moridpour S, Currie G (2012) Efficient transit schedule design of timing points: a comparison of ant colony and genetic algorithms. Transportation Research Part B: Methodological 46(1):217–234
31. Newman S (2015) Building Microservices. " O'Reilly Media, Inc."
32. Patnaik J, Chien S, Bladikas A (2004) Estimation of bus arrival times using apc data. Journal of public transportation 7(1):1
33. Pradhan SM, Dubey A, Gokhale A, Lehofer M (2015) Chariot: a domain specific language for extensible cyber-physical systems. In: Proceedings of the Workshop on Domain-Specific Modeling, ACM, pp 9–16
34. Rama GM, Patel N (2010) Software modularization operators. In: Software Maintenance (ICSM), 2010 IEEE International Conference on, IEEE, pp 1–10
35. Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20:53–65
36. Sarkar S, Ramachandran S, Kumar GS, Iyengar MK, Rangarajan K, Sivagnanam S (2009) Modularization of a large-scale business application: A case study. IEEE software 26(2):28–35
37. Shalaby A, Farhan A (2003) Bus travel time prediction model for dynamic operations control and passenger information systems. Transportation Research Board 2
38. Shekhar S, Sun F, Dubey A, Gokhale A, Neema H, Lehofer M, Freudberg D (2016) Transit hub. Internet of Things and Data Analytics Handbook pp 597–612
39. Sun A, Hickman M (2005) The real–time stop–skipping problem. Journal of Intelligent Transportation Systems 9(2):91–109
40. Sun D, Luo H, Fu L, Liu W, Liao X, Zhao M (2007) Predicting bus arrival time on the basis of global positioning system data. Transportation Research Record: Journal of the Transportation Research Board (2034):62–72
41. Sun F (2016) Transit hub - shared route segment network generation algorithm. `https://github.com/visor-vu/thub-shared-route-segment-network`
42. Sun F, Pan Y, White J, Dubey A (2016) Real-time and predictive analytics for smart public transportation decision support system. 2016 IEEE International Conference on Smart Computing (SMARTCOMP) pp 1–8, DOI 10.1109/SMARTCOMP.2016.7501714
43. Thönes J (2015) Microservices. IEEE Software 32(1):116–116
44. Weigang L, Koendjbiharie W, de M Juca R, Yamashita Y, MacIver A (2002) Algorithms for estimating bus arrival times using gps data. In: Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on, IEEE, pp 868–873
45. Wu CH, Ho JM, Lee DT (2004) Travel-time prediction with support vector regression. Intelligent Transportation Systems, IEEE Transactions on 5(4):276–281
46. Yang JS (2005) Travel time prediction using the gps test vehicle and kalman filtering techniques. In: Proceedings of the 2005, American Control Conference, 2005., IEEE, pp 2128–2133
47. Yu B, Lam WH, Tam ML (2011) Bus arrival time prediction at bus stop with multiple routes. Transportation Research Part C: Emerging Technologies 19(6):1157–1170
48. Zhang C, Teng J (2013) Bus dwell time estimation and prediction: A study case in shanghai-china. Procedia-Social and Behavioral Sciences 96:1329–1340