MOBI-AID: A Big Data Platform for Real-Time Analysis of On Board Unit Data

Arnau Dillen
Machine Learning Group, Université Libre de Bruxelles
Brussels, Belgium
arnau.dillen@ulb.ac.be

Giovanni Buroni
Machine Learning Group, Université Libre de Bruxelles
Brussels, Belgium
giovanni.buroni@ulb.ac.be

Yann-Aël Le Borgne
Machine Learning Group, Université Libre de Bruxelles
Brussels, Belgium

Karl Determe
Brussels Mobility
Brussels, Belgium

Gianluca Bontempi
Machine Learning Group, Université Libre de Bruxelles
Brussels, Belgium
gbonte@ulb.ac.be

ABSTRACT

Every day large amounts of goods are transported by heavy-goods vehicles over the road network. Being able to monitor and analyse heavy-goods vehicle traffic is essential to define policies able to minimize the impact of negative effects. However, this requires dealing with large amounts of data and often a dense road network, especially in an urban setting. This paper introduces a platform that makes use of state-of-the-art big data technologies to process data pertaining to the positions and properties of heavy-goods vehicles. This platform aims to provide policy-makers and other stakeholders with the tools that allow large-scale analysis of heavy-goods vehicle data in a near real-time fashion. Additionally, the platform allows for forecasting of future traffic conditions based on historical data.

1 INTRODUCTION

Road freight transport is an essential aspect of any country's infrastructure policy due to its economic, environmental and social impact. Among other issues, freight vehicles are responsible for a large part of the congestion on urban road networks (economic impact), pollutant emissions such as carbon dioxide (environmental impact) and physical consequences of pollutant emissions on public health (social impact) [1].

Urban planners and policy-makers therefore demand Intelligent Transportation Systems (ITS) which are able to foresee the mobility behavior and support the definition of appropriate policies [26, 29]. Tools such as accurate traffic forecasting models [30], advanced mobility indicators of freight transport [15] and more general mobility models [3] can assist policy makers in making appropriate decisions.

Traffic on a road network exhibits features which are common to most complex systems: self-organization, emergence of transient space-time patterns based on local and global feedback loops, which makes analysis of these types of data difficult. Due to this, few studies [3, 18, 29, 31] address a complete transportation network including both freeways and urban contexts or limit themselves to offline analysis [15, 25]. One of the main reasons is the scarce availability of data gathered from point detectors or interval detectors and the lack of methods able to tackle the traffic prediction problem at a larger scale [2, 29]. However, thanks to the more ubiquitous availability of new information and communication technologies, more traffic data, especially moving sensors data, are collected and made openly available by both public and private companies, allowing the development of data-intensive approaches for traffic analysis.

In Belgium, traffic data is gathered for heavy-goods vehicles (HGV) by *Bruxelles Mobilité*[1], the public administration responsible for equipment and infrastructure related to mobility issues in the Brussels Capital Region (BCR). They continuously receive data on HGV positions, which is normally used to charge HGVs for kilometers driven on toll roads in Belgium. Every day, an average of 19 Gigabytes of data are therefore accumulated and need to be processed in a timely manner, in order to monitor HGV traffic in Brussels.

Bruxelles Mobilité currently stores this data in a centralized `PostgreSQL` [12] database which is set up to handle geographical data through the `PostGIS` [14] extension (see figure 1). However, this solution is unable to cope with the massive amounts of data that are ingested on a daily basis. While it would be possible to optimize queries and create database indices to minimize the time it takes to retrieve a solution to a query, the main issue with a classical relational database system lies in the constant updates and additions of rows. Even the most performant database on the fastest hardware will result in a bottleneck. Additionally, reading and writing these amounts of data from and to a regular file system is too slow for the amounts of data that are being dealt with.

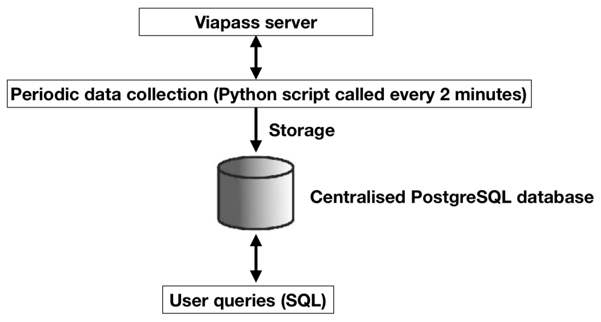

Viapass server
Periodic data collection (Python script called every 2 minutes)
Storage
Centralised PostgreSQL database
User queries (SQL)

Figure 1: Current Brussels mobility architecture for ingesting and storing HGV data.

[1] https://mobilite-mobiliteit.brussels/en

The Machine Learning Group (MLG) of the Université Libre de Bruxelles (ULB) collaborates with *Bruxelles Mobilité* to design a big data architecture able to provide near real-time processing and querying of the incoming data. For example, a query that retrieves the number of trucks on each street is required to make forecasts on future traffic conditions.

An initial version of the architecture was implemented in [5] and has been collecting data on the MLG cluster for some time now. We were able to successfully collect and process large amounts of data thanks to the joint use of an `Apache Hadoop` cluster [21] and `Apache Spark` [32]. However, big data technologies are evolving fast and an appropriate interface to visualize and analyze traffic related data is necessary. Data aggregation is necessary to get a high level view and loading large amounts of data into the interface client is slow and impedes responsiveness of the interface. These are important aspects to take into account when deciding what visualizations the interface should provide and which data should be loaded to the client.

The aim of this research is to be able to perform network-scale analysis and forecasting in near real-time. The presented architecture allows to make real-time forecasts based on incoming data using both well-established [29, 30] and state-of-the-art [2, 18] methods on a network-wide scale. It also enables performing analyses, such as identifying important points of congestion caused by HGV traffic in changing conditions for example, which were previously computed offline, in real-time. Next to the previously mentioned models, there is a fair amount of related work that proposes possible forecasting models which could be candidates for a real-time forecasting model on road networks and their different sections [13, 23, 28]. A large corpus of literature discusses this issue.

The main contributions of this paper are twofold. In a first place it introduces an extension to the big data architecture that was implemented in [5] which enables near real-time processing of the incoming data. Secondly, it proposes a design for a dashboard that enables analyses and visualization of data, which is implemented as a web interface. Together, these make up a platform that provides the tools that are necessary to *Bruxelles Mobilité* to monitor the traffic of HGVs in Brussels and provide insights that should be useful in establishing future policies related to transportation of goods within the BCR. The platform was named the MOBIlity Advanced Indicators Dashboard (MOBI AID), after the project that supports this research. Additionally, the work done in this research could also serve as an example for other cities and potentially whole countries to deploy their own platforms to assist in decision making on policies with regards to road freight transport.

2 METHODS AND IMPLEMENTATION

The data that are gathered concern all HGVs that are currently present in the Belgian territory. At this time we are only interested in HGVs that are present in the Brussels Capital Region, which still concerns thousands of HGVs on a daily basis. To get useful insights from this data, a platform is necessary that can handle such large amounts of data and present forecasts or the results of analyses in a meaningful way. For this purpose, next to the data, two essential components were identified to implement the envisioned platform.

The remainder of this section is structured as follows. Firstly, we will describe the gathered data. Secondly, we discuss the architecture that allows processing and storage of such large amounts of data. Finally, we present a prototype web interface that would be used by policy makers and data scientists to get insights on traffic conditions, from the processed data. This would assist policy makers in making informed decisions regarding urban planning with relation to road infrastructure and freight transport.

2.1 Viapass and On Board Unit (OBU) Data

As of April first of the year 2016, heavy-goods vehicles having a Maximum Authorized Mass (MAM) exceeding 3.5 tonnes must pay a kilometer charge for driving on certain paying toll roads in Belgium. Any vehicle that is not exempt from the toll must have an On Board Unit (OBU) installed. The public organization in charge of supervising the kilometer charge is called Viapass[2]. With the aid of GPS/GNSS satellite technology and mobile data, the OBU records the distance that a HGV travels on Belgian public roads. Mobile wireless technology is used to send the number of kilometers charged to the Viapass data center, after which an invoice is issued to the owner of the vehicle.

Because of their evident value as a mobility indicator, the OBU data are also made available to several mobility agencies, including *Bruxelles Mobilité* which uses this data to analyze freight traffic in the Brussels Capital Region (BCR). The BCR is a separate region from the Flanders region, where it is geographically located, and consists of 19 administrative districts named communes. These districts will be referred as such for the remainder of this article. The models and analyses used in this paper will use OBU data from HGVs within the BCR and its communes.

On average more than nine thousand HGVs are recorded every working day in the larger Brussels Metropolitan Area [4]. Each OBU device sends an update to the server approximately every 30 seconds. An OBU record contains an anonymous identifier, which is reset every day at 4 a.m., the timestamp at which the position was recorded, the GPS coordinates (latitude, longitude), the speed (km/h) and the direction (degrees). Additionally, the data includes vehicle characteristics such as the weight category (MAM), country code and European emission standards classification of the engine (EURO value). This results in an average of 19GB of data incoming on a daily basis and several terabytes of data being generated every year.

2.2 Design of The Big Data Architecture

Handling such large amounts of data requires an architecture that can process the incoming data fast enough and store processed data in an efficient manner. A well-known architecture that meets these requirements is the Lambda architecture [19, 20, 27] which has proven itself in several settings [10, 16] and is used in practice by Twitter among others [17]. An overview of our current implementation of the architecture can be found in figure 2.

With this architecture, three separate layers can be distinguished, which each handle different aspects of the platform. The speed layer takes care of processing incoming data in a timely manner and send the processed data to the serving layer for visualization and analysis. This layer handles the real-time aspect of the platform. The batch layer stores immutable data (i.e. observations) and processes it for later user queries on historical data. The serving layer consists of multiple views that are each used to fulfill a specific type of user queries. For example, data that are stored in a specific format which is used for a specific visualization, or predetermined queries that retrieve data that

[2]https://www.viapass.be/

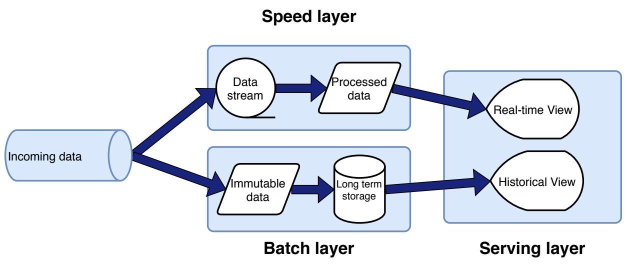


Figure 2: Overview of the Lambda architecture.

are required for a specific analysis. This layer can also merge the information that comes from both speed and batch layers, such as discrepancies between the real-time traffic conditions and the typical case for example. In our current implementation there are two views available. The real-time view provides data that comes from the speed layer directly. The historical view uses the data from the batch layer to query for events and states that have been observed in the past.

The initial implementation of this architecture was deployed on an Apache Hadoop [21] cluster, which is an open-source framework for distributed computing that is widely used for big data processing [7, 24, 27, 32]. The data are collected with a Python script that queries the Viapass servers for new data at a fixed time interval, which is currently set to two minutes. The script loads the data in a GeoPandas [11] DataFrame (a data structure with named columns and index-based rows), which is an extension of the well-known Pandas library for the Python programming language, to support geometric data types and functions. The DataFrame contains all observations that were collected by Viapass since the last data request.

Observations consist of a HGV's current position as a geometry point, which is represented by a given latitude and longitude, together with the unique ID that was assigned to the HGV for that day. Additionally, an observation contains a timestamp of when it was recorded by the OBU and the HGV's characteristics, which were described in section 2.1. Observations are augmented with the current date and time to indicate when the observation was received by our servers. This is done because there is no guarantee that the observations within the retrieved batch will all be for the current day, as it is not uncommon to have observations from previous days come in. As it can not be known when all observations for a day have been received, the system needs to take this into account.

The observations that were retrieved by the script are consequently split by the day on which the observation was recorded and then saved to CSV files on the local file system. The files are stored in a folder that corresponds to the day on which the observations were recorded. These CSV files are used to run simulations of the Lambda architecture by reading batches of data that represent incoming data from Viapass and sending them to the appropriate layers. In real-world scenarios, the incoming data would be sent directly to the appropriate layers of the Lambda architecture.

For the currently deployed implementation of the batch layer, we aggregate the CSV files per day and store them on Hadoop Distributed File System (HDFS) in Parquet format. HDFS allows distributed storage with replication and improved read and write speeds compared to regular file systems. The Parquet file format is a column-oriented format that provides efficient data compression and fast query access. To process the raw CSV files, Apache Spark [9] is used to deduplicate the observations and store them in HDFS as Parquet files. HDFS takes care of distributing file data over the different nodes of the cluster. With this approach, these operations can be processed in parallel and distributed over multiple compute nodes thanks to the integration of Hadoop and Spark. Using Spark we can efficiently run SQL queries and advanced analytics on the data by parallelizing a large part of the computations. An overview of this process is shown in figure 3.

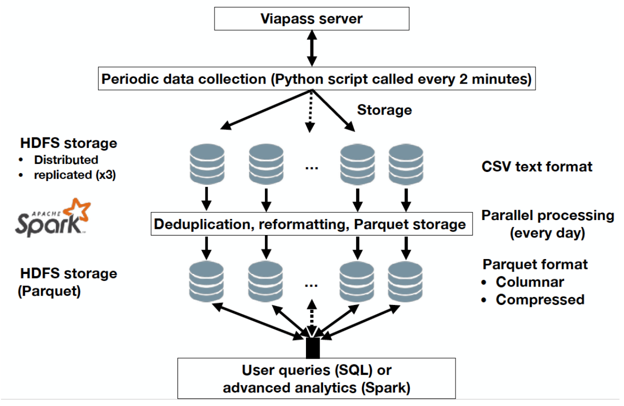
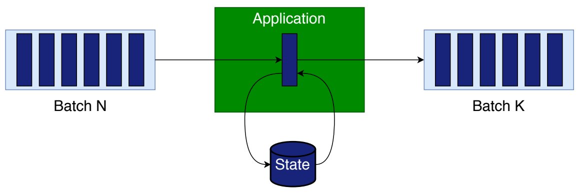

Viapass server
Periodic data collection (Python script called every 2 minutes)
Storage
HDFS storage
• Distributed
• replicated (x3)
CSV text format
Deduplication, reformatting, Parquet storage
Parallel processing (every day)
HDFS storage (Parquet)
Parquet format
• Columnar
• Compressed
User queries (SQL) or advanced analytics (Spark)

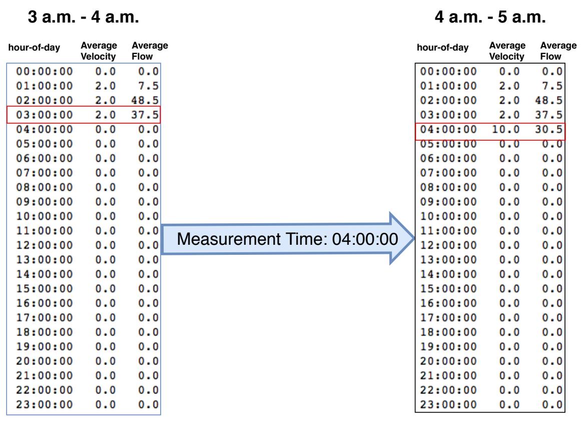
Figure 3: Data retrieval and the batch layer pipeline.

In experiments with an alternative implementation of the batch layer the CSV data is read into a PostGIS database that stores the daily route of a HGV with a given ID. The route is stored as a LineString object (i.e. a sequence of points) constructed from all available observations for a given HGV ID on a given day. In the same database information on Brussels communes is stored, both geographical (e.g. commune boundaries) and non-geographical (e.g. name, population, etc.). Using the geographical operations that are provided by PostGIS, information such as the number of HGVs in a given commune at a given time can efficiently be queried. This alternative batch layer implementation was created, because the current approach lacks data types and functions that are optimized for operations with regards to space and time. Ideally we would like to use both approaches in conjunction, for example by storing raw observations in Parquet format and aggregate these observations over a day to form the route of a truck over that day, to take advantage of the strengths of both approaches.

However, while PostGIS introduces the concept of space with geographic data types and functions, it lacks a concept of both space **and** time taken together without having to introduce additional complexity. PostGIS is not optimized for queries that involve both space and time dimensions taken together. This means that while the sequence of HGV positions can be stored for a certain day, the associated time at which the HGV was at that position can not be stored without introducing additional fields or dimensions and having to make certain assumptions about the data. This results in a loss of speed and data efficiency, which is one of the essential aspects of this platform. For this reason, we are currently investigating a further extension of PostGIS that introduces data types that introduce the concept of a position at a certain time, which is called MobilityDB [33]. This would allow us to perform the necessary queries without being concerned with the underlying representation of the data



(a) State updates according to incoming data stream.



3 a.m. - 4 a.m.

hour-of-day	Average Velocity	Average Flow
00:00:00	0.0	0.0
01:00:00	2.0	7.5
02:00:00	2.0	48.5
03:00:00	2.0	37.5
04:00:00	0.0	0.0
05:00:00	0.0	0.0
06:00:00	0.0	0.0
07:00:00	0.0	0.0
08:00:00	0.0	0.0
09:00:00	0.0	0.0
10:00:00	0.0	0.0
11:00:00	0.0	0.0
12:00:00	0.0	0.0
13:00:00	0.0	0.0
14:00:00	0.0	0.0
15:00:00	0.0	0.0
16:00:00	0.0	0.0
17:00:00	0.0	0.0
18:00:00	0.0	0.0
19:00:00	0.0	0.0
20:00:00	0.0	0.0
21:00:00	0.0	0.0
22:00:00	0.0	0.0
23:00:00	0.0	0.0

4 a.m. - 5 a.m.

hour-of-day	Average Velocity	Average Flow
00:00:00	0.0	0.0
01:00:00	2.0	7.5
02:00:00	2.0	48.5
03:00:00	2.0	37.5
04:00:00	10.0	30.5
05:00:00	0.0	0.0
06:00:00	0.0	0.0
07:00:00	0.0	0.0
08:00:00	0.0	0.0
09:00:00	0.0	0.0
10:00:00	0.0	0.0
11:00:00	0.0	0.0
12:00:00	0.0	0.0
13:00:00	0.0	0.0
14:00:00	0.0	0.0
15:00:00	0.0	0.0
16:00:00	0.0	0.0
17:00:00	0.0	0.0
18:00:00	0.0	0.0
19:00:00	0.0	0.0
20:00:00	0.0	0.0
21:00:00	0.0	0.0
22:00:00	0.0	0.0
23:00:00	0.0	0.0

(b) Transition from the 3 a.m. hour-of-the-day window to the 4 a.m. window when data comes in that was sampled at 4 a.m..

Figure 4: Stateful streaming as implemented in the pipeline.

and optimization of the geographic functions. We are currently in the process of experimenting with the mentioned alternatives to identify the most appropriate approach for the batch layer.

The speed layer of our Lambda architecture implementation uses the `Apache Kafka` [8] streaming platform to store incoming data from queries to Viapass as a continuous stream of data. For the purpose of initial simulations, a `Python` script reads a batch of observations from the stored `CSV` files into a `GeoPandas` `DataFrame`. As a preprocessing step, a different `DataFrame`, which was loaded in memory beforehand, contains the geographic information of a set of Brussels street segments. We used a subset of Brussels streets for testing, however, in practice this would contain all streets in Brussels. By performing a join of the two `DataFrames` with the `within` geographic function provided by `GeoPandas`, we obtain a new `DataFrame` where every observation also contains the internal ID of the street segment the HGV was on at that time. These data are sent to `Kafka` for processing in the next step of the streaming pipeline.

At the receiving end of the data stream, the streaming facilities that are provided by `Spark` are used to process the data, which can directly be integrated with a `Kafka` stream. Incoming data is processed accordingly and used to update the current state of all street segments that are being kept track of. This approach, which is referred to as stateful streaming, is illustrated in figure 4a. The state of a street segment is represented by the average number of HGVs and the average velocity of passing HGVs for every hour-of-the-day of the current day. For every new day at midnight, the state for each street segment is re-initialized to zero values for all properties. Values are subsequently updated continuously with a running mean for the current hour-of-the-day. Values for past hours-of-the-day will contain the mean observed statistics for that day and future values will be zero until the current time falls within the window for that hour-of-the-day. This process is illustrated by figure 4b.

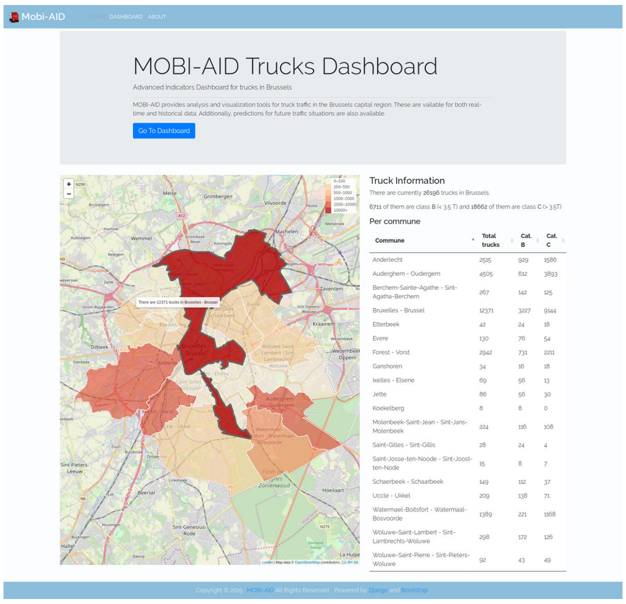
In addition to keeping track of the observed values, forecasts are also made for future hours-of-the-day. Currently, predictions are made using a type of model that is referred to as a persistence model, more specifically, a sliding window persistence model. With this type of model a forecast is based directly on previously observed values for the same day-of-the-week and hour-of-day. In this implementation, the data is divided in one week seasons, meaning that predictions look at the data for the whole week rather than at the currently observed values, to make forecasts. As an example, if the data is hourly and the forecasting target is 9 a.m. on Monday, then given a window size of 1, the observation of last Monday at 9 a.m. will be returned as the predicted forecast. A window of size 2 means returning the average of the observations of the last two Mondays at the same hour and so on for larger window sizes. However, while simple and explainable, this approach is rather naive, as it does not take the current traffic conditions or information that is known in advance, such as a special event that is planned for example, into account. More advanced Machine Learning methods could incorporate this type of additional information for improved forecasting.

The final results are written to a `JSON` file, which is formatted according to the `GeoJSON` [6] specification. In this format, every street segment is described by a `LineString` instance that corresponds to the path of the street segments. In addition to this, each street segment is annotated with HGV counts and average velocities for each hour-of-the-day as properties. The outputted file serves as the real-time view for the considered street segments and can be read by the dashboard for display on a map, or to perform further analysis using the data, such as identifying the busiest streets at the current time for example.

2.3 Implementation of The MOBI-AID Dashboard

To provide an interface that would allow stakeholders to monitor the current traffic situation for HGVs in Brussels or perform historical analyses for future planning, a dashboard interface was implemented. A web interface provides this dashboard and was implemented with the `Django` [22] web framework, additionally making use of the first-party `GeoDjango` extension. Using this extension provides a direct integration with databases such as `PostGIS` and other useful geographical tools. These technologies where chosen for their flexibility, maturity and due to the fact that they required minimal additional learning, given our computer science backgrounds. The fact that these components are also very low level allows us to easily experiment with different alternative approaches.

The web interface is comprised of three pages: `Home`, `Dashboard` and `About`. The `Home` page provides an overview of the available features and displays a map that shows real-time HGV counts for the different communes that compose the Brussels Capital Region. Hovering over a specific commune will show the total number of HGVs that have last been observed in this commune. The HGV counts per commune are also shown in a table beside the map, where they are also divided by weight category. Figure 5 shows a prototype implementation for the home page with the user hovering over the Brussels City commune. The About page provides more detailed information on the web interface and contains the documentation on the dashboard. It also mentions the sources of our funding and the project supporters.


Mobi-AID
HOME
DASHBOARD
ABOUT
Go To Dashboard
0–100
100–500
500–1000
1000–2000
2000–10000
10000+
There are 12371 trucks in Bruxelles - Brussel
Copyright © 2019 MOBI-AID All Rights Reserved · Powered by Django and Bootstrap

MOBI-AID Trucks Dashboard

Advanced Indicators Dashboard for trucks in Brussels

MOBI-AID provides analysis and visualization tools for truck traffic in the Brussels capital region. These are vailable for both real-time and historical data. Additionally, predictions for future traffic situations are also available.

Truck Information

There are currently **26196** trucks in Brussels.

6711 of them are class B (< 3.5 T) and **18662** of them are class C (> 3.5T)

Per commune

Commune	Total trucks	Cat. B	Cat. C
Anderlecht	2515	929	1586
Auderghem - Oudergem	4505	612	3893
Berchem-Sainte-Agathe - Sint-Agatha-Berchem	267	142	125
Bruxelles - Brussel	12371	3227	9144
Etterbeek	42	24	18
Evere	130	76	54
Forest - Vorst	2942	731	2211
Ganshoren	34	16	18
Ixelles - Elsene	69	56	13
Jette	86	56	30
Koekelberg	8	8	0
Molenbeek-Saint-Jean - Sint-Jans-Molenbeek	224	116	108
Saint-Gilles - Sint-Gillis	28	24	4
Saint-Josse-ten-Noode - Sint-Joost-ten-Node	15	8	7
Schaerbeek - Schaarbeek	149	112	37
Uccle - Ukkel	209	138	71
Watermael-Boitsfort - Watermaal-Bosvoorde	1389	221	1168
Woluwe-Saint-Lambert - Sint-Lambrechts-Woluwe	298	172	126
Woluwe-Saint-Pierre - Sint-Pieters-Woluwe	92	43	49

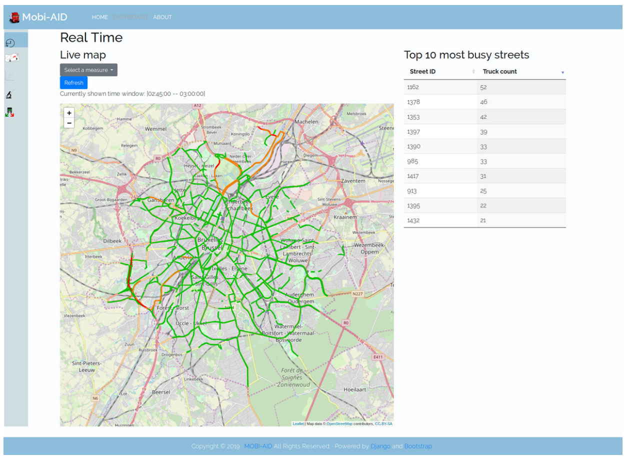
Figure 5: Prototype home page of the web interface.

The Dashboard page provides the core functionality of the web application. This page consists of several tabs which provide a certain type of visualizations or allows for specific analyses to be performed. In it's current implementation, the dashboard consists of the following tabs: `Real-time`, `Maps`, `Charts`, `Analytics` and `Predictions`.

The `Real-time` tab is composed of several panels that display different types of real-time information, which are retrieved from the Lambda architecture's real-time view. In this tab, users can select the type of information they want to see, which will then be displayed on the map. A table next to the map displays a user selected overview of the information that is displayed on the map. For example, the top ten most busy streets can be displayed in this table. Figure 6 shows the current prototype for the `Real-time` tab.

Note that in this figure the time-window for collecting statistics is 15 minutes as opposed to the one hour window that is used for the state of a street. This window corresponds to the interval between consequent updates of the state rather than the hour-of-day window that is being updated in the state. Additionally note that streets in the table are identified by ID's. In practice we would use street names in the final implementation.

The `Maps` tab contains a large map that shows historical data about the observed HGV traffic as selected by the user. We distinguish two distinct ways to look at historical data in this situation. The user can select to either look at the data at a specific time on a specific date, or they can choose to look at data that is typical


Mobi-AID
HOME
DASHBOARD
ABOUT
Select a measure
Refresh
Copyright © 2019 MOBI-AID All Rights Reserved · Powered by Django and Bootstrap

Real Time

Live map

Currently shown time window: [02:45:00 -- 03:00:00]

Top 10 most busy streets

Street ID	Truck count
1162	52
1378	46
1353	42
1397	39
1390	33
985	33
1417	31
913	25
1395	22
1432	21

Figure 6: Work-in-progress Real-time tab of the MOBI-AID dashboard.

for a certain hour-of-the-day on a certain day-of-the-week. The user can also select at which level of aggregation they want to see information displayed on the map. The currently provided levels of aggregation are commune level, street level and at the level of individual HGVs. Individual HGVs can not be shown when looking at the typical traffic situation, as concrete HGV positions evidently vary with time. However, in this case clusters would be shown at locations where HGVs are often present at the chosen hour-of-day and day-of-the-week. Figure 7 shows the work-in-progress Maps tab, without the website header, footer and the tab-selection menu. Note that the selection controls should be separated based on the previously selected type of visualization. These controls would also be shown on the map rather than above, as is currently the case.

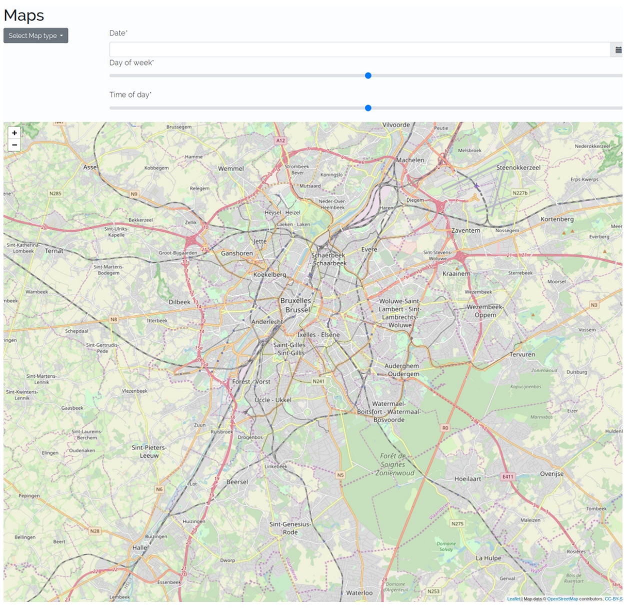


Figure 7: Work-in-progress Maps tab of the dashboard. Without site headers and dashboard tab-selection side menu.

3 EVALUATION OF THE INITIAL PLATFORM

For the MOBI-AID dashboard to provide an optimal user-experience and be a useful contribution to the field of big mobility data, two main aspects are of particular importance. These essential features are adequate performance of the real-time data processing pipeline and the usability of the web interface. To evaluate performance, scalability tests were performed with a simulated stream that is read from the data which is currently being collected from *Bruxelles Mobilité*. The user interface was evaluated through user testing and feedback.

3.1 Experimental setting

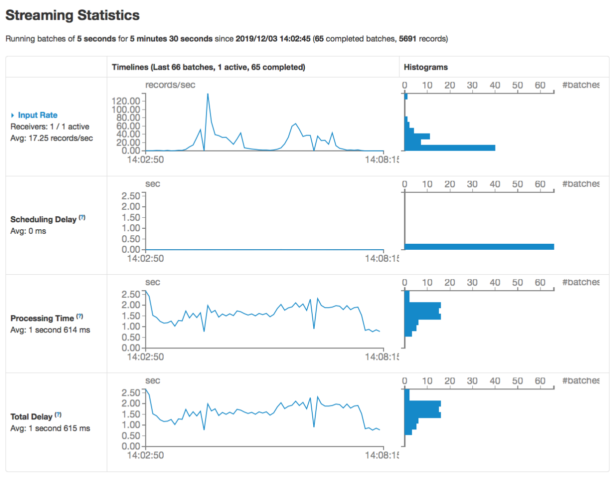
Scalability testing was already performed with a previous version of the architecture in [5]. These experiments were performed on the `Hadoop` big data cluster of the MLG. This cluster is made up of 10 slave nodes, each with 24 CPU cores, managed by a master node which is the point of access for users and handles user interaction (interactive node). The resource manager `Yarn`, which is an integral part of the `Hadoop` ecosystem, allocated 150 cores and 805GB RAM for the purpose of these tests.

Preliminary experiments with the new real-time architecture were run on a local machine with a 2.3 GHz `Intel Core i5` CPU with 4 cores and 16 GB of RAM. This hardware setup is far from the processing power that is available on the cluster and will have much slower IO due to the absence of `Hadoop`. However, it should give an initial insight of potential real-time capabilities of the implemented pipeline. Note that the code that is used in these simulations has not yet been optimized, as implementing the architecture was the priority in this phase. There are also some overheads introduced by the simulation environment, such as running docker containers and local applications from the testing machine sharing CPU cycles.

The implemented simulation uses previously collected data that was stored in `CSV` files. These files contain collected observations for three days, being the 23d, 24th and 25th of September of the year 2018. As the simulation was performed on limited hardware and accelerates the ingestion of data compared to the real situation, these files were filtered beforehand to only contain observations concerning three predetermined streets. New data is sampled from these files to simulate incoming data over one hour windows. This is a much larger sampling rate than in the real case, as we want to accelerate the simulations and are mostly interested in the correct functioning of the pipeline. The batch interval within which the processing should be completed was set to 10 seconds. This means that the simulation has to process the incoming batches 360 times faster than in the real case. This is one of the main reasons why the number of observed streets were so severely limited for the simulation. To evaluate the simulation, the output provided by the `SparkUI` interface, which is used to inspect the state of `Spark` execution, was analyzed. A snapshot of `SparkUI` after running the simulation is shown in figures 8 and 9.

Regarding user evaluation of the web interface, informal user evaluations were performed. Stakeholders from *Bruxelles Mobilité* were shown the work-in-progress interface and asked to provide informal feedback on the application. Additionally, colleagues with expertise in the area of data visualization, especially regarding mobility data, also gave their initial feedback on the currently provided functionalities.

3.2 Results

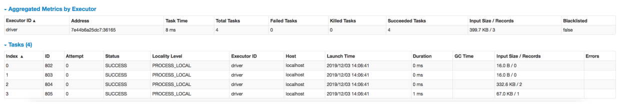

Streaming Statistics

Running batches of **5 seconds** for **5 minutes 30 seconds** since **2019/12/03 14:02:45** (**65** completed batches, **5691** records)

	Timelines (Last 66 batches, 1 active, 65 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 17.25 records/sec		
Scheduling Delay (?) Avg: 0 ms		
Processing Time (?) Avg: 1 second 614 ms		
Total Delay (?) Avg: 1 second 615 ms		

Figure 8: Overview of the SparkUI stream statistics for the simulation.

Figure 8 shows an overview of some relevant statistics collected by `SparkUI`. Here, the most informative charts are the top (input rate) and third from the top (processing time) ones. The variation in input rate shows that data ingestion peaks at certain points in a day, this illustrates the variation in HGV traffic depending on hour-of-the-day. The most important aspect of this figure is that the processing time for a batch is below the batch interval. As can be seen in the figure, the average batch processing time is 1.6 seconds, which is well below the batch interval of 5 seconds. The second chart from the top shows scheduling delay, i.e. delay between scheduling of the job and the start of processing, which always remained 0 as batches were always processed within the batch interval. For this reason the bottom chart (total delay) is the same as the processing time chart, since processing time is the only source of delay.

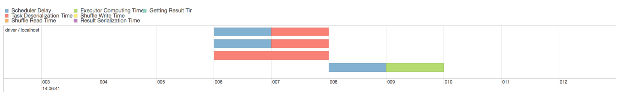

Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Blacklisted
driver	7e44b6a25dc7:36165	8 ms	4	0	0	4	399.7 KB / 3	false

Tasks (4)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	802	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/12/03 14:06:41	0 ms		16.0 B / 0	
1	803	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/12/03 14:06:41	0 ms		16.0 B / 0	
2	804	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/12/03 14:06:41	0 ms		332.6 KB / 2	
3	805	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/12/03 14:06:41	1 ms		67.0 KB / 1	

(a) Table showing the different tasks of the job, distributed over 4 cores.



(b) Event timeline of the parallel execution of the job.

Figure 9: Some important information provided by `SparkUI` on the `Spark` job that processes a single batch of data.

Figure 9 shows essential information which `SparkUI` provides on a specific `Spark` job. Figure 9a shows that the job which processes a batch was parallelized over four tasks that are each handled by a different CPU core. Figure 9b shows the timeline of events that are part of handling a `Spark` job. The blue parts of the timeline correspond to scheduling of the job, the red parts to deserialization of the data and the green parts to actually processing the incoming records. The timeline shows that most of processing time is actually spent on scheduling an deserialization of the tasks. This is because the number of records in a batch in this experiment are much smaller than in the real-world data stream. Figure 10 shows the same timeline as figure 9b when running the same task on the full dataset, i.e. with significantly more records in the processed batch. In this experiment 8 cores were allocated.

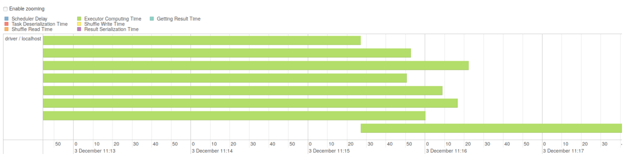


Figure 10: The event timeline of a `Spark` job when performing the simulation with all observations of a day. Ran with 8 cores allocated.

Regarding user evaluation of the web interface, the general consensus was that the current interface can already provide some basic insights, but requires more advanced tools and visualizations to provide an added value to our potential users, compared to equivalent tools that are currently available.

3.3 Discussion

The results from the performed experiments indicate that the current architecture is promising for use in a real-life scenario. Taking the results from the previous experiments in [5] and the well-known reliability of the used technologies into account, it is expected that given appropriate hardware and optimization, there should be no issue in dealing with the amounts of data we are working with.

Initial tests with the full data set where also performed on the same hardware as the preliminary experiments. Results are promising given the single node setting, but further experiments are needed to assess the architecture on a cluster setting. However, these preliminary results let us anticipate that no performance issues should be expected when using the full processing power of a big data cluster.

`SparkUI` was an important tool in debugging and analyzing performance of the implemented pipeline. The insights it provides into the execution of jobs enables detailed monitoring of how well the implemented code for a big data project performs in the `Hadoop + Spark` environment. These insights are especially useful for assessing whether the implemented pipeline will perform well, even without the use of big-data capable hardware. For example, it is with the help of `SparkUI` that we can clearly see that the scheduling and serialization overheads that can be seen in figure 9b become insignificant when working with larger data batches, as shown by the results seen in figure 10.

4 FUTURE WORK

Future work consists of finalizing the pipeline architecture and connecting the different components of the MOBI-AID big data platform together. One possible extension that is currently envisioned is to add a merged view that uses data from both the speed and batch layers to, for example, show discrepancies between the real-time traffic conditions and typical conditions. Figure 11 visualizes this extension of our current implementation.

Given this finalized implementation, we will perform extensive experiments on the MLG big data cluster which is powered by `Apache Hadoop`, as opposed to a regular office machine.

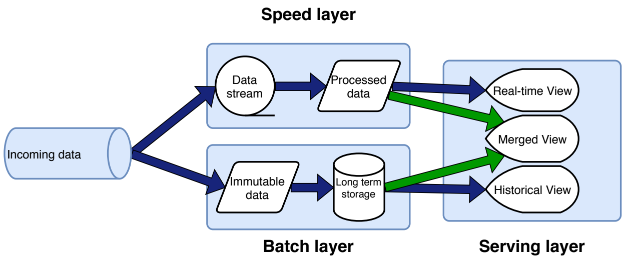


Figure 11: Overview of the future lambda architecture with a merged data view.

MLG is currently in the process of migrating to a new cluster which should provide the necessary facilities for large-scale experiments. The goal of these experiments would be to move beyond simulation. Concretely, we would hook up the implemented pipeline to the actual stream of incoming data.

Implementing and experimenting with more advanced Machine Learning approaches for forecasting will also be an important task in providing more nuanced predictions. Additionally, integrating existing mobility indicators and advanced ITS models from related research will provide appropriate metrics to policy makers. The platform should be able to perform such processing in real time and use the forecasts to simulate the impact of a policy.

Next to this, a finalized web interface will provide stakeholders with the necessary tools to make informed decisions on how to optimize traffic of goods in the Brussels Capital Region. Further extending the current interface with feedback from the users should allow us to provide this ideal interface. Concretely, further versions of the real-time tab will also include other visualizations besides the map, such as relevant charts and differences with the typical traffic situation at this hour-of-the-day. The final version of this tab should allow users to easily spot anomalies in the current traffic situation compared to historical observations.

Prototypes for the `Charts`, `Analytics` and `Predictions` tabs have not been implemented yet. It is currently under review whether these should be separate tabs, or if they should be combined into a single general Analysis tab. Conceptually, the Charts tab would contain several types of charts that show useful information, such as the typical distribution of HGVs over communes for example. The analytics tab would contain tools that allow the user to perform a specific analysis, such as constructing a model of traffic flow based on the available data. The predictions tab would put more emphasis on training and using the previously mentioned forecasting methods to predict future states of the HGV traffic in Brussels. These models could then be used by policy makers to simulate effects of certain decisions, such as modifying existing roads for example. Determining where the functionality that is envisioned should live will be one of the next steps in the design of the interface.

After the full prototype of the web interface has been implemented, extensive user studies and formal retrieval of user requirements will be done to get a better insight as to what the final web interface should provide. Iterating further and using agile software development methods should allow us to provide the end-users with the tools they need in a user friendly manner.

Finally, packaging the platform for deployment will give the different stakeholders the envisioned platform that fits their requirements and allow them to easily deploy it on their own hardware. This platform should also scale to be used for the whole country and given appropriate data, it could also be used for other countries.

ACKNOWLEDGMENTS

Arnau Dilen, Giovanni Buroni, Yann-aël Le Borgne and Gianluca Bontempi acknowledge the support of Programme Opérationnel FEDER 2014-2020 de la Région de Bruxelles Capitale (ICITY MOBI-AID project). The authors are also grateful to Bruxelles Mobilité for having provided the OBU data necessary for the work.

REFERENCES

[1] Stephen Anderson, Julian Allen, and Michael Browne. 2005. Urban logistics––how can it meet policy makers' sustainability objectives? *Journal of Transport Geography* 13, 1 (2005), 71 – 81. https://doi.org/10.1016/j.jtrangeo.2004.11.002 Sustainability and the Interaction Between External Effects of Transport (Part Special Issue, pp. 23-99).

[2] J. S. Angarita-Zapata, A. D. Masegosa, and I. Triguero. 2019. A Taxonomy of Traffic Forecasting Regression Problems From a Supervised Learning Perspective. *IEEE Access* 7 (2019), 68185–68205. https://doi.org/10.1109/ACCESS.2019.2917228

[3] Hugo Barbosa, Marc Barthelemy, Gourab Ghoshal, Charlotte R. James, Maxime Lenormand, Thomas Louail, Ronaldo Menezes, José J. Ramasco, Filippo Simini, and Marcello Tomasini. 2018. Human mobility: Models and applications. *Physics Reports* 734 (2018), 1 – 74. https://doi.org/10.1016/j.physrep.2018.01.001 Human mobility: Models and applications.

[4] Giovanni Buroni, Yann-Aël Le Borgne, Gianluca Bontempi, and Karl Determe. 2018. Cluster Analysis of On-Board-Unit Truck Big Data from the Brussels Capital Region. *21st IEEE International Conference on Intelligent Transportation Systems* (2018).

[5] Giovanni Buroni, Yann-Aël Le Borgne, Gianluca Bontempi, and Karl Determe. 2018. On-Board-Unit Data: A Big Data Platform for Scalable storage and Processing. 1–5. https://doi.org/10.1109/CloudTech.2018.8713342

[6] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Hagen Stefan, and Tim Schaub. 2016. *GeoJSON*. Internet Engineering Task Force. https://tools.ietf.org/html/rfc7946

[7] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. 2018. SCARFF: A scalable framework for streaming credit card fraud detection with spark. *Information fusion* 41 (2018), 182–194.

[8] Apache Kafka Comitters. 2019. *Apache Kafka*. Apache Software Foundation. https://kafka.apache.org/

[9] Apache Spark Committers. 2019. *Apache Spark*. Apache Software Foundation. https://spark.apache.org/

[10] Konstantinos Demertzis, Lazaros Iliadis, and Vardis-Dimitris Anezakis. 2019. A Machine Hearing Framework for Real-Time Streaming Analytics Using Lambda Architecture. In *Engineering Applications of Neural Networks*, John Macintyre, Lazaros Iliadis, Ilias Maglogiannis, and Chrisina Jayne (Eds.). Springer International Publishing, Cham, 246–261.

[11] GeoPandas developers. 2019. *GeoPandas*. GeoPandas developers. http://geopandas.org/index.html#

[12] PostgreSQL Developers. 2019. *PostgreSQL*. The PostgreSQL Global Development Group. https://www.postgresql.org

[13] Anzhelika Dombalyan, Viktor Kocherga, Elena Semchugova, and Nikolai Negrov. 2017. Traffic Forecasting Model for a Road Section. *Transportation Research Procedia* 20 (2017), 159 – 165. https://doi.org/10.1016/j.trpro.2017.01.040 12th International Conference on Organization and Traffic Safety Management in large cities, SPbOTSIC-2016, 28-30 September 2016, St. Petersburg, Russia.

[14] PostGIS Development Group. 2019. *PostGIS*. The Open Source Geospatial Foundation. https://postgis.net/

[15] S. Hadavi, S. Verlinde, W. Verbeke, C. Macharis, and T. Guns. 2019. Monitoring Urban-Freight Transport Based on GPS Trajectories of Heavy-Goods Vehicles. *IEEE Transactions on Intelligent Transportation Systems* 20, 10 (Oct 2019), 3747–3758. https://doi.org/10.1109/TITS.2018.2880949

[16] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja. 2015. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*. 2785–2792. https://doi.org/10.1109/BigData.2015.7364082

[17] Narayan Kumar. 2017. Twitter's tweets analysis using Lambda Architecture. https://blog.knoldus.com/twitters-tweets-analysis-using-lambda-architecture/.

[18] I. Lana, J. Del Ser, M. Velez, and E. I. Vlahogianni. 2018. Road Traffic Forecasting: Recent Advances and New Challenges. *IEEE Intelligent Transportation Systems Magazine* 10, 2 (Summer 2018), 93–109. https://doi.org/10.1109/MITS.2018.2806634

[19] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of massive datasets*. Cambridge university press.

[20] Nathan Marz and James Warren. 2015. *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co.

[21] Apache Hadoop Project Members. 2019. *Apache Hadoop*. Apache Software Foundation. https://hadoop.apache.org/

[22] Django Team Members. 2019. *Django*. Django Software Foundation. https://www.djangoproject.com/

[23] David Myr. 2003. Real time vehicle guidance and traffic forecasting system. US Patent 6,615,130.

[24] Daiga Plase, Laila Niedrite, and Romans Taranovs. 2016. Accelerating data queries on Hadoop framework by using compact data formats. In *Advances in Information, Electronic and Electrical Engineering (AIEEE), 2016 IEEE 4th Workshop on*. IEEE, 1–7.

[25] Mohammed A. Quddus, Chao Wang, and Stephen G. Ison. 2010. Road Traffic Congestion and Crash Severity: Econometric Analysis Using Ordered Response Models. *Journal of Transportation Engineering* 136, 5 (2010), 424–435. https://doi.org/10.1061/(ASCE)TE.1943-5436.0000044 arXiv:https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29TE.1943-5436.0000044

[26] John Ratcliffe and Ela Krawczyk. 2011. Imagineering city futures: The use of prospective through scenarios in urban planning. *Futures* 43, 7 (2011), 642 – 653. https://doi.org/10.1016/j.futures.2011.05.005 Alternative City Futures.

[27] Dilpreet Singh and Chandan K Reddy. 2015. A survey on platforms for big data analytics. *Journal of Big Data* 2, 1 (2015), 8.

[28] Hongyu Sun, Henry X. Liu, Heng Xiao, Rachel R. He, and Bin Ran. 2003. Use of Local Linear Regression Model for Short-Term Traffic Forecasting. *Transportation Research Record* 1836, 1 (2003), 143–150. https://doi.org/10.3141/1836-18

[29] CP Van Hinsbergen, JW Van Lint, and FM Sanders. 2007. Short term traffic prediction models. In *PROCEEDINGS OF THE 14TH WORLD CONGRESS ON INTELLIGENT TRANSPORT SYSTEMS (ITS), HELD BEIJING, OCTOBER 2007*.

[30] JWC Van Lint and CPIJ Van Hinsbergen. 2012. Short-term traffic and travel time prediction models. *Artificial Intelligence Applications to Critical Transportation Issues* 22, 1 (2012), 22–41.

[31] Eleni I. Vlahogianni, Matthew G. Karlaftis, and John C. Golias. 2014. Short-term traffic forecasting: Where we are and where we're going. *Transportation Research Part C: Emerging Technologies* 43 (2014), 3 – 19. https://doi.org/10.1016/j.trc.2014.01.005 Special Issue on Short-term Traffic Flow Forecasting.

[32] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.

[33] Esteban Zimányi, Mahmoud Sakr, Arthur Lesuisse, and Mohamed Bakli. 2019. MobilityDB: A Mainstream Moving Object Database System. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases (SSTD '19)*. ACM, New York, NY, USA, 206–209. https://doi.org/10.1145/3340964.3340991