Senior Thesis Technical Report

IOT Traffic Simulation: Predictive Analysis

By: Neeraj Asthana (under Professor Brunner)
Collaborators: Anchal, Vaishali, Rishabh

Contents

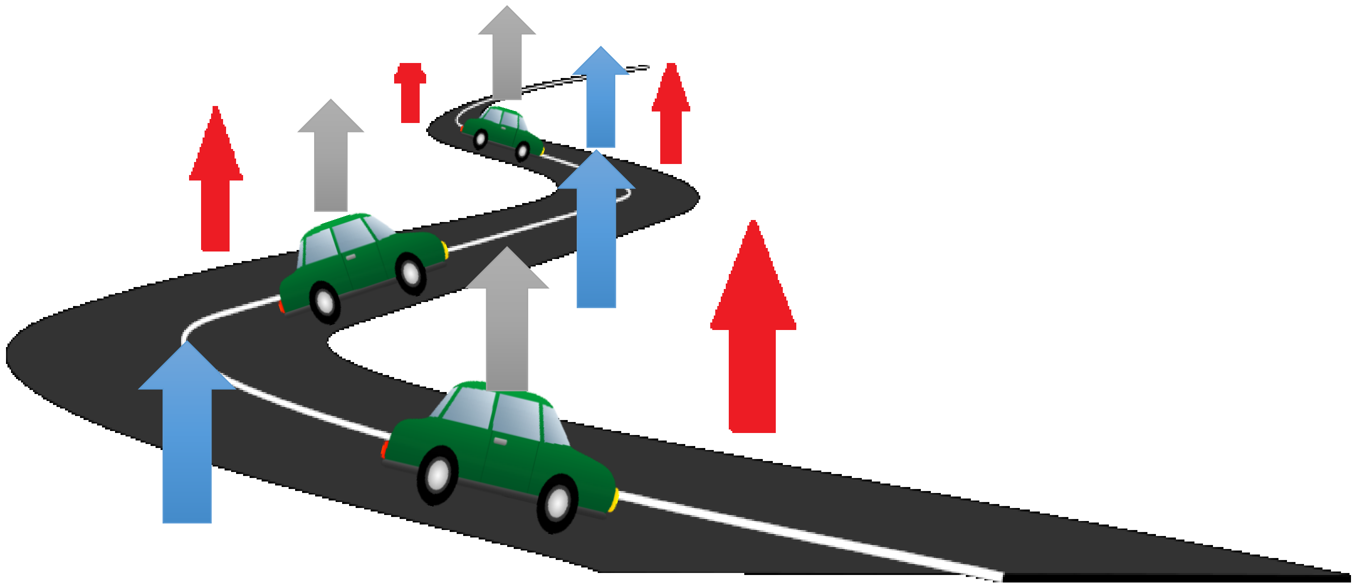
Introduction

Problem Statement

Traffic congestion and associated effects such as air pollution pose major concerns to people who are on the move. Congestion has increased dramatically during the past 20 years U.S. cities. During this time, the number of hours lost each year by an average driver to congestion has increased by 300 percent. According to Newsweek, the average United States commuter spends an average of 42 hours stuck in traffic every year, resulting in a $160 billion loss from loss of productivity. Currently, drivers are only informed by current and historical traffic conditions in their area. Additionally, increasing the capacity of the roadways is expensive and, in some areas where land is scarce, is not an option.

Solution

Many popular methods predict traffic states by aggregating streamed GPS data (coordinates, velocities, and timestamps) of many vehicles. With the recent advances in sensor technology, infrastructure for estimating temperature, pressure, humidity, precipitation, etc. are readily available on roadways and can be used to determine driving conditions. The goal and novelty of our project is to combine streamed GPS traffic estimation algorithms with dynamic road sensor data to accurately predict future traffic states and efficiently advise and direct drivers. We will make use of the time decay model for aggregating stream GPS data suggested in "Aggregating and Sampling Methods for Processing GPS Date Streams for Traffic State Estimation" and include methods to incorporate road sensor data. By using both and current traffic conditions and sensors on the road, we hope to understand how traffic may change on a single road segment in the near future (30 minutes, 1 hour, etc.).



Data Sources

There are two major data sources necessary for this project: streamed GPS data and road sensor data. I will assume that these data sets exist and are continually streaming to our backend servers as this portion of the project was produced by my fellow project group members. Rishabh's work focused on the generation of streamed GPS data while Vaishali's work focused on generating road sensor data.

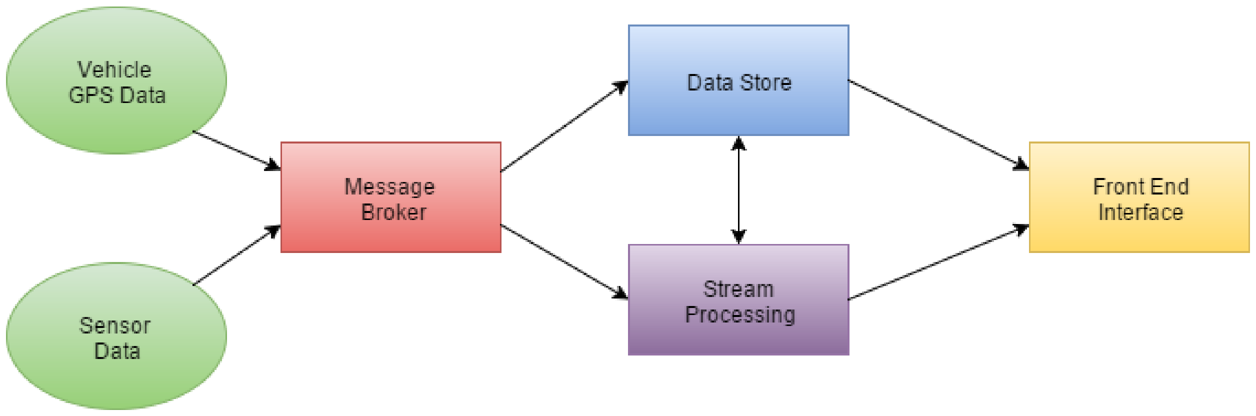
Streamed GPS Data

There will be many vehicles travelling along the roads of our model and each vehicle will be continuously be emitting data to our backend server at a predefined interval (i.e. every second, every minute, etc.). Each emitted data packet from the vehicle will be of the form (**vehicle_id**, **location**, **velocity**, **timestamp**) which can be abbreviated as (**id**, $\mathbf{v_i}$, $\mathbf{l_i}$, $\mathbf{t_i}$). The **vehicle_id** is a unique identifier for each vehicle. The **location** consists of a geodesic latitude and longitude coordinate. The **velocity** field consists of the vehicle's current velocity which will be measured in meters per second. The **timestamp** references the time at which the location and velocity data was recorded in the form "*%Y-%m-%d %H:%M:%S*".

Streamed Road Condition Data

Our model allows us to generalize to any number or type of sensor we desire, however for now I will only focus on a single type of sensor which measures road surface temperature. In the future I will focus on diversifying and combining the data that other types of sensors produce. Sensors will be placed along a roadway at a predefined interval ((i.e. every mile, every five miles, etc.) and I will assume that these locations are known. Each sensor will periodically emit information to our backend server at a predefined interval (i.e. every second, every minute, etc.). Each sensor data packet will be in the form: (**sensor_id**, **temperature**, **timestamp**). The **sensor_id** is a unique identifier for each sensor. The **temperature** defines a road surface temperature at the location of the sensor in Celsius degrees. The **timestamp** references the time at which the temperature was recorded in the form "*%Y-%m-%d %H:%M:%S*".

Technical Specifications



Our backend will be implemented on the NCSA ACX cloud computing cluster. The purpose of these systems is to efficiently stream our data sources to allow for real time traffic state prediction. I will assume that all of these services are running and are properly benchmarked as Anchal will be working on this portion of the project. My work will mostly be on the stream processing/cloud computing system by implementing model mentioned below with the data streamed from our message broker. A network diagram of the data flow is included for reference along with descriptions of what happens at each step.

Cluster Resources

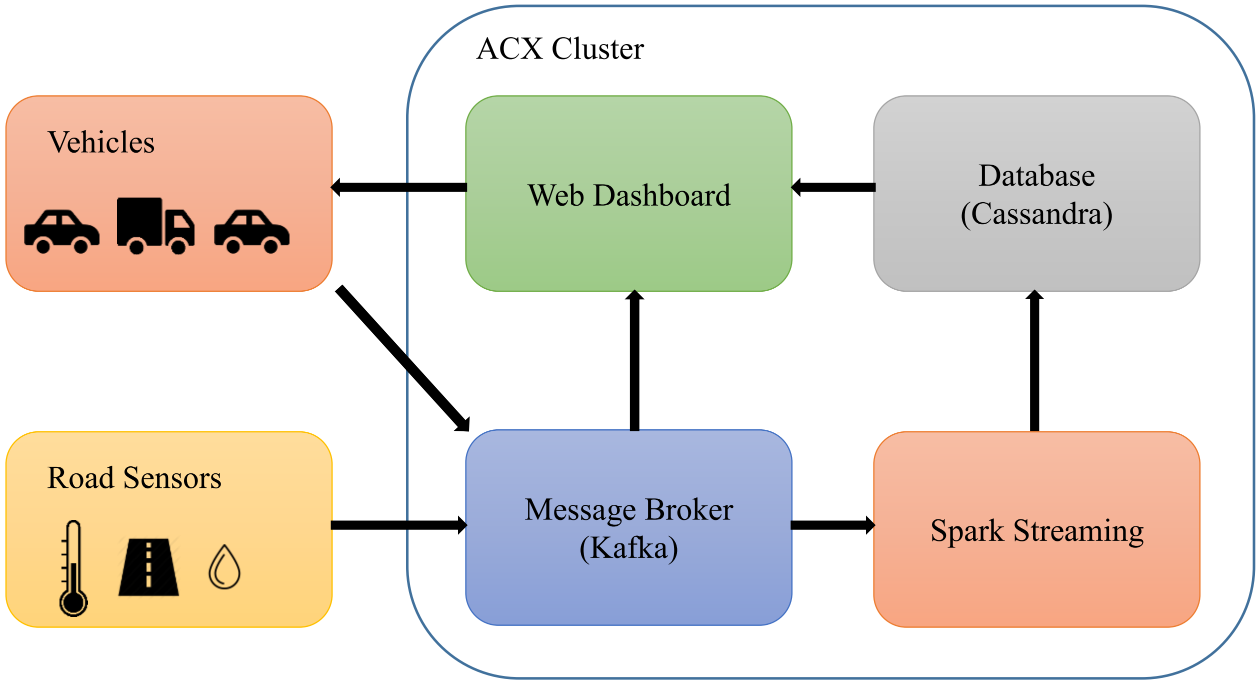
Cluster	Workers	Cores	RAM
Kafka	2	8	8 GB
Cassandra	2	4	4 GB
Spark	6	24	17.2 GB

Message Broker (*Kafka*) – All vehicle and sensor data will be aggregated and ordered by a single message broker for efficient processing. Large amounts of real time data will continually be streaming into our backend and a message broker like Apache Kafka will help us efficiently manage large amounts of messages. All the data captured at this step will be aggregated and ordered so that it can be sent to the data storage and stream processing units. The Kafka cluster contains two major queues, “traffic” and “weather”, known as topics. The “traffic” topic contains streamed vehicle data and each entry is of the form (*latitude, longitude, velocity, time, vehicle_id*). The “weather” topic contains streamed vehicle data and each entry is of the form (*road_segment_id, time, measurement*).

Stream Processing (*Spark*) – The most recent data from the cars and the sensors will be sent to this unit to update the current road condition models. The newest data will be streamed into the system and be used (along with historical GPS data from the data store) to generate a new model describing the road conditions. There are considerations to also use or benchmark *Apache Flink* or *Apache Storm* in place of Spark as these may scale better to larger data streams.

Date Store (*Cassandra*) – All data will be stored in this unit to either be included in future models or displayed on the front end interface.

Data Pipeline



Describes the general flow of the data between different clusters. Data is generated by the Car and

Time Decay Model

Our model incorporates the time decay model suggested in “Aggregating and Sampling Methods for Processing GPS Date Streams for Traffic State Estimation” with road sensor data to estimate traffic states on specific road segments. I will implement the aggregation based method mentioned in the paper as it is easier to implement and is efficient at estimating average speeds on road segments. I will then adjust the calculated average speed on each road segment based on data from road sensors.

I will assume that the client side (each vehicle) sends packets of the form (**vehicle_id**, **location**, **velocity**, **timestamp**) and each sensor sends packets of the form (**sensor_id**, **temperature**,

timestamp), which all will be ordered and aggregated by the message broker to form a stream and sent to the stream processing system.

Previous models of estimating traffic states from GPS data streams focus on either historical data or a sliding window (SW) sampling method. Historical methods attempt to aggregate previous speeds of vehicles on a particular road (each of which has an equal weight); however this method is not extremely accurate in the short term as GPS data streams are volatile and evolve rapidly based on different circumstances. The sliding-window sampling method conversely saves the most recent GPS data stream and uses only a select few point to estimate new traffic states. This also may not be completely accurate when estimating traffic states as it creates unstable traffic conditions that may not reflect the average behavior of a specific road. Therefore, I plan to implement a time decay model which highly values new data records but also factors old data records. This method is also online and real time, allowing us to process data quickly and return current information.

The idea of the time decay model is to weight new records higher than old records in that same location. This is accomplished by weighting each record with a value *w* using the following exponential decay function where *i* represents the ith GPS record, *t* is the processing time, and t_i is the timestamp of the current record:

$$\boldsymbol{w(i\,,t) = exp\big(\beta(t_i - t)\big)} \;\;, \quad \textit{where } \beta > 0 \textit{ and } t \geq t_i$$

Using this model, I receive weights for each of the data records on a specific road segment and older records are weighted less than newer records.

We must also notice that high velocity data is more useful than low velocity data as low velocities can signal events that are not related to traffic flow such as parked vehicles or stopping at a red light. Therefore, I must weigh high velocity data as more important than low velocity data which is accomplish by assigning another weight to each record using a function *g* that is positive, nondecreasing, and monotonic as follows:

$$\boldsymbol{w^v(i) = g(v_i)}$$

Then I can assign a full weight to a specific GPS data point by the following equation:

$$\boldsymbol{w^*(i,t) = w(i,t) \times w^v(i)}$$

For each road segment, defined as *r* (where r is a latitude and longitude coordinate), I can estimate the average velocity at time *t*, using the following equation:

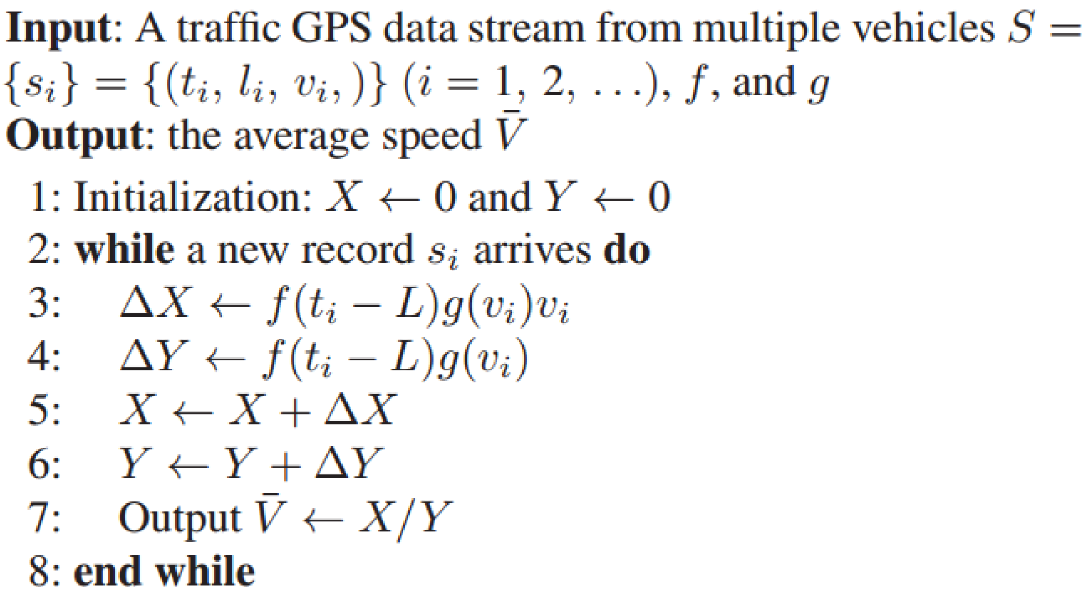
$$\boldsymbol{v(r) = \frac{\sum_{i=1}^{m} w^*(i,t)\ \times\ v_i}{\sum_{i=1}^{m} w^*\ (i,t)}}$$

This model can easily be updated with newly streamed data points as well. Let *X* and *Y* be the numerator and denominator of *v(r)* respectively

$$\boldsymbol{X = \sum_{i=1}^{m} w^*(i,t)\ \times\ v_i}$$

$$\boldsymbol{Y} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{m}} \boldsymbol{w}^*(\boldsymbol{i},\boldsymbol{t})$$

Then, we can update the model by incrementally updating the values for *X* and *Y* in real time using the following algorithm:



Input: A traffic GPS data stream from multiple vehicles $S = \{s_i\} = \{(t_i, l_i, v_i,)\}$ $(i = 1, 2, \ldots)$, f, and g
Output: the average speed $\bar{V}$

```
1: Initialization: $X \leftarrow 0$ and $Y \leftarrow 0$
2: while a new record $s_i$ arrives do
3:     $\Delta X \leftarrow f(t_i - L)g(v_i)v_i$
4:     $\Delta Y \leftarrow f(t_i - L)g(v_i)$
5:     $X \leftarrow X + \Delta X$
6:     $Y \leftarrow Y + \Delta Y$
7:     Output $\bar{V} \leftarrow X/Y$
8: end while
```

Dynamic Map Matching Algorithm

Intuition

Spark Code

Spark Streaming

Mechanics

State Preservation

Model Update

Future Work

In the future we would like to incorporate many features into our models which were not possible to include during the spring semester.

Dropped/Missing Data

Not all data is properly delivered to our backend servers as receivers could lose service or turn off. We must adapt our model to successfully predict traffic conditions even when every piece of data is not completely available.

Security

How can we protect our network from hackers?

Unidentified Vehicles

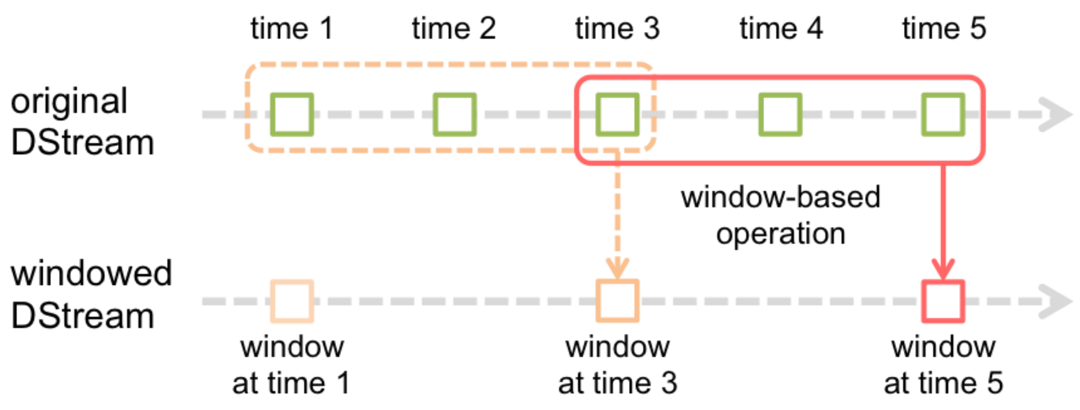
Some vehicles are not equipped with GPS devices and we must adapt our models to incorporate these vehicles as well.

Variable Sensors

Many types of sensors are available to monitor weather and traffic conditions. Currently, our model only uses road surface temperature sensors. However, we must incorporate other types of sensors into this model to try and see how different combinations of road conditions change vehicle behavior.

Sliding Window

Delete insignificant and outdated data inside the database.



Acknowledgements

We acknowledge support from the National Science Foundation Grant No. AST -1313415, the National Center for Supercomputing Applications, the University of Illinois and Microsoft Azure.

References

Laboratory for Cosmological Data Mining

National Center for Computing Applications ACX Cluster

Apache Spark 1.5.0 Documentation → http://spark.apache.org/docs/latest/

Ipython Notebooks → http://ipython.readthedocs.org/en/stable/

Docker → https://docs.docker.com/

Spark-Ipython Notebook Integration → http://www.kuntalganguly.com/2015/06/configure-ipython-notebook-with-apache.html

Cassandra → http://docs.datastax.com/en/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html

Time Decay Scholarly Paper Intelligent Transportation Systems, IEEE Transactions on, Issue Date: Dec. 2013, Written by: Jia-Dong Zhang; Jin Xu; Liao, S.S.

Kafka → http://kafka.apache.org/documentation.html

https://arrayofthings.github.io/

Appendix