Framework for Semantic Integration and Scalable Processing of City Traffic Events

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

by

Surendra Brahma Marupudi
B.Tech., Bapatla Engineering College, 2012

2016
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Surendra Brahma Marupudi ENTITLED Framework for Semantic Integration and Scalable Processing of City Traffic Events BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Amit P. Sheth, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Amit P. Sheth, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Tanvi Banerjee, Ph.D.

Robert E. W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT


Intelligent traffic management requires analysis of a large volume of multimodal data from diverse domains. For the development of intelligent traffic applications, we need to address diversity in observations from physical sensors which give weather, traffic flow, parking information; we also need to do the same with social media, which provides live commentary of various events in a city. The extraction of relevant events and the semantic integration of numeric values from sensors, unstructured text from Twitter, and semi-structured data from city authorities is a challenging physical-cyber-social data integration problem.

In order to address the challenge of both scalability and semantic integration, we developed a semantics-enabled distributed framework to support processing of multimodal data gushing in at a high volume. To semantically integrate traffic events related complementary data from multimodal data streams, we developed a Traffic Event Ontology consistent with a Semantic Web approach. We utilized Apache Spark and Parquet data store to address the volume issue and to build the scalable infrastructure that can process and extract traffic events from historical as well as streaming data from 511.org (sensor data) and Twitter (textual data). We present large scale evaluation of our system on real-world traffic-related data from the San Francisco Bay Area over one year with promising results. Our scalable approach was able to decrease the processing time of the test case we present in this work from two months to less than 24 hours. We evaluated our scalability method by varying input data loads and the system showed stability in the performance. Additionally, we evaluated the performance of our semantic integration method by answering questions related to traffic anomalies using multimodal data.
# Contents

1 Introduction ................................. 1
  1.1 Challenges .................................. 3
  1.2 Approach .................................... 4

2 Related Work ................................. 6
  2.1 Scalable processing ......................... 6
    2.1.1 Lambda Architecture ..................... 7
  2.2 Semantic Integration ....................... 9

3 Scalable Processing ....................... 13
  3.1 Data Sources ............................... 13
    3.1.1 511.org .................................. 13
    3.1.2 Twitter .................................. 14
  3.2 Scalable Processing Infrastructure ........ 15
    3.2.1 Spark ...................................... 15
      3.2.1.1 Spark Streaming API ................. 15
      3.2.1.2 Spark SQL ............................. 17
    3.2.2 Apache Parquet ......................... 18
  3.3 Building the models ....................... 18
  3.4 Event Extraction ......................... 22
    3.4.1 Traffic Anomaly Detection from Sensor Observations 22
    3.4.2 Traffic Events Extraction from Twitter ........ 25

4 Semantic Integration ....................... 27
  4.1 Semantic Web Standards .................... 27
  4.2 Traffic Event Ontology .................... 30
  4.3 Publishing Traffic Event Stream Infrastructure .... 32

5 Evaluation ................................. 35
  5.1 Evaluation over Scalability ............... 35
  5.2 Evaluation over Semantic Integration .... 37

6 Future Work .................................. 40
7 Conclusion 42

Bibliography 43
# List of Figures

1.1 Multimodal input data formats. ........................................ 3
1.2 The Components of City Traffic Events extraction, integration, and pub-
lishing Framework. .......................................................... 5

2.1 Overview of the lambda architecture; ................................. 7

3.1 Spark Streaming input and output formats ......................... 16
3.2 Spark Streaming flow .................................................... 17
3.3 Traffic Anomaly Detection from sensor observations. .......... 19
3.4 Overview of hash key generation. .................................. 20
3.5 Sample RDD partitions after applying hash partitioner. ........ 21
3.6 Traffic Anomaly Detection from Sensor Observations .......... 23
3.7 Sliding window operations to process small batches in near real-time 24
3.8 Traffic Event Extraction from the Twitter Stream Pipeline. .... 25

4.1 Semantic Web technology stack ............................... 28
4.2 The Event Ontology .................................................... 30
4.3 Traffic Event Ontology;  ............................................. 31
4.4 Part of OWL Traffic Event Ontology represented as follows 32
4.5 Integration and Publishing pipeline ............................... 33

5.1 Evaluation of serial and parallel implementations ............... 35
5.2 Evaluation of Scalable Processing with varying configurations 36
5.3 Evaluation of scalable processing after increasing the input data 10 times 37
5.4 Evaluation of question ”What are all the events that affect traffic on link 101010?” ......................................................... 38
5.5 Evaluation of question: ”What might be the reason for traffic delay on link 101010?” ................................................. 39
List of Tables

4.1 Referenced Ontologies ................................................. 30
Acknowledgement

I would like to take this opportunity to thank everyone who has helped me throughout my journey. First and foremost, I would like to express my sincere gratitude to my advisor Prof. Amit Sheth for the continuous guidance and opportunity to work and learn from outstanding students at Ohio Center of Excellence in Knowledge Enabled Computing (Kno.e.sis). His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Krishnaprasad Thirunarayan, Dr. Tanvi Banerjee for their insightful comments and encouragement. I would specially like to thank my mentor Dr. Pramod Anantharam.
Dedicated to

my mother Siva Parvathi and my sister Nagasree
Introduction

According to United Nations, by 2050 around 70% of the expected global population of 9.3 billion will live in big cities [36]. Consequently, traffic congestion is set to become a major problem in the world’s cities. This is aptly characterized or described by the following conclusion presented by a major study: "In many cities, the survey recorded significant increases, when compared with last year, in the number of respondents who said that roadway traffic has increased their levels of personal stress and anger and negatively affected their performance at work or school." [48]

A growing network of smart objects can be used to enhance the services provided to the citizens (e.g., traffic and weather updates from physical objects connected to the Internet). The physical objects can be anything from sensors to computers (or smartphones). In order to conserve resources with improved efficiency, the Internet of Things (IoT) can be employed to make significant impact on many sectors such as smart health, industrial automation, smart homes, and smart cities. In addition, recently Twitter, a microblogging platform, has become the primary medium for sharing public opinions on varying topics. It has more than 288 million regular users worldwide generating over 500 million tweets a day [43]. Interactions among its users cover a wide range of topics of varying importance. Twitter has developed into a near real-time source of information. With the availability of huge amounts of data from the IoT and social media, city authorities around the world are overloaded by data from the sensors deployed throughout the city, data from social media, and data from incident reports reported on the Web. Thus, infrastructures that can process,
analyze, integrate, and make sense of the huge amount of multimodal data flowing through the Web from cities have gained importance. Most of the current approaches focused on mining either social media data or IoT data [33], [57], [55], [42]. For example, consider a traffic scenario in a city. If we process the stream of observations from the sensors deployed on the road, this would only give us an indication of whether there is a traffic congestion or not. Based on this information alone, decision makers do not have a good way to interpret the traffic variations. They can utilize events reported on other modalities such as Twitter and 511 incident reports to interpret the sensor data. However, traffic events (i.e. traffic anomalies) extracted only from sensor observations cannot be used to answer the questions like:

1. What is the cause of traffic congestion on link N?
2. When did traffic congestion happen due to an accident on link M?
3. What are all the events (e.g. accident, football game, construction) that affect traffic on link K?

With a view to answering the questions mentioned above, this thesis explores a scalable framework named the City Traffic Events extraction, integration and publishing Framework (CTEF). It processes and extracts traffic-related events from the huge amount of multimodal and heterogeneous input data sources, such as sensor observations (numerical values), citizen sensors (textual observations), and incident reports (semi-structured incidents data) from 511.org in near real-time to get more insights on traffic congestion. This is an excellent instance of emerging breed of applications exploiting physical-cyber-social data [53].

We now discuss the challenges in building the CTEF infrastructure and present our overall approach.
1.1 Challenges

The amount of data generated from the Internet of Everything by 2013 was around 4.4 zettabytes (i.e. $10^{21}$ bytes), and this grows exponentially every year. It will reach around 44 ZB by 2020 [IDC 2014]. With this huge amount of data generation, city authorities around the world are overloaded with the data from the sensors (e.g., video cameras, environmental sensors, traffic sensors, and mobile phones) deployed throughout cities, data from social media, and content from the Web [11], [10]. It is challenging to build an infrastructure to process the data and make sense of it since this involves the processing and integration of a vast amount of multimodal, heterogeneous, historical, and near real-time input data.

![Figure 1.1: Multimodal input data formats.](image)

Figure 1.1 demonstrates a traffic scenario in a city that involves three modalities with heterogeneous data formats (1) part of a semi-structured incident report from 511.org, (2) sample tweets from Twitter data, and (3) sample travel time and speed sensor observations from the SF Bay Area.
1.2 Approach

This thesis provides (1) an infrastructure for extracting traffic anomalies (also known as traffic events) from sensor observations and traffic events from the Twitter data, (2) an approach for the integration of multimodal traffic data, (3) the ability to publish city data on the Web, and finally, (4) a comprehensive evaluation of the developed methods. In order to achieve the aforementioned outcomes of this work we built a scalable infrastructure to process and integrate a large amount of historical and near real-time sensor and Twitter data using a Lambda Architecture (LA), which is a scalable data processing architecture designed by Nathan Marz [4]. LA is used to develop CTEF, which is a unified architecture for both real-time and batch processing in a single framework for city traffic data (i.e. sensor observations and tweets from Twitter). CTEF can be divided into four main components as shown in Figure 1.2. They function as follows:

1. **Data Collection:** The infrastructure starts with data collection from multiple input data sources such as travel time and speed observations from 511.org, tweets from Twitter, and incident reports from the 511.org.

2. **Traffic Event Extraction:** This has two layers, which include a batch layer and a speed layer, to provide the event extraction for both historical as well as real-time data. The whole incoming stream is sent to the batch layer to be stored in the Master Data Store; it is then processed at the end of every month in order to create the models used to extract traffic events in the speed layer. In addition to that, data is also sent to the speed layer, where the incoming stream is processed in near real-time and then traffic events are extracted.

3. **LOD Creator:** Traffic events extracted from the Twitter, sensor observations, and incident reports from 511.org are transformed into RDF triples using the traffic event ontology and then published on the Web consistent with Linked Open Data (LOD) framework [35] for broader consumption.
4. **Traffic Events Access:** In this module, we provide a user interface built on the CTEF infrastructure so that the people can access the traffic events in near real-time.

![Figure 1.2: The Components of City Traffic Events extraction, integration, and publishing Framework.](image-url)
Related Work

In this chapter, we will discuss the related past work. First, we explain prior work on scalable processing of data from dynamic sources like IoT and social media. Second, we discuss the integration of different traffic event sources.

2.1 Scalable processing

Numerous efforts from both the industry and academia have been made towards building the scalable infrastructure for IoT applications and/or social media applications [8], [54]. In this thesis we focused more heavily on an intelligent traffic data platform that can deal with both IoT and social media data. We also consider both volume and velocity requirements of historical and near real-time data.

There has been much effort made to enable real-time stream processing in the cloud. These stream processing platforms are developed for applying complex logic over continuous data flows with high performance. Collaborative Open Market to Place Objects at your service.

COMPOSE [40] provides scalable back-end infrastructure for IoT end-to-end applications; in particular, it provides a platform for the storing, streaming, and processing of sensor data. Antonio et al.[39] developed a platform at the University of Bologna to extract mobility habits and Points of Interest (POIs) of citizens in cities from a huge number of harvested hotspots. It uses a NoSQL database MongoDB in the backend for storage and
processing. Matador [37] is a crowd-sensing application to provide context-awareness by exploiting internet-connected and sensor equipped portable devices. In addition to that, real-time social data analysis platforms such as Twitris [8] and Twinder [54] provide near real-time analysis of social data streams. We follow it up with a discussion on Lambda Architecture [4] for Big Data processing.

2.1.1 Lambda Architecture

Nathan Marz designed the Lambda Architecture (LA) when he was working on distributed processing systems at Backtype and Twitter. It is a generic, fault-tolerant, and scalable architecture for solving Big Data problems [4].

![Overview of the lambda architecture](http://lambda-architecture.net/)

Figure 2.1: Overview of the lambda architecture;

Source: [http://lambda-architecture.net/](http://lambda-architecture.net/)

As shown in Figure 2.1, LA has 5 major components. The first is a new data which is input stream entering system is forwarded to both speed layer and batch layer for processing. The second is a master dataset which stores all incoming data. The third is a batch view, which is an end result from the batch layer. The fourth is a real-time view which is a
result of speed layer. The fifth is to provide an interface to respond with computed results from both speed layer and batch layer for incoming queries.

Mariam et al.[44] implemented the Lambda Architecture design pattern to handle sensors and the smart phone data-handling backend on Amazon EC2. It provides affordable real-time data processing of Big Datasets in any given scenario by utilizing database management, query management, and cloud computing. In addition to that, they also utilize cost models to efficiently use Amazon AWS cloud services to minimize the cost for overall computation of the given dataset. For real-time stream data processing, they used Kinesis, which is an Amazon AWS cloud service for real-time streaming data.

AllJoyn Lambda is an architecture proposed by Massimo et al.[12] to address large-scale storage and analysis of data from IoT by utilizing a basic Lambda Architecture. It was developed using Storm and MongoDB and was evaluated in the context of a "smart home" case study. The Lambda-CoAP architecture is discussed in Manuel et al.[5] for processing, analysing, and querying large-scale IoT data with the help of stochastic functions in real time. On other hand, it also provides support for the three most important Vs of Big Data—the volume, velocity, and variety of data produced by IoT devices.

Fangjin et al.[56] developed open-source platform using the Lambda Architecture for fast and flexible queries on real-time data, which is known as RADStack. The underlying infrastructure it is built with Samza for low-latency analytic queries on the real-time data stream and Hadoop for correctness and flexible batch processing of historical data for a specific time duration. Additionally, it also uses Kafka to provide input streaming data and Druid to support interactive queries.
2.2 Semantic Integration

Integration of heterogenous data sources has been a research topic for many years. In the early days, standards like HTML\(^1\) and XML\(^2\) were developed for reliable data exchange. And afterwards vocabularies or languages evolved to describe concepts and structures of data, but these are only syntactic descriptions and lacked in formal semantics. There has been many efforts which use XML markup languages to describe traffic events. For example, CARS (Condition Acquisition and Reporting System) \([3]\) is a real-time system deployed to view road, weather, travel, and traffic information. It uses the national ITS\(^3\) standard XML\(^4\) and TMDD (Traffic Management Data Dictionary) to exchange incident reports. This application can be seamlessly integrated with any other ITS application.

Zhai et al. 2008a \([59]\) developed an integration platform on Semantic Web\(^5\) using ontology for a digital city. It has three main layers. The first layer is a distributed heterogeneous data source layer. The second layer is an information integration layer. The third layer is an application system layer. The first layer consists of various city management systems as the main data sources like e-government, public transport, GIS, and energy systems. The second layer has two main components. First, an ontology server which extracts an initial ontology from given data sources using software reverse-engineering and is later refined by the ontology designers. Consequently every data source generates its own local ontology. Afterwards, a global ontology is generated using an ontology reasoner to check ontology consistency and merge ontologies. It is also capable of returning relationships between terms, attributes associated with terms, and mapping terms. Second, an integrated database creates a set of queries when application requires data from the information platform. The third layer does information retrieval and supports decision making by utilizing the large amount of traffic data; it manages this through data fusion and data mining tech-

\(^1\)W3 HTML – [https://www.w3.org/html/](https://www.w3.org/html/)
\(^2\)W3 XML – [https://www.w3.org/XML/](https://www.w3.org/XML/)
\(^3\)ITS – [https://www.standards.its.dot.gov/](https://www.standards.its.dot.gov/)
\(^4\)W3 XML – [https://www.w3.org/XML/](https://www.w3.org/XML/)
\(^5\)W3 SemanticWeb – [https://www.w3.org/standards/semanticweb/](https://www.w3.org/standards/semanticweb/)
Traffic information on Semantic Web is represented using a Resource Description Framework (RDF\textsuperscript{6}) and a fuzzy ontology\textsuperscript{[60]}. It incorporates fuzzy theory into the ontology to handle uncertainty of information and knowledge. Traffic accident data is described using the fuzzy linguistic variable ontologies. With this, they introduce a new fuzzy linguistic variables, like "age" and "speed", as a RDF data model. Semantic relations between fuzzy concepts are used by SeRQL (Sesame RDF Query Language) to construct semantic query expansion.

Consoli et al.\textsuperscript{[38]} The Smart City data model is used to extract data from sources and build an ontology which describes data sources and publish them as Linked Open Data. It followed W3C standards for good practices of ontology design. It also integrated data sources coming from various organizations related to the municipality of Catania, such as (1) geodata from the Geographic Information System (GIS) of the city, (2) data from the public bus system, (3) data from the public lighting system, (4) data on the maintenance of the sidewalks, signs, and state of roads, and (5) historical data from an urban fault reporting service. All these data sources are heterogenous in nature so they developed customized engineering approaches to deal with data, such as one script to parse data which is in JSON format, and another script to parse data which is in XML format and to create RDF/OWL triples. The knowledge discovery step is performed on each data object produced and linked to DBpedia to enrich the knowledge base. Data are stored in RDF graphs, and SPARQL endpoint is accessible as a REST Web service. The open source publishing framework Exhibit\textsuperscript{7} is used for data-rich interactive web pages.

ASSIST (Access, Semantic, Search, and Integration Service and Translation)\textsuperscript{[46]} is a platform for performing translation, data integration, and semantic data retrieval. They explored data integration from OpenStreetMap\textsuperscript{8} and Geo Sensor Web Services using existing

\textsuperscript{6}W3 RDF – https://www.w3.org/RDF/
\textsuperscript{7}Smile-Widgets – http://www.simile-widgets.org/exhibit/
\textsuperscript{8}OpenStreetMap – https://www.openstreetmap.org/
standards to allow sensor data to be accessed over the Web by OGC Sensor Web Enablement. Linked Sensor Middleware (LSM) [45] is a platform that integrates live, real-world sensor data using the Semantic Web. It uses a Semantic Sensor Network (SSN) ontology\(^9\) to describe the sensor metadata and graph-based sensor data stream. It also provides an interface for publishing, visualizing, annotating, and querying sensor data (e.g. flight status, weather, train/buses arriving times, nearby metro stations, street cameras, etc.).

Patni et al.[47] framework integrated the heterogeneous sensor data streams based on Semantic Web technologies. Initially, the raw sensor stream was converted to an O&M\(^10\) stream, and later it was converted to RDF stream. It extracts the features from the sensor data streams in real-time and retrieves definitions for the features from weather ontology and National Oceanic and Atmospheric Administration (NOAA)\(^11\). They are used to filter sensors based on their capability to detect a feature which will substantially improve the performance. Reasoning with features is done by using SPARQL later feature stream generated from the sequence of detected features. At the end, the feature stream is published as Linked Data and accessed using SPARQL queries. Samper et al.[50] proposed a formalized knowledge model for road traffic using Semantic Web technologies. Development of this knowledge model in the traffic domain served as a proof concept and it made handling heterogeneous and distributed information flexible. For this, they used several techniques, such as XML. Ontologies and concepts are translated into different languages based on the creation of correspondences, even though, in theory, they are different in their structure and content. Searching the information given by traffic services is easy using this integration platform.

Finally, the majority of the existing works target a specific problem such as processing and integrating the IoT data or social data without providing a general framework that allows for the processing and integrating of both IoT and social data. In our solution, we

---

\(^10\)O&M – http://www.opengeospatial.org/standards/om
\(^11\)NOAA – http://www.noaa.gov/
developed a framework by utilizing Spark, Parquet, and Semantic Web technologies to process and integrate IoT and social data as well as handle the querying of the traffic data.
Scalable Processing

One of the aims of this thesis is to build the models from the huge volume of historical data and extract the traffic related events from incoming high velocity input streams. Here the key challenge is building the model for each link and extracting traffic related events in near real-time. In this section, we will discuss the implementation. First, we discuss input data sources such as Twitter and 511.org. Second, we discuss the infrastructure used to build this system such as Spark and Parquet. Third, we explain building the models from the huge volume of historical data. Finally we discuss extracting the traffic events from the both Twitter and sensor data in near real-time.

3.1 Data Sources

We utilize both machine sensor observations (travel time and speed observations) and observations reported by people about traffic conditions (tweets and publicly available events) from the San Francisco Bay area.

3.1.1 511.org

511.org\(^1\) partnered with Caltrans\(^2\), which is the lead agency in operating the California freeway system. They use loop-detectors embedded in the pavement along with off-
pavement sensors to check hotspots on the freeway. They also use closed circuit television cameras (CCTVs) and partner with the California Highway Patrol (CHP), which provides important data on accidents, stalled vehicles, debris, etc., to the 511 system [7]. It provides a streaming API for retrieving the speed, travel time, and incident data for individual links on freeways, highways, and expressways in the nine-county San Francisco Bay area.

Sensor observations were collected from the San Francisco Bay Area starting from May 2014 through July 2015. We collected around 1.4 billion sensor observations (including travel time and speed observations) from around 3,622 sensors deployed, one for each road link in the San Francisco Bay Area. We preprocessed the dataset to encompass only links with data, so we filtered to select only the links with data for us to create models resulting in 2,548 links.

We collected the general incident reports, scheduled, and active events from 511 using their traffic data feed API. There were around 1,500 active and scheduled events and incident reports from May 2014 to August 2015. We used the TOMS schema defined by 511 to process the input traffic data feed in the form of an XML file and stored them in the MySQL database.

### 3.1.2 Twitter

Twitter is microblogging platform. It has more than 500 million users worldwide and generates over 500 million tweets a day [43]. Interactions among its users cover the wide range of topics of varying importance, with this it has developed into a near real-time source of information. An emerging trend has been the extraction of events related to traffic from social streams [33], [55], [49]. Integrating traffic events extracted from social media with events detected from sensor observations can provide a valuable addition [34].

The Twitter dataset was collected from the San Francisco Bay Area starting from May 2014 and extending to July 2015 by using Twitters public streaming API [9]. The dataset contains around 18 million tweets. In this dataset, we just keep the location, timestamp,
text, tweet ID, user ID, and bounding box. The tweets location is stored in the form of latitude and longitude which will help to identify the location of the events extracted from the tweets.

### 3.2 Scalable Processing Infrastructure

In the previous section, we discussed input data sources and the amount of data being generated from the San Francisco Bay area. We now discuss the scalable processing infrastructure, such as Spark and Parquet to deal with huge amount of historical and real-time data from both 511.org and Twitter.

#### 3.2.1 Spark

Spark is a open source platform for large-scale data processing. It is flexible and runs on Mesos, Hadoop YARN, standalone, or in the cloud (e.g EC2). We can also access the data sources from different platforms such as HDFS, HBase, Cassandra, and S3. It supports quick application development in R, Scala, Python, and Java [1]. It supports both stream processing and batch processing when compared with Hadoop (for a batch processing engine) and Apache Storm (for a stream processing engine). In certain situations, it runs 100 times faster than Hadoop [6]. We can build robust interactive applications using Apache Spark. First we discuss Spark Streaming API, which provides the ability to process the incoming data streams in near real-time. Second we discuss Spark SQL, which provides the ability to batch process the large data sets.

#### 3.2.1.1 Spark Streaming API

The Spark Streaming API is used to process the streaming data in near-real time. We can reuse the code written for Spark Streaming for batch processing as well. It can
read data from Twitter\(^3\), ZeroMQ\(^4\), Flume\(^5\), Kafka\(^6\), HDFS\(^7\), TCP sockets\(^8\), Kinesis\(^9\), and custom data sources [2].

Figure 3.1: Spark Streaming input and output formats


The Spark Streaming API, along with Twitter and TCP sockets, is used to ingest the tweets and sensor observations from the respective sources. To ingest the Twitter data source, it uses the Twitter Streaming API\(^{10}\) to get the public tweet stream. The Twitter Streaming API establishes a persistent HTTP connection; additionally, it requires authentication to connect to the Twitter server [9]. To get the tweets from SF Bay Area, the Twitter Streaming API requires bounding box\(^{11}\), which is an area defined by two latitudes and longitudes where latitude is a -90 to 90 decimal number and longitude is a -180 to 180 decimal number. The Twitter server returns only the geo-tagged tweets within the given bounding box of SF Bay Area. Further, to ingest the 511 data source, we are using the TCP sockets to receive 511 observations written to TCP socket. Here, we use the 511 traffic data feed API to retrieve the sensor observations stream from the San Francisco Bay Area. Authentication

\(^3\)Twitter – [https://twitter.com/](https://twitter.com/)
\(^5\)Flume – [https://flume.apache.org/](https://flume.apache.org/)
\(^{10}\)Twitter Streaming API – [https://dev.twitter.com/streaming/overview](https://dev.twitter.com/streaming/overview)
\(^{11}\)Bounding Box – [http://wiki.openstreetmap.org/wiki/Bounding_Box](http://wiki.openstreetmap.org/wiki/Bounding_Box)
Spark Streaming generates a DStream, which is a high-level abstraction of the input data stream and a sequence of RDDs (Resilient Distributed Datasets). RDDs is a collection of elements that support parallel operations. One important parameter for RDDs is the number of partitions, which can cut the data into chunks and can be utilized to run the task for each partition of the cluster. Normally, Spark sets the default number of partitions automatically based on the cluster size, but we can also configure our own partitions based on the dataset to optimize the performance. We can also apply the transformations to perform customized operations on the input data stream in RDD.

Figure 3.2: Spark Streaming flow


Figure 3.2 presents the overall flow of Spark Streaming application. It takes the input data stream and divides it into batches of input data, which are then processed by the Spark engine to provide final output stream in batches.

### 3.2.1.2 Spark SQL

Spark SQL enables seamless integration of SQL queries with Spark programs to access a variety of data sources, including Parquet, JDBC, JSON, Avro, and Hive. Spark SQL scales to thousands of nodes when it is required to process or run queries against a huge amount of data. Data Frames, and Datasets are used to process and analyse the data in Spark SQL. A Data Frame is conceptually equivalent to a relational database with a distributed collection of data organized into named collections. It provides richer opti-
mizations under the hood. A Dataset is another format used to optimize the Spark SQL execution engine. Additionally, it also acts as a distributed query engine [16].

Spark SQL supports many languages such as Python, Java, R, and Scala. We have used the Python Spark SQL to process the historical Twitter and 511 data. We store the data in Parquet, making it easy to load into the Spark SQL environment. The input data from Parquet is directly loaded as Data Frame; then, we can apply any transformations to perform custom operations like we do on RDD. Finally, the results after performing transformations can be stored in any supported format such as Data Frame or Dataset.

### 3.2.2 Apache Parquet

Apache Parquet is a columnar database. Parquet is supported by any project in the Hadoop ecosystem, regardless of the choice of data processing framework. It supports very efficient compression and encoding schemes per column level [15]. Its compression schemes are promising, and when we stored 511 and Twitter historical data it showed significant reduction in space. In addition, it also improved performance when we were loading the data into the Spark SQL environment to apply transformations on data and run SQL queries against it.

### 3.3 Building the models

In this section, we discuss building the models to understand the speed and travel time dynamics in response to city traffic related events. We capture the non-linearity of speed and travel time observations by building models. It is personalized to spatiotemporal aspects (i.e., road link or location, day of the week, hour of speed and travel time observations). Location has a major influence on the traffic dynamics, e.g., busy links near shopping malls, and movie theaters often result in slow moving traffic. Hour of the day and

\[ \text{http://bit.ly/1iZ7mAU} \]
day of the week are two more important influencers which will affect the normal speed and travel time dynamics. We use location, hour of the day, and day of the week to partition the traffic dynamics and learn the normalcy model. We segmented the data based on the spatiotemporal values for each speed and travel time observation and we built the model for each segment.

As shown in Figure 3.3, there three main steps in building the normalcy models for sensor observations.

**Figure 3.3: Traffic Anomaly Detection from sensor observations.**

1. **Hash Partitioner:** In this step, we load all historical speed and travel time observations from the Apache Parquet database. It will create RDD and then we apply the transformation operation, *map*, which will generate a unique hash key value for each record in the RDD. Figure 3.4 shows the overview of hash key generation, which include the two columns linkid and timestamp utilized to generate the hashkey. To partition the speed and travel time observations based on their spatiotemporal values.
to create custom model for each cluster. We used the linkid, day of week, and hour of day to generate the hashkey which is used to partition the input data.

We perform the RDD partitionBy operation to group all the records with identical hash key values into one partition. Here, the number of partitions depends on the number of unique hash keys. Figure 3.5 shows sample records in default partitions and RDD after repartition with custom hash partitioner. There are four unique hash key values and then RDD is repartitioned into four partitions with all records with same hash key value is same partition for further processing in partition level.

2. **Medoid Computation:** Computing medoid for each partition in RDD, which contains sensor observations for a specific link, day of the week, and hour of day. We apply the mapPartitions, which is a transformation operation, to perform medoid computation on each partition in RDD. In this, we compute a mean for normalized data and then we find the Euclidean distance of each time series to the mean. Finally, a time series of observations that is closest to the mean is identified as medoid for
Figure 3.5: Sample RDD partitions after applying hash partitioner.
that partition.

3. **Model Creation:** In this step, we create a model for each partition capturing the location, day of the week, and hour of the day. It uses the historical speed and travel time observations in the partition to create the model. As discussed above, to create Kalman Filter [27] models, we utilize a Python library named Kalman Filter to learn normalcy dynamics. Kalman Filter [27] is an unsupervised algorithm. It does not need any labeled training data and is able to handle noisy observations. First, it creates the Data Frame using the speed and travel time observations. Second, performs the transformation operations on Data Frame and compute the normalcy values. Finally, we create a pickle file using these normalcy values and then store them in a database.

### 3.4 Event Extraction

In this section, we will discuss techniques for traffic event extraction. This work is motivated from Pramod et al.[34] to correlate the traffic anomalies detected from the sensor observations and events extracted from tweets in near real-time. First, we will detect anomalies in the average speed of vehicles passing through a road network reported in the form of sensor observations. Second, we will extract traffic events from city related tweets.

### 3.4.1 Traffic Anomaly Detection from Sensor Observations

Anomaly detection is performed on the speed observations collected from each link in the road network. Two step process for anomaly detection includes: 1) A data collection module, and 2) An anomaly detection module. Below, Figure 3.6 shows the architecture of the anomaly detection module for traffic anomaly detection from the speed observations.

The data collection module utilizes the 511.org traffic data feed API to retrieve the travel time and average speed observations and then writes it as a record with timestamp
Figure 3.6: Traffic Anomaly Detection from Sensor Observations.

and location to the socket. Ingesting data from the TCP socket to Spark Streaming enables the processing of live data streams. Spark Streaming creates data server which listens to a TCP socket and provides a high-level abstraction known as Dstream, which represents streaming data. Spark Streaming provides windowed operations, which allows us to apply transformations over a sliding window of input travel time and speed observations data. Figure 3.7 shows the window-based operations. Whenever the window slides over an original DStream, the original RDDs that were inside the window are grouped to produce an RDDS of a windowed DStream. In this particular case, the window operation is applied on the most recent 3 time units of data and slides by 2 time units. For this we need to specify two parameters:

1. **Window length**: The extent of the window. In Figure 3.7, it is three time units.

2. **Sliding interval**: The interval at which the window operation is applied. In Figure 3.7, it is two time units.
One additional parameter is batch interval, which is used to specify the interval in which to divide the input DStream into batches. We specify this batch interval contingent on system processing ability to process data as fast as it is being generating. The window length and sliding interval parameters must be multiples of the batch interval of the input source DStream [23]. In order to apply the anomaly detection transformation for hourly sensor data, we set the batch interval to 10 minutes and then set both the window and sliding interval to 6 batch intervals to process incoming sensor observations every one hour. Consequently, whenever the window slides over an input DStream, the input RDDs that fall within the window are grouped together to produce RDDs of the windowed DStream. Each window contains travel time and speed observations for an hour from the San Francisco Bay Area road network and anomaly detection applied to each link independently.

First, a hash partitioner is applied on RDD with one hour speed and travel time observations to divide it into partitions based on location. Here, we assign one unique hash for each partition based on the location. Second, anomaly detection is applied on each complete partition input travel time and speed observations. In this module, we compute speed dynamics for the partition as we discussed in the Building Models section. This one is compared against the existing normalcy model created from the historical data to check for
anomaly. If there is an anomaly, then the LOD creator creates triples using Traffic Event Ontology. We discuss this in more detail in the Semantic Integration section.

3.4.2 Traffic Events Extraction from Twitter

In this section, we discuss traffic event extraction from tweets. Tweet consists many features, such as text, longitude, latitude, and a timestamp. All these features can help us extract the event type, location, and time of the event being discussed in the tweets. To extract event-related information from tweets in near real-time, we have used Spark Streaming API. Figure 3.8 shows the event extraction pipeline, which includes the four main components: 1) tweet collection, 2) pre-processing, 3) event extraction, and 4) DB Handler.

Figure 3.8: Traffic Event Extraction from the Twitter Stream Pipeline.

1. **Tweet collection**: We utilize the Twitter Streaming API\(^\text{13}\) to retrieve the tweets us-

\(^{13}\)Twitter Streaming API – [https://dev.twitter.com/streaming/overview](https://dev.twitter.com/streaming/overview)
ing the location filter to get tweets from the SF Bay Area. We discussed Twitter Streaming API in more detail within the Spark Streaming Section 3.3.1.1. The Spark Streaming API generates a tweets DStream, which is a sequence of RDDs. Further, it takes only part of the input DStream for further processing using windowed operations in Spark Streaming. We discussed more details about DStream in Section 3.3.4.1. In Figure 3.6, window length is three time units and the sliding interval is two time units.

2. **Pre-processing:** In this step we pre-process the incoming tweet stream. Each input tweet is made up of a lot of features, but most of them are not required for event extraction. The most important features we used to extract events from the tweets such as text, latitude, longitude, created-at. It retrieves only the required features from the raw tweet in the tweet stream and saves them to the database as well as keeps them for further real-time processing to extract the events.

3. **Event extraction:** In this module we are using the state-of-art approach proposed by P. Anantharam et al [33], which utilizes the sequence labeling technique (Conditional Random Fields) to annotate tweets; Further, this is used to extract the events. An RDD map operation is used to apply the CRF model to extract events from the tweets.

4. **DB handler:** It will take the extracted events as input from the event extraction module and insert them into the MongoDB.
Semantic Integration

This section explains the Semantic Web standards and technologies used in the integration module of the thesis, which includes answering questions such as: 1) Which links have anomalies in sensor data and have events near link?, and 2) What are all corroborative events from Twitter and 511 incident reports within a specified radius? In order to answer these questions, we are utilize Semantic Web technologies to integrate multimodal data from diverse domains. It addresses the fact that there is no platform to integrate traffic events. We also discuss the infrastructure for publishing and retrieving the extracted traffic events relevant to answer specific questions listed above.

4.1 Semantic Web Standards

The use of Semantic Web technologies has allowed us to capture the meaning of real-world concepts and their relationships. This provides a rich abstraction of raw data (i.e., converting low level observations to human readable high level observations.) using the background knowledge relevant to the domain extending beyond syntactics. It plays an important role in integrating the heterogeneous information from multimodal data sources [51]. Part of this work involves applying the extended semantics which include ontologies and knowledge bases, in traditional IR ranking models to leverage the semantic search instead of keyword-based search [41]. We used the Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL (from the Semantic Web technology stack)
to integrate, and query and Linked Open Data (LOD) and to publish the extracted events from the sensor observations, social media, and web.

Figure 4.1: Semantic Web technology stack

Source: https://www.wikipedia.org/

1. **Resource Description Framework**: Resource Description Framework (RDF) is a general framework is used to integrate diverse data even when the underlying schemas are different. It is a simple model used to integrate, expose and share structured/semi-structured data. The idea is linking the resources on the Web using URIs to name the two ends of the link as well as the relationship between things is known as a "Triple" in RDF terminology [22]. It is a standard model for data interchange on the Web.

2. **Web Ontology Language (OWL)**: Web Ontology Language (OWL) is used to represent rich and complex relations between things, groups of things and knowledge
about things. It is used to create ontologies which are a formal representation of relationships between concepts of a specified domain knowledge [32]. The ontologies authored using OWL are used to represent information on the Web coming in from heterogeneous data sources [31].

3. **SPARQL**: The SPARQL Protocol and RDF Query Language (SPARQL) is a semantic query language which can insert, update, and query data stored in RDF format [24]. It can query data from diverse data sources such as data stored in RDF or viewed as RDF using middleware [26]. SPARQL terminology includes: 1) a SPARQL Protocol client that sends HTTP requests, 2) a SPARQL Protocol service that is a URI server and listens for requests, also known as SPARQL endpoint, 3) a RDF Dataset that is a default graph and zero or more named graphs. The SPARQL processing service uses HTTP to take requests from clients, execute them against the specific RDF Dataset mentioned in the query, and send results back to client, using response status codes to specify the success or failure of an operation defined in HTTP [25].

4. **Linked Open Data**: Wikipedia [20] defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.” Linked Open Data (LOD) which links related resources and provides a way to publish structured data that allows us to enrich and connect metadata so that we can find different representations of the same data. It has 4 important aspects: 1) it makes interoperability easy by publishing the data on web, 2) RDF is used to connect the published resources in a well-organized way, 3) URI is used to uniquely identify the interconnected resources, and 4) standard web resource protocol (HTTP) is used to put linked data online. Connecting the Open Data with other Open Data sources will make data rich, helping us to build applications based on this data so that resources can found easily and sharing information is much more complete [18].
4.2 Traffic Event Ontology

We propose Traffic Event Ontology as a formal model for traffic events and their metadata. This formal model enables semantic integration, interoperability and sharing it on web.

Table 4.1: Referenced Ontologies

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Event Ontology</td>
<td>event</td>
<td><a href="http://purl.org/NET/c4dm/event.owl#">http://purl.org/NET/c4dm/event.owl#</a></td>
</tr>
<tr>
<td>Basic Geo Positioning Ontology</td>
<td>geo</td>
<td><a href="http://www.w3.org/2003/01/geo/wgs84_pos#">http://www.w3.org/2003/01/geo/wgs84_pos#</a></td>
</tr>
<tr>
<td>OWL-Time</td>
<td>time</td>
<td><a href="http://www.w3.org/2006/time#">http://www.w3.org/2006/time#</a></td>
</tr>
</tbody>
</table>

The Traffic Event Ontology extends a well accepted Event Ontology which was developed in the Queen Mary University of London\(^1\). This ontology can model the Spatio-Temporal-Thematic (STT) semantics.

\(^1\)Queen Mary University – [http://www.qmul.ac.uk/](http://www.qmul.ac.uk/)
Various external standard ontologies are utilized such as W3C Basic Geo Positioning ontology [30] to capture event location and W3C OWL-Time ontology [28] to capture the event time. The above table shows list of ontologies with their respective prefix and namespace.

Events are described using the "What, Where and When" semantics [52]. Following Figure shows how event, time, and space regions as they describe with every event having two important properties: geo location and time.

![Traffic Event Ontology](image)

**Figure 4.3: Traffic Event Ontology;**

Details of the Traffic Event Ontology is shown in Figure 4.2. The Traffic Event sub-classes are chosen based on the traffic event type schema produced by 511.org\(^2\) with over 800 traffic event types. The traffic event types dataset is publicly available in a XML Schema Definition Language (XSD\(^3\)) file from 511.org. In this thesis we used the Apache


\(^3\)w3c – [https://www.w3.org/TR/xmlschema11-1/](https://www.w3.org/TR/xmlschema11-1/)
Jena API [14] to read hierarchy of traffic event types and subtypes and created the Traffic Event Ontology by importing the Event Ontology. The Traffic Event Ontology defines two data properties: impact and geohash, to align event output produced from different data sources [33].

![Part of OWL Traffic Event Ontology](image)

Figure 4.4: Part of OWL Traffic Event Ontology represented as follows

The Traffic Event ontology is based on the concepts of traffic events such as collision, incident, closure, and road work. It supports the description and high level traffic events for low level observations. Here observation refers to the data from the social media, sensors, and web and traffic events are defined as the abstraction of low level observations. One of the main focus of this thesis is on publishing these high level traffic events as RDF triples on web. For example, traffic event may refer to any event identified as causing the traffic congestion or increasing the travel time of part of the road network. The properties time, geo location, impact, and geohash better describes the traffic event.

### 4.3 Publishing Traffic Event Stream Infrastructure

In this thesis, we address the integration of heterogeneous event streams such as the events extracted from tweets, sensor data related to traffic, official incident reports from
511. We address the key challenge of building a processing and publishing infrastructure for traffic event stream on the web in near real-time as shown in Figure 4.5.

1. **LOD Creator**: It creates the RDF triples from the input traffic events using the Traffic Event Ontology and Semantic Web Standards discussed in 3.1 section. The traffic event streams aim to represent the features of the event stream which allow for the publishing of events on Web. It provides concepts such as geolocation (with latitude and longitude) and time (with various temporal concepts such as interval and instant). The traffic event streams use the broad definition of W3C Basic Geo Positioning ontology which helps to relate an event to some meaningful place (e.g. ”Berkeley”) or an event to multiple places (e.g. Illuminate SF Festival of light happens in multiple places such as North Beach, Embarcadero [19]). W3C OWL-Time ontology is used in order to express classification of a time region.

2. **Traffic Event LOD**: Linked Open Data (LOD) design principles are used in order to
make the extracted traffic event data shareable on Web [29]. A Traffic Event knowledge base was created which stores RDF triples and provides a SPARQL endpoint for search and discovery in accordance with LOD principles as well. The knowledge base is simply a populated ontology and the knowledge base is made public under the Creative Commons Attribution 4.0 International License [17], thereby adhering to the Open principle.

3. **Traffic Event Streams:** There are multiple event streams, which includes traffic events extracted from the travel time and speed sensor observations, tweets, and official events from the Web.

4. **Semantic Integration System Capability:** Currently it can semantically integrate three types of event streams such as traffic events from sensor observations, tweets, and Web. It can support as many data sources as desired by simply adding custom script for each data source to convert from its native format to RDF triples using the Traffic Event Ontology. This system also addresses the volume issue by adding a number of nodes to the computing cluster based on the resource availability.
Evaluation

We conducted a large-scale evaluation of our system on data collected from the traffic sensors and Twitter from the SF Bay Area over the course of a year. We focused on scalability and semantic integration of multimodal data.

5.1 Evaluation over Scalability

Learning model parameters and the criteria for anomalies is computationally expensive. We analysed the data of 2,534 road links. We created 168 models \(= 7(\text{days}) \times 24(\text{hours})\) for each link by analysing over 1.4 billion speed and travel time observations, which resulted in a total of 425,712 models \(= 168 \times 2,534\).

![Figure 5.1: Evaluation of serial and parallel implementations](image)
Figure 5.1 shows the evaluation of serial and parallel implementations. Initial evaluation was done using a 2.66 GHz, Intel Core 2 Duo with 8GB main memory. We estimated 25 minutes for learning model parameters and 15 minutes for computing the criteria of anomalies, resulting in an aggregated processing time of 40 minutes per link in the SF road network for one year worth of data. Hypothesizing processing time for all the links for one year worth of data would result in an estimated aggregated processing time of 1,689 hours \((\frac{40\text{minutes} \times 2,534}{60\text{minutes}})\approx 2\text{ months}\). A scalable and more faster technology was needed in order to process more data in less amount of time. Therefore, a scalable implementation of our evaluation approach was implemented on Apache Spark [58] cluster with 864 cores and 17 TB main memory which resulted in a processing time of less than a day. Our approach has been extensively tested with varying input data loads and configurations of the Apache Spark cluster with varying number of executors to evaluate the performance.

Figure 5.2: Evaluation of Scalable Processing with varying configurations

Figure 5.2 presents the learning model parameters and computing criteria for anomalies by configuring different numbers of executors in the Apache Spark cluster. Our evaluation shows that by increasing the number of executors the time taken for learning the model parameters and computing criteria for anomalies was significantly decreased. In our evaluation, it was clear that when increasing the number of executors more than 60 the in-
crease in the communication cost render that insignificant in terms of the processing time. Therefore, 60 executors were chosen as our default system configuration.

In addition to that, we evaluated our approach after increasing the input data by a factor of 10. Figure 5.3 presents the evaluation results for learning models parameters and computing criteria for anomalies after that increase in the amount of data and varying the number of executors for each run. As shown in the figure, our approach has a stable performance showing scalability even after increasing the input data to 10 times the original size. It also shows that increasing the number of executors, as shown before, does not actually provide any improvement to the performance of the system after a certain point, and that is why there is no significant decrease in processing time after using 300 executors.

![Figure 5.3: Evaluation of scalable processing after increasing the input data 10 times](image)

5.2 Evaluation over Semantic Integration

We evaluated our semantic integration approach by answering the questions discussed in the Introduction section of this thesis (Section I). We developed a user interface for evaluation to allow interacting with the event store which has events from the multi modal data.
Answer to Question 1:

For the first question, by using the user interface anyone can get insights and analyze the reasons causing the traffic congestions per link from the SF Bay area road network with data using question and link id. Figure 5.4 shows the results for the question: ”What are all the events that affect traffic on link 101010?” It shows the event categories on the left side of the webpage which caused the traffic congestion on link 101010 most frequently; on the right side of the map, the events are shown with details.

![Event Categories]

**Figure 5.4: Evaluation of question ”What are all the events that affect traffic on link 101010?”**

Answer to Question 2:

Figure 5.5 shows the evaluation of the semantic integration by asking one more additional question: i.e, ”What might be the reason for traffic delay on link 101010?”. The interface shows the recent accident reported from the official sources or from tweets on the link 101010 from the data we collected over a one year period.
Evaluation of our approach using the semantic integration is able to answer the questions related to anomalies detected from sensor observations and is able to show the context. In addition to that, city authorities can use this framework to get context for traffic congestion occurring on each road link in the SF Bay area with a detailed look at the data.

![Image](image.png)

**Figure 5.5:** Evaluation of question: "What might be the reason for traffic delay on link 101010?"
Future Work

In this thesis, we presented the scalable processing of historical and live data from 511 and Twitter using Spark and Parquet. In addition to that, we also presented the integration of diverse domain data sources using Semantic Web technologies. We plan to extend this thesis further by incorporating weather and parking sensor data sources. These additional input data sources will give us more context, in addition to existing traffic events extracted from Twitter and events reported to 511. With this, we can move forward to predict traffic in the future using the data from weather, parking, and scheduled events from Twitter and 511.

Further extension of the Traffic Event Ontology is desirable. Since we only used hierarchy of traffic event types from the 511, which has some areas of domain may not have been fully represented. Therefore, in future the ontology could benefit from broader research into the domain of traffic.

A challenge remaining for the future efforts is dynamically updating the models for detecting the traffic anomalies from the travel time and speed observations. In consideration of models developed on based on static data, those may not be accurate when there are significant changes in travel time and speed observations. Further, developing the way to update the models based on the new data added to the existing historical data from the input data streams.

In addition to that an extension of this work could be integrating this platform with existing social network platform like Twitris [8]. It deal with social data and provide spatio-
temporal-thematic (STT) analysis in near real-time. Additionally it also provides context based semantic integration of Web resources (e.g. Wikipedia, blogs, news, images, and videos), sentiment analysis, and tracking trends.
Conclusion

In a city, IoT and social media enables collection of Big Data. It require analysis of a large volume of sensor observations like weather, traffic flow, parking, and social media (i.e. citizen sensing). A wide range of systems and algorithms are built for near real-time processing of data. However, the challenge of detecting anomalies from a large volume of sensor observations using personalized models and semantically integrating diverse domains like numeric observations, unstructured text, and semi-structured is still unexplored with respect to the traffic domain.

This thesis focused on addressing these challenges. We developed a semantic-aware distributed framework which uses Spark and Parquet to build a distributed infrastructure. The framework focuses on scalable processing and storage of large volumes of physical sensor and social media data. This work also integrates data from multiple sources using the Traffic Event Ontology. Additionally, we built personalized models to capture the spatio-temporal influence on traffic. It semantically integrates and publishes the extracted events from the sensor observations, social media, and also events from the 511. Finally, it provides a user interface for end-users to run specific queries and get results related to traffic events.
Bibliography


[17] Creative Commons. https://creativecommons.org/licenses/by/4.0/, 2016.


[45] Danh Le-phuoc, Hoan Nguyen, Mau Quoc, and Josiane Xavier Parreira. The linked sensor middleware connecting the real world and the semantic web.


[56] Fangjin Yang, Gian Merlino, and Xavier Léauté. The radstack: Open source lambda architecture for interactive analytics.


