Journal of Big Data Analytics in Transportation (2019) 1:83–94
https://doi.org/10.1007/s42421-019-00006-8

ORIGINAL PAPER

A Cyberinfrastructure for Big Data Transportation Engineering

Md Johirul Islam[1] · **Anuj Sharma**[1] · **Hridesh Rajan**[1]

Received: 27 July 2018 / Revised: 4 April 2019 / Accepted: 9 April 2019 / Published online: 9 May 2019

Abstract
Big data-driven transportation engineering has the potential to improve utilization of road infrastructure, decrease traffic fatalities, improve fuel consumption, and decrease construction worker injuries, among others. Despite these benefits, research on big data-driven transportation engineering is difficult today due to the computational expertise required to get started. This work proposes BoaT, a transportation-specific programming language, and its big data infrastructure that is aimed at decreasing this barrier to entry. Our evaluation, that uses over two dozen research questions from six categories, shows that research is easier to realize as a BoaT computer program, an order of magnitude faster when this program is run, and exhibits 12–14× decrease in storage requirements.

Keywords Big data · Domain-specific language · Cyberinfrastructure

Introduction

The potential and challenges of leveraging big data in transportation has long been recognized (Adu-Gyamfi et al. 2017; Barai 2003; Chakraborty et al. 2017; Chen and Zhang 2014; Fan et al. 2014; Huang et al. 2016; Jagadish et al. 2014; Kitchin 2014; Laney 2001; Liu et al. 2016; Lv et al. 2015; Seedah et al. 2015; Wang et al. 2017; Zhang et al. 2011). For example, researchers have shown that big data-driven transportation engineering can help reduce congestions, fatalities, and make building transportation applications easier (Barai 2003; Huang et al. 2016; Zhang et al. 2011). The availability of open transportation data that are accessible, e.g. on the web under a permissive license, has the potential to further accelerate the impact of big data-driven transportation engineering.

Despite this incredible potential, harnessing big data in transportation for research remains difficult. To utilize big data, expertise is needed along each of the five steps of a typical data pipeline namely data acquisition; information extraction and cleaning; data integration, aggregation, and representation; modeling and analysis; and interpretation (Jagadish et al. 2014). First three steps are further complicated by the heterogeneity of data from multiple sources (Seedah et al. 2015), e.g. speed sensors, weather station, and national highway authority. A scientist must understand the peculiarities of the data sources to develop a data acquisition mechanism, clean data coming from multiple sources, and integrate data from multiple sources. Modeling and analysis are complicated by the volume of the data. For example, a dataset of speed measurements from a commercial provider for Iowa for a single day can be in multiple GBs, exceeding the limits of a single machine. Analyses that aim to compute trends over multiple years require storing, and computing over, tens of TBs of just speed sensor data.

A possible solution could be to use the big data technologies like Hadoop and Apache Spark running over a distributed cluster. Using a distributed cluster with an adequate number of nodes, problems related to the storage and time of computation can be addressed. But these big data technologies are not so easy to use. Getting started requires technical expertise to set up the infrastructure, efficient design of data schema, data acquisition strategy from multiple sources, high level of programming skills, adequate knowledge of distributed computing models, and a lot more efficiency in writing distributed computer programs which is significantly different than writing a sequential computer program in Matlab, C, or Java. The analysis of big data in transportation

✉ Md Johirul Islam
mislam@iastate.edu

Anuj Sharma
anujs@iastate.edu

Hridesh Rajan
hridesh@iastate.edu

[1] Iowa State University, Ames, IA, USA

is almost an elite job due to these barriers. The research groups interested in big data-driven transportation engineering have to hire technically skilled people or train their own staff members to use these highly sophisticated technologies. Both approaches incur additional costs.

This work describes a transportation-specific big data programming language and its infrastructure aimed at solving these problems. We call this language BoaT (Boa (Dyer et al. 2015) for Transportation). The BoaT infrastructure provides built-in transportation data schemas and converters from existing data sources. A notable advantage of BoaT's data schema is a significant reduction in storage requirements. A transportation researcher or engineer can express their queries as simple sequential-looking BoaT programs. The BoaT infrastructure automatically converts a BoaT program to a distributed executable code without sacrificing correctness in the conversion process. This also often results in an order of magnitude improvement in performance, which is the third advantage of our approach. The BoaT infrastructure provides built-in transportation data schemas and converters from the existing data sources. The four notable advantages of BoaT are (a) significant reduction in storage requirement using specially designed data schema, (b) a transportation researcher or engineer can express their queries as simple sequential-looking BoaT programs, (c) auto-conversion of sequential programs to parallelly executable programs without sacrificing correctness in the conversion process, and (d) the number of lines of code significantly reduces thus reducing the debugging time for the program. Owing to these advantages, even users that are not experts in distributed computing can write these BoaT programs that lower the aforementioned barrier to entry.

The remainder of this article describes the BoaT approach and explores its advantages. First, in the next section, we motivate the approach via a small example. Next, we compare and contrast this work with related ideas. Then, we describe the salient technical aspects of the technique. Next, we evaluate the usability, and scalability of the technique, show some use case examples that we have realized, and highlight the benefits of our storage strategy. Finally, we conclude.

Motivation

Transportation agencies collect a lot of data to make critical data-driven decision for Intelligent Transportation System (ITS). There has been a lot of initiative to make data available for researchers to spur innovation (US Department of Transportation 2017). But the analysis of this ultra-large-scale data is a difficult task; given the technology needed to analyze the data is still a luxury (Biuk-Aghai et al. 2016). These data come from multiple sources with a lot of varieties, velocities, and volumes. Given the availability of a variety of sources of data, technically skilled people also often face challenges due to the kinds of input, data access patterns, type of parallelism, etc. (Kambatla et al. 2014). The need to write complex programs can be a barrier for domain researchers to take the advantage of this large-scale data. To illustrate the challenges, consider a sample question "Which counties have highest and lowest average temperature in a day?" A query like this is simple when the data are already provided by county; but in case we have data for every 5 min for every square mile of Iowa for last 10 years, the query becomes hard to solve in Matlab or even R and could potentially run for a long time in Java. Answering this question in Java would require knowledge of (at a minimum) reading the weather data from the data provider service, finding the locations and county information of different grids from some other APIs, additional filtering code, controller logic, etc. It would need upwards of 100 lines of code and require knowledge of at least 2 complex libraries and 2 complex data structures. A heavily elided example of such a program is shown in Fig. 1, left column.

This program assumes that the user has manually downloaded the required weather data, preprocessed the data, and written to a CSV file. It then processes the data and collects weather information in different grids at different times of the day. Next, the county information of each grid is found from another API. Finally, the data are stored in some data structures for further computation. The presented program is sequential and will not scale as the data size grows. One could write a parallel computation program which would be even more complex.

We propose a domain-specific programming language called BoaT to solve these problems. We intend to lower the barrier to entry and enable the analysis of ultra-large-scale transportation data for answering more critical data-intensive research challenges. The main features of BoaT data analysis originated from Dean and Ghemawat (2008), Dyer et al. (2015), Pike et al. (2005), and Urso (2012). To this, we add built-in transportation-specific data types and functions for analysis of large-scale transportation data, schema, and infrastructure to preprocess data automatically and store efficiently. The main components come as an integrated framework that provides a domain-specific language for transportation data analysis, a data processing unit, and a storage strategy.

Related Work

Due to the rapid growth of data-driven Intelligent transportation system (ITS) (El Faouzi et al. 2011; Zheng 2015) applications and smart cities, the necessity of harnessing the power of ultra-large-scale data is becoming more important

Java

```
1     ... // imports
2
8     public class CountiesMaxAvgTmpc {
9     public static void main(String[] args){
10    ... // File operation
11    PriorityQueue<CountyTemp> maxheap , minheap;
12    ... // data processing
13
65    private String buildHeaps(/*input heaps*/) {
66    ... // Populate the heaps with Objects
67    }
68
72    ... // Iterating over the Map to find average temperature of
      each county
73    for (Map.Entry<String, List<Double>> entry : map.entrySet())
      {
74    String county = entry.getKey();
75    List<Double> countyTemps = entry.getValue();
76    for(double temp : countyTemps){
77    .../ Iterating over the county temp for avg
78    }
81
89    // Getting County name from another data file given the grid
      id
90    public static String getCounty(int gridid) {
91    ...    // Code to find county from grid
92    }
93    // Internal class to hold county data
94    private static class CountyTemp {
95    ... // County variables
96    public CountyTemp(String countyName, double temperature) {
97    ... // Constructor
98    }
99    ... // codes to manage county data
100   }
101   public static String getCounty(int gridid)  {
102   ... // Getting County from a  grid id from API
103   String county = getCountNameFromAPI(gridid);
104
```

BoaT

```
1     p: County = input;
2     max: output maximum(1) of string weight float;
3     min: output minimum(1) of string weight float;
4     count := 0;
5     sum := 0.0;
6     visit(p, visitor{
7     before n: Grid -> {
8     weatherRoot := getweather(n, "5-11-2017");
9     foreach(s : int; def(weatherRoot.weather[s])) {
10    sum = sum + weatherRoot.weather[s].tmpc;
11    count++;
12    }
13    }
14    });
15    max << p.countyName weight sum/count;
16    min <<  p.countyName weight sum/count;
17
```

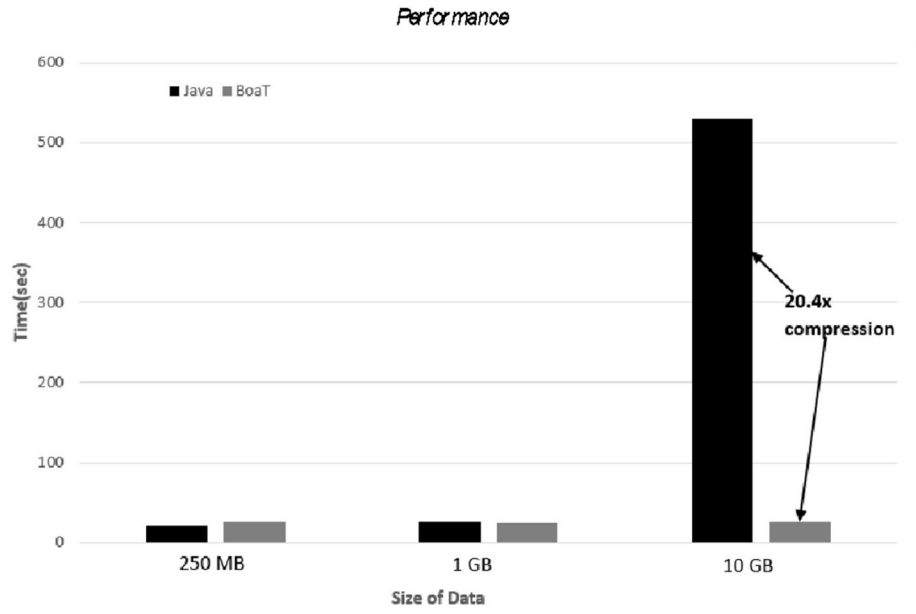


Fig. 1 Programs for answering "Which countries have highest and lowest average temperature in a day?" and the performance with the size of data

today than any time before. Though a lot of works are done on data-driven smart city design and big data analysis, trying to tackle the challenges of transportation big data from the domain-specific language perspective is few. In other domains, a lot of advantages are being taken from big data using domain-specific programming languages. For example, Dyer et al. (2015) used the early version of Boa to analyze ultra-large-scale software repository data (data from repositories like GitHub). However, Dyer et al.'s work is limited to software repositories whereas BoaT built on top of Boa provides the support of transportation data analysis at ultra-large scale, transportation domain types and an infrastructure of efficient data storage from a variety of transportation data sources.

There have been some efforts to support domain types and computation in transportation in an integrated modeling tool called UrbanSim (Borning et al. 2008a, b; Waddell et al. 2003). UrbanSim is an integrated modeling environment that provides a modeling language which provides access to urban data for finding models to coordinate transportation and land usage (Waddell et al. 2003). While UrbanSim focuses on simulation, BoaT is for analyzing gathered data. Furthermore, supporting analysis of large-scale data has not been the focus of UrbanSim, whereas BoaT focuses on providing scalable support for data analysis. Simmhan et al. (2013) provide a cloud-based software platform for data analytics in Smart Grids, whereas BoaT is focused on transportation data. Du et al. (2016) proposed City Traffic Data-as-a-Service (CTDaaS). They have used service-oriented architecture to provide access to data, but do not focus on the scalable analysis of big data.

In general, the current approaches using big data analytics are either using costly cloud computation or have custom build design for solving specific problems using open source solution with on-premise servers. Works such as Yang and Ma (2015) and Wang and Li (2016) highlight the challenges of doing big data-driven transportation engineering today. For example, Yang and Ma (2015) use HDFS, MLlib, and cluster computing to solve their problems, essentially like our motivating example. Each of these technologies creates its own barrier to entry. There is a need for a framework that would overcome the barrier to use big data analytics, provide a domain-specific language, reduce the efforts of data preprocessing, and will be available at a mass scale.

BoaT: Design and Implementation

To address the challenges of easy and efficient analysis of big transportation data, we propose a transportation-specific programming language and data infrastructure. The language provides simple syntax, domain-specific types, and massive abstractions. An overview of the infrastructure is shown in Fig. 2.

The user writes the BoaT program and submits it to the BoaT infrastructure. The BoaT program is taken by the infrastructure and converted by a specialized compiler that we have written to produce an executable that can be deployed in a distributed Hadoop cluster. This executable is run automatically on curated data to produce output for the user.

To illustrate, we consider the question in section "Motivation" "Which counties have the highest and the lowest average temperatures in a day?" A BoaT program to answer this question is shown in Fig. 1, right column. Line 1 of the program says that it takes a County as input. So, if there are n counties in the dataset, the statements on lines 4–16 of this program would be automatically run in parallel by the BoaT infrastructure (once for each county). Lines 2 and 3 of this program declare output variables. These write only output

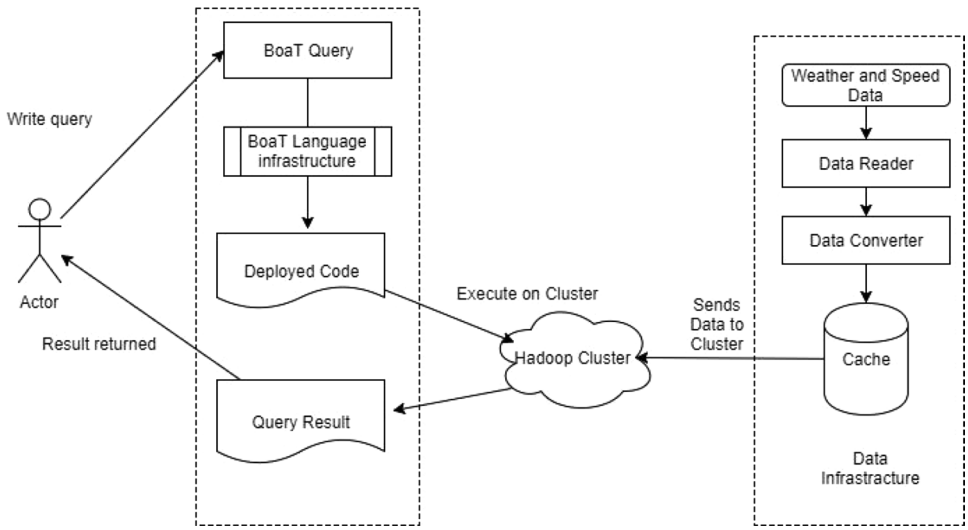


Fig. 2 An overview of BoaT, showing workflow of a BoaT user and BoaT infrastructure

Fig. 3 Domain types for transportation data in BoaT

Type	Attributes	Details
County	countyCode	Code of the county
	countyName	Name of the county
	Grids	List of Grid in the county.
Grid	ID	ID of a grid
	Location	Spatial location of the grid
	WeatherRoot	Link to the Weather data for the grid
	SpeedRoot	Link to the speed data for that grid
SpeedRoot	speedRecords	List of SpeedRecord
WeatherRoot	weatherRecords	List of WeatherRecord
SpeedRecord	detectorcode	The code of the detector giving the current record
	type	Type of the vehicle
	speed	Speed of the vehicle
	reference	Reference speed
	time	Time of the record
	roadname	Name of the road of the record
WeatherRecord	tmpc	2 m above the ground level temperature
	wawa	Watches, warnings, and advisories issued by the National Weather Service
	ptype	Type of Precipitation
	dwpc	Dew point temperature
	smps	Wind speed
	drct	Wind direction
	vsby	Horizontal visibility from sensors in Km
	roadtmpc	Pavement surface temperature
	srad	Solar radiation
	snwd	Snow fall depth
	pcpn	Precipitation accumulation
	time	Time of the reading

variables that are shared between all parallel tasks created by the BoaT infrastructure and the infrastructure manages the details of effectively interleaving and maximizing performance. Line 2 says that this output variable will collect values written to it and compute the maximum of those values. This is called aggregation in BoaT and several other kinds of aggregation algorithms are supported as shown in Fig. 4. Line 15 shows an example of writing to that output variable. Lines 4–16 are run sequentially for each county. They look into each grid of the county (lines 6, 7, 14) to find temperature data of the grid while maintaining a running sum and frequency to compute average on lines 15–16. While the details of this program are also important, astute readers would have surely observed that writing this program needed no knowledge of how the data are accessed, what is the schema of the data, and how to parallelize the program. No parallelization and synchronization code is needed. The BoaT program produces result running in a Hadoop cluster. So the program scales well saving hours of execution time.

As the program runs on a cluster, it outperforms the Java program (sequential) as the input data size grows. A comparison is shown in Fig. 1 on the lower right corner. The BoaT program provides output almost 20.4 times faster only on 1-day weather data of Iowa (10 GB). To achieve these goals, we have solved following problems:

- providing transportation domain types and functions;
- designing the schema for efficient storage strategy and parallelization; and
- providing an effective solution to data fusion.

Language Design

The language BoaT is the extended version of the work done by Dyer et al. (2015). They provide the syntax and tools to analyze the mining software repository data. We extended their work to provide domain types, functions, and computational infrastructure for big data-driven transportation engineering. We create the schema using the Google protocol buffer. Google protocol buffer is an efficient (Dyer et al. 2015) data representation format that provides faster memory efficient computation in BoaT.

Fig. 4 Aggregators in BoaT to reduce manual coding requirements for parallel computations

Aggregator	Description
MeanAggreagtor	Calculates the average
MaxAggreagtor	Finds the maximum value
QuantileAggregator	Calculates the quantile. An argument is passed to tell the quantile of interest
MinAggregator	Finds the minimum value
TopAggregator	Takes an integer argument and returns that number of top elements
StDevAggregator	Calculates the standard deviation

Domain Types

The transportation-specific types in BoaT are shown in Fig. 3. As we and others use this infrastructure, these types will surely evolve, and the BoaT infrastructure is designed to support such evolution. County is the top-level type. This type has attributes that relate to the code of the county, name of the county, and a list of grids in the county. A grid is related to a location in a county. For the convenience of computation, the whole Iowa is split into 213,840 Grids by Iowa DOT. So we used Grid as the domain type. The Grid has attributes such as ID, location (spatial location of the Grid), reference to the WeatherRoot which refers to the weather records in that Grid, and reference to SpeedRoot which refers to the speed records in that Grid. WeatherRoot contains WeatherRecords (a list of WeatherRecords). SpeedRoot contains SpeedRecords (a list of SpeedRecords). So we can easily go to the speed or weather data of a particular location in a particular Grid under a particular County without searching through all the data in the cluster. SpeedRecord contains the attributes like DetectorCode, type of detector, speed (average speed for a detector), reference (reference speed for a detector), roadname, and time.

The data design has led to two innovations:

- First to balance query speed, flexibility, and storage capacity.
- Second to allow future extension via data fusion (Fig. 4).

While designing the schema, we came to a successful data reduction strategy after multiple trials. Initially, we were using all the data at the top level. That means when we access a row we accessed all the relevant data for that row like weather and speed. Following this strategy, the storage size increased than the raw data. Then, we split the data keeping county data at the top level and the relevant weather and speed records at the second level in the same list. We were not getting enough mappers to make a lot of parallelization in the program as the splitting was not possible. And at the same time, storage size was almost near the raw data size. Then, we made multiple levels of hierarchy in our type system. The top level is the county. The county contains a list of grids (spatial locations), each grid contains two optional fields to point to speed data and weather data. This strategy of data representation gives us benefit in storage as well as in faster computation as only relevant data are accessed. We can store incremental data without regenerating the whole dataset from the beginning. Without this hierarchical schema strategy, all the data need to be merged together creating a merged schema hampering the sustainability, scalability, and storage benefit of the system. And the addition of new data would be impossible.

Fusion of multiple data sources in existing big data frameworks is difficult due to size, the necessity of join and parallel queries in the data sources. In BoaT, we addressed this problem in data infrastructure. Any new dataset can be added to the infrastructure easily. For example, we started with speed dataset initially and we were able to answer questions on speed data. The access link to speed data is optional. That means we do not load the data unless it is necessary. Then, we added another optional link to weather dataset. We came up with a successful fusion of data and were able to answer queries that cover both speed and weather dataset without losing any performance. The queries of category E in Fig. 5 are examples of using the fusion of weather and speed dataset. And the performance is not affected by this. This makes our infrastructure sustainable to any new datasets of interest to be added to the infrastructure. To do that, we have to just add an optional link to that new dataset after providing the schema for new dataset. The infrastructure will take care of all other complexities related to data generation and type generation.

Evaluation and Results

This section evaluates applicability, scalability, and storage efficiency of BoaT and its infrastructure. By applicability we mean whether a variety of transportation analytics use cases can be programmed using BoaT. By scalability, we mean whether the resulting BoaT programs scale when more resources are provided. By storage efficiency, we mean whether storage requirements for data are comparable to the raw data, or whether BoaT requires less storage, and if so how much.

Applicability

To support our claim of applicability, we use BoaT to answer queries on weather and speed data to provide answers to multiple queries from different categories and classes. A small BoaT program can answer queries that would need a lot of efforts with other general purpose languages, distributed system, and data processing. We provide a range of queries in six different categories and four different classes in the table shown in Fig. 5.

As an example scenario, we consider that a researcher wants to know the maximum and minimum temperature in different counties of a state in the USA in a date in May 2017. To achieve the result in the above scenario, we have to write a small program in Fig. 6. All the complex technical details of big data analytics are abstracted from the user. In Line 1, we are taking the data as input. In our BoaT infrastructure, we currently use county as the top-level entry point. In lines 2 and 3, we are declaring

Task	Classification	LOC			RTime (sec)		
		Java	BoaT	Diff	Java	BoaT	Speedup
A. Temperature Statistics							
1. Compute the mean, standard deviation of temperature	Central Tendency	84	10	8.4x	465	22	21.14x
2. Find the top ten counties with highest temperature	Rank	90	17	5.29x	470	20	23.50x
3. Find the top ten counties with lowest temperature	Rank	85	14	6.07x	489	25	19.56x
4. Find the highest and lowest temperature in different counties	Rank	70	8	8.75x	485	23	21.09x
5. Find the correlation between solar radiation and temperature	Correlation	65	10	6.50x	460	18	25.56x
6. Find the locations below a threshold temperature	Anomaly	69	11	6.27x	498	17	29.29x
B. Wind Behavior							
1. Compute the mean, standard deviation of wind speed	Central Tendency	97	12	8.083x	2753	48	57.35x
2. Find the top ten locations with higher wind speed	Rank	85	9	9.44x	2960	57	51.93x
3. Find the range across of wind different counties	Rank	65	12	5.42x	2793	45	62.07x
4. Find the correlation between temperature and wind speed	Correlation	90	15	6.00x	2743	43	63.79x
5. Find the locations above a threshold weather speed	Anomaly	65	10	6.50x	2894	47	61.57x
C. Study of precipitation behavior							
1. Compute the mean, standard deviation, quartile of precipitation	Central Tendency	65	17	3.82x	2883	51	56.53x
2. Find the top ten counties with high precipitation	Rank	35	10	3.50x	2945	46	64.02x
3. Find the correlation of temperature with precipitation	Correlation	80	9	8.89x	2980	45	66.22x
4. Find the correlation of visibility with precipitation	Correlation	80	8	10.00x	2401	53	45.30x
5. Find the locations where precipitation was above a threshold limit	Anomaly	72	6	12.00x	2180	48	45.42x
D. Study of Speed							
1. Compute the mean, standard deviation of speed at different locations	Central Tendency	100	17	5.88x	860	31	27.74x
2. Find the top ten counties with higher average speed	Rank	80	9	8.89x	815	25	32.60x
3. Find the road names with higher average Speed?	Rank	75	11	6.82x	810	27	30.00x
4. Find the county with maximum and minimum average speed	Rank	90	8	11.25x	986	35	28.17x
E. Effect of weather on speed							
1. Find the correlation between speed and precipitation	Correlation	150	13	11.54x	4230	130	32.54x
2. Find the correlation between speed and visibility	Correlation	150	13	11.54x	4213	132	31.92x
3. Find which weather parameter is more correlated with speed	Correlation	165	17	9.71x	4560	135	33.78x
F. Speeding Violations							
1. Find the number of vehicles running above a speed limit in different locations	Anomaly	80	12	6.67x	980	28	35.00x
2. What is the percentage of vehicles above reference speed at different locations?	Anomaly	95	15	6.33x	953	23	41.43x
3. Find the number of under speeding vehicles at different locations	Anomaly	80	12	6.67x	910	27	33.70x

Fig. 5 Examples of BoaT programs to compute different tasks on transportation data

```
p: County = input;
average : output mean[string] of int;
stdev : output stdev[string] of int;
visit(p, visitor {
before n: Grid -> {
speedRoot := getspeed(n,"5-11-2017");
foreach(s : int; def(speedRoot.speeds[s])) {
average[p.countyName] << speedRoot.speeds[s].speed;
stdev[p.countyName] << speedRoot.speeds[s].speed;
}
}
});
```

Fig. 6 Task A.4: find the highest and lowest temperature in different counties

```
p: County = input;
average : output mean[string] of int;
stdev : output stdev[string] of int;
visit(p, visitor {
before n: Grid -> {
speedRoot := getspeed(n,"5-11-2017");
foreach(s : int; def(speedRoot.speeds[s])) {
average[p.countyName] << speedRoot.speeds[s].speed;
stdev[p.countyName] << speedRoot.speeds[s].speed;
}
}
});
```

Fig. 8 Task D.1: compute the mean and standard deviation of speed at different locations

two output variables. The declaration tells clearly that one variable is going to store the maximum of some floating point numbers having a String, i.e. the county name as key and the other variable is going to store the minimum of some floating point numbers. The floating point numbers here are temperature found from the data. In the next line, there is a loop to iterate over all the grids of the county and for each county, we assign the temperature at that grid as weight. The program keeps track of the temperature values for each county and at the end returns maximum and minimum temperature at different counties in a day.

The output of the program is shown in Fig. 7. It also contains average temperature which is computed from the Task A.1.

To go through another example consider the Task D.1. Here, we calculate the mean and standard deviation of speed at different locations. The program is given in Fig. 8.

The program shown in Fig. 6 first declares the output types. The output variable for mean uses the MeanAggregator in BoaT and the output variable for standard deviation uses the StDevAggregator. The program iterates through each county one by one and all the grids in that county. While visiting a grid of the county, the program gets the speed data at that grid using a domain-specific function getspeed(). The function getspeed() has multiple versions and the version that we are using in this program takes the grid and a date as input and returns the speed data of that grid on that day. Then for each record of the speed data, we

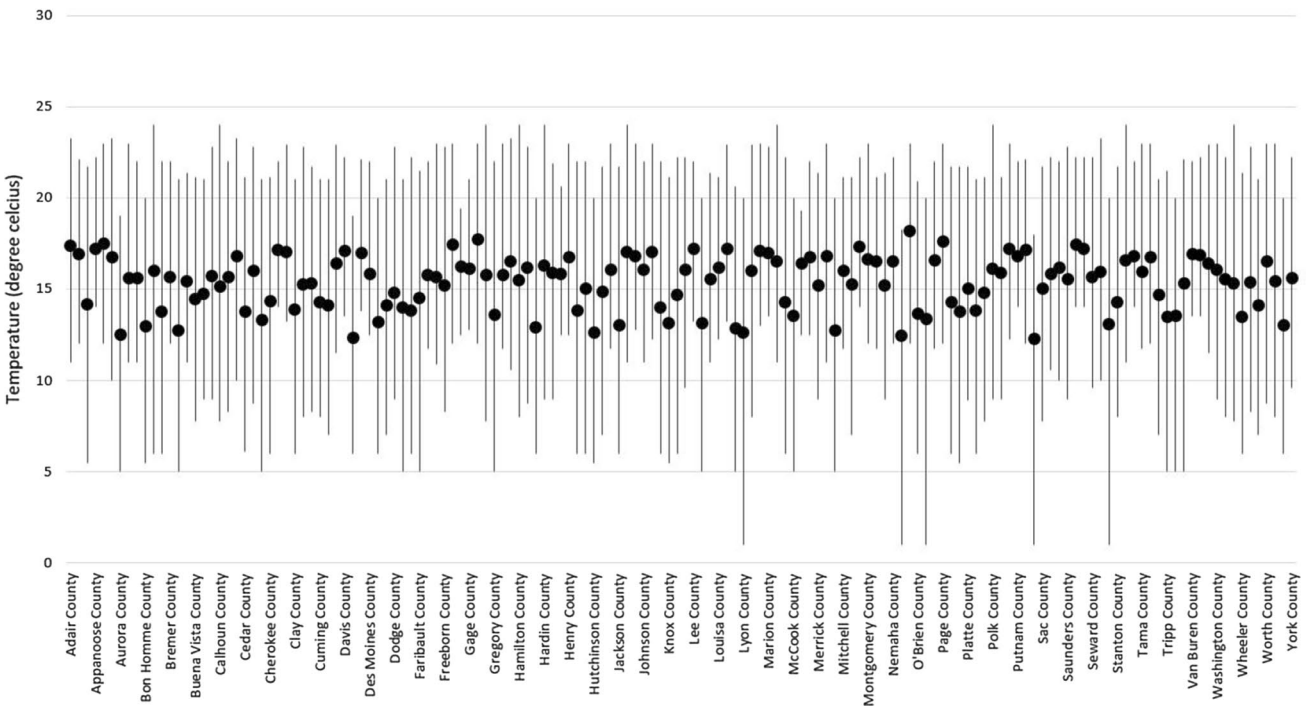


Fig. 7 Error bar graph of temperature showing minimum, maximum, and average temperature of different counties in a day. The result is produced from the code in Fig. 6 and average is found from Task A.1

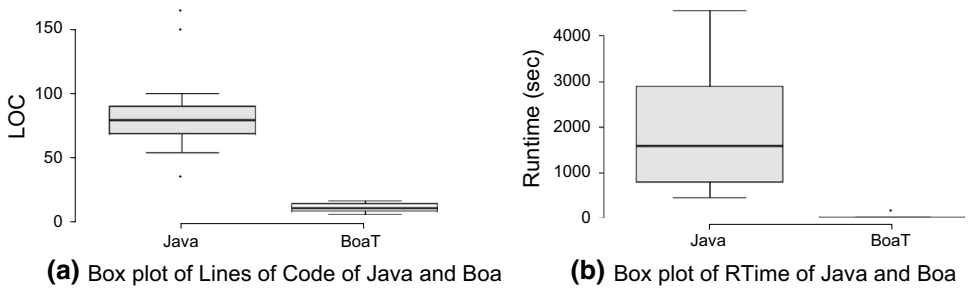


Fig. 9 Lines of code and run time comparison between Java and BoaT codes

(a) Box plot of Lines of Code of Java and Boa **(b)** Box plot of RTime of Java and Boa

aggregate the values in the output. These visits run in different mapper nodes and the aggregation is done in different reducer nodes. Finally, the result is returned to the user.

We use two metrics to evaluate BoaT's applicability.

- LOC: Line of Code. The total lines needed to write the program.
- RTime: Runtime of the program.

We show the comparison of these metrics for different programs in Fig. 5. The Java column shows the metric for Java program and the BoaT columns shows the values of the metrics for equivalent BoaT programs. The diff column shows how many times the BoaT program is efficient compared to Java in terms of Line of code. These Java programs are only for sequential operation. The Hadoop version of these programs can also be written, but that would require additional expertise and significantly larger lines of code.

BoaT can be adapted to any new transportation dataset. We presented the use of Speed and Weather data, which contains totally different schema. If we want to add new dataset, then we need to add the schema of the new dataset in our compiler using Google protocol buffer and the compiler automatically converts the schema to usable types in the language. Then, the user can write wrappers to convert raw dataset to BoaT dataset. In the overview shown in Fig. 2, the BoaT Language infrastructure and Data Reader components need to be updated to support new dataset. Then the user can write BoaT query on the

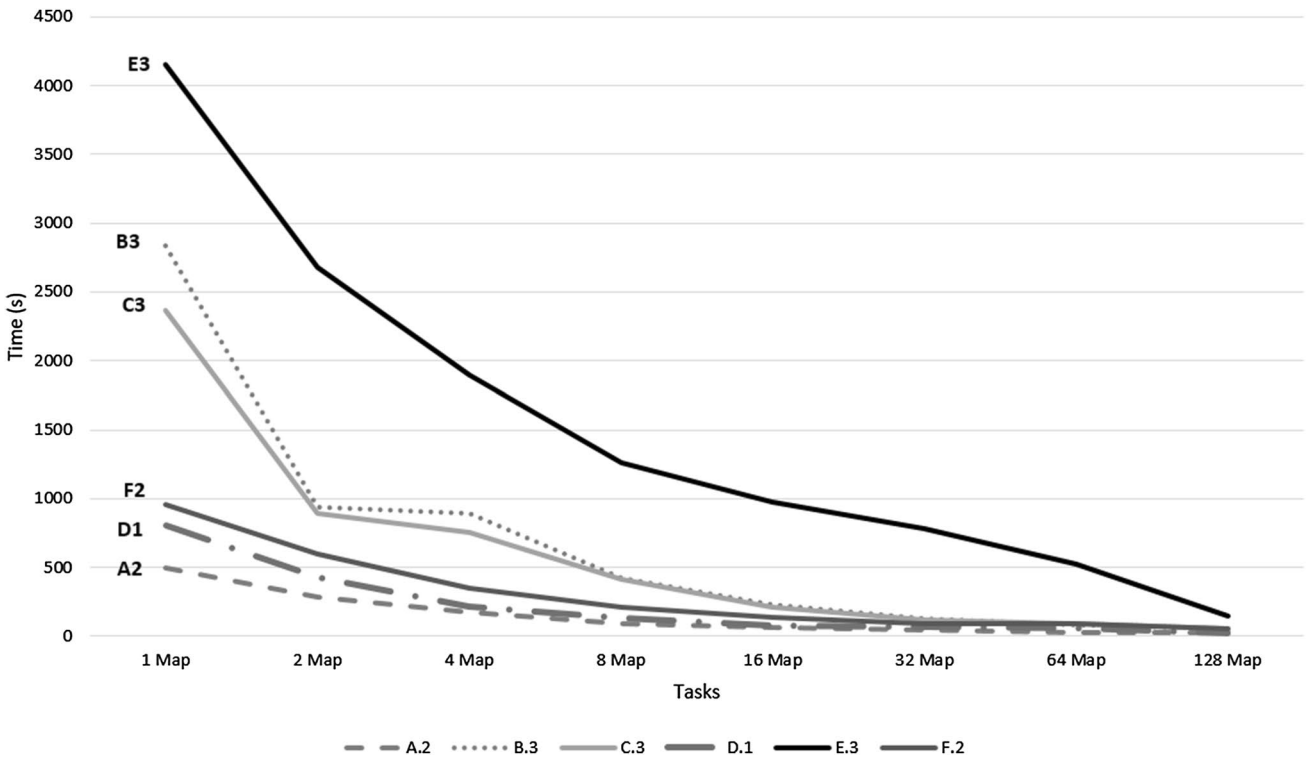
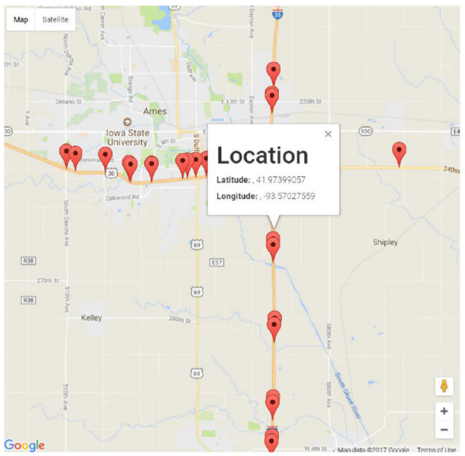
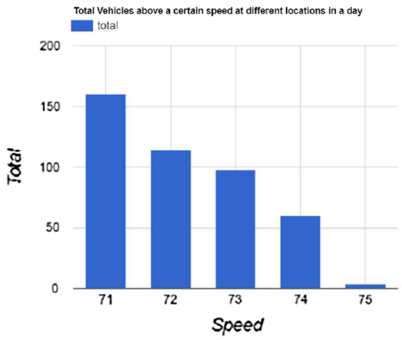
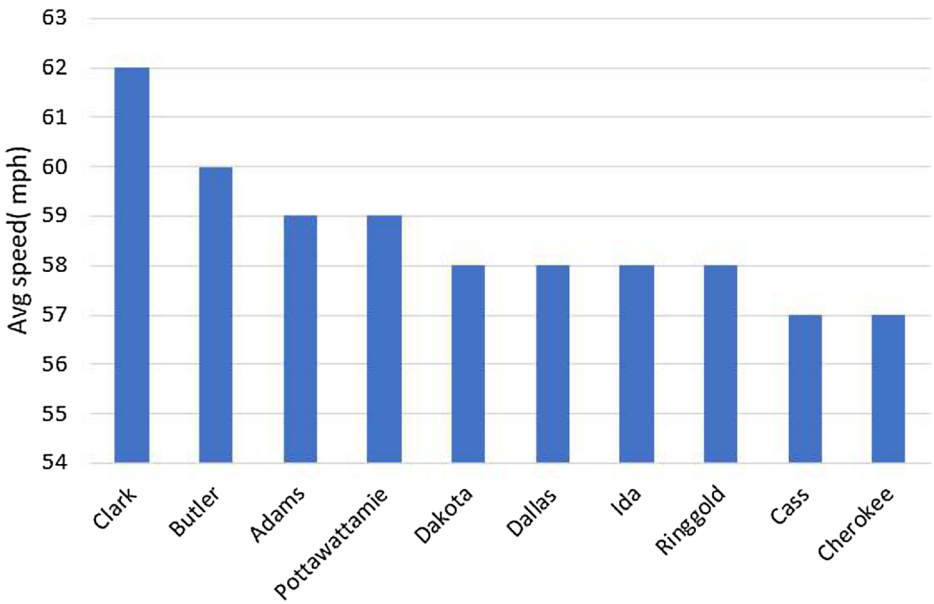


Fig. 10 Scalability of BoaT programs. The trends show that BoaT program is capable to effectively leverage the underlying infrastructure





(a) Markers show the location of different speeding incidents. Once a marker is clicked the chart on the right side shows the number of high-speed incidents categorized by speeds. This visualization is created from the result of the Task F.1



(b) Chart shows the counties with higher average speed on a day. This visualization is created from result of the Task D.2

Fig. 11 Visualization of Tasks F.1 and D.2

new dataset and use the domain-specific types automatically created by the compiler.

Scalability

Now, we evaluate the scalability of BoaT programs. The compiled BoaT program runs in a Hadoop cluster. BoaT provides all the advantages of parallel and distributed computation to the users that a Hadoop user would get (Fig. 9).

To evaluate scalability, we set up a Hadoop cluster with 23 nodes and with a capability of running 220 map tasks. We select one BoaT program from each category in Fig. 5. Then, we run the programs gradually increasing number of map tasks. The result of running the programs is shown in Fig. 10. The vertical axis represents the time in seconds. We see as the number of maps increases, the run time of the program decreases.

Example Dashboard Visualization

BoaT query results can be used to create interactive visualizations and dashboards. To support this claim, we present a few examples of simple visualizations.

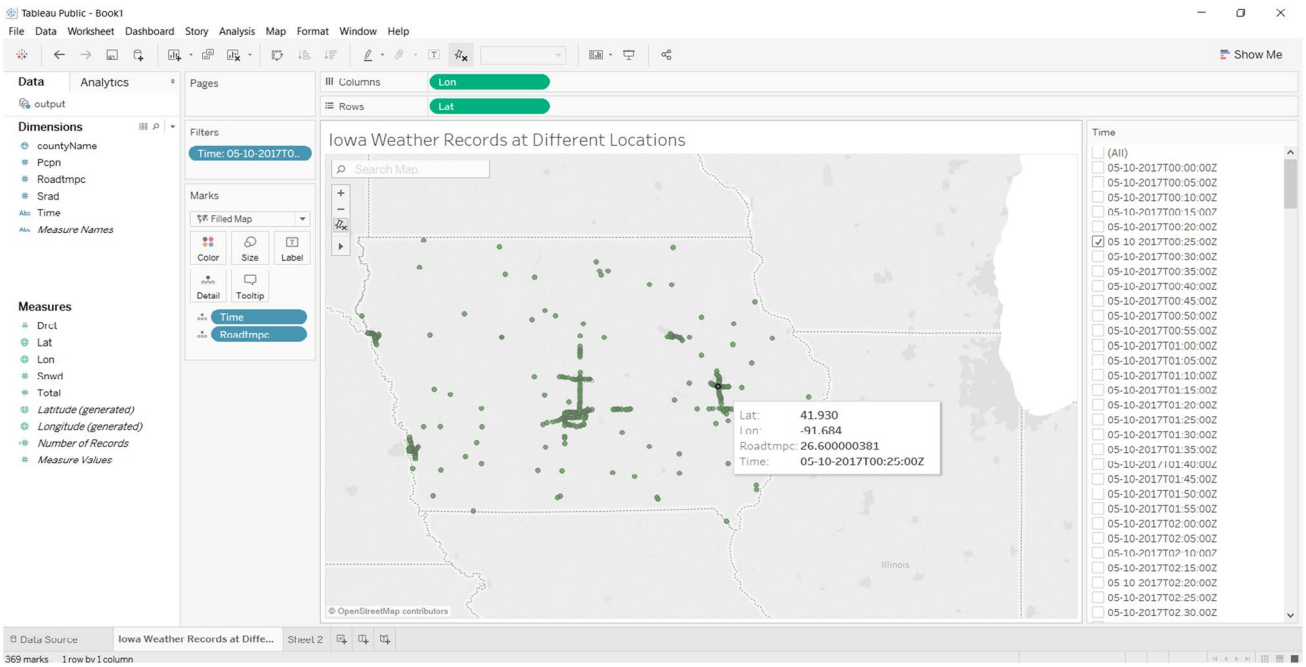


Fig. 12 This Tableau dashboard shows the road temperatures in degree Celsius at different times of the day at different locations. We can select the time from the time selector panel on the right. And once hovering the marker, we will be able to see the road temperature at that location at that time

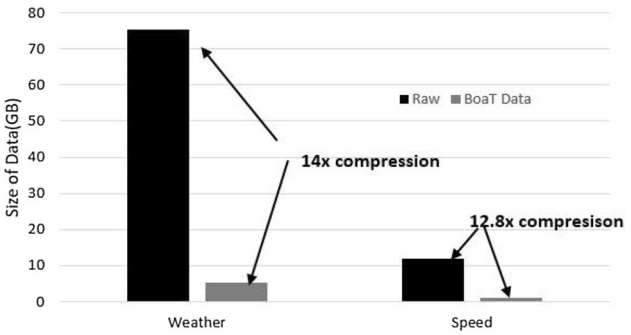


Fig. 13 Reduction in data storage size in BoaT data infrastructure compared to the raw data

We present the query result from Task F.1 in a simple dashboard created using JavaScript and Google Map in Fig. 11a. The markers show different speeding incident locations. Once a marker is clicked, then the chart on the right side shows the number of vehicles recorded above 70 mph at that location. For example, at location (41.97399057, −93.5702799) more than 150 vehicles were running at 71 mph on that day.

We provide another visualization of Task D.2 in Fig. 11b. In this task, we find out the top ten counties where the average speed was higher than other counties on that day. BoaT output can be easily imported into Tableau or other visualization software. To show an example of this, we visualize the result of Task A.5 in tableau in Fig. 12. DOTs and researchers who use visualization tools like tableau can directly benefit from the BoaT results.

Storage Efficiency

For evaluating the benefit, we compare raw data along with the data storage in BoaT. If we compress the raw data to reduce the size, we would lose the performance of the query; therefore, a compressed format is not desirable. But in BoaT, we can achieve the desired performance even after a huge reduction in the data size. The language reads the objects according to the domain type and emits the result from the Hadoop nodes to produce the final result. For comparison, we used the weather and speed data of 1 week for the state of Iowa. The weather data contains different weather information-related grids at different locations at 5-min interval. The speed data contains the readings from Inrix sensors at 20-s intervals. The preprocessed raw weather data size is 75.5 GB and the preprocessed raw speed data size 12.07 GB. We took these datasets to generate a model BoaT dataset. On top of the raw weather and speed data, we add a lot more other data like county names of grids, county code, county names where the speed detector is located, and road names of speed detectors. We collect some of this additional information from other metadata sources and some others using Google API. Even after adding a lot more additional data, our generated BoaT dataset size is much smaller than the original raw data. The original 75.5 GB speed dataset is reduced to

5.38 GB in BoaT and the original 12.07 GB speed dataset is reduced to 942 MB in BoaT as shown in Fig. 13.

Conclusion and Future Work

Big data-driven transportation engineering is ripe with potential to make a significant impact. However, it is hard to get started today. In this work, we have proposed BoaT, a transportation-specific big data programming language that is designed from the ground up to simplify expressing data analysis task by abstracting away the tricky details of data storage strategies, parallelization, data aggregation, etc. We showed the utility of our new approach, as well as its scalability advantages. Our future work will try out more application as well as create a web-based infrastructure so that others can also take advantage of BoaT's facilities.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grants CCF-15-18897 and CNS-15-13263. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Adu-Gyamfi YO, Sharma A, Knickerbocker S, Hawkins NR, Jackson M (2017) Framework for evaluating the reliability of wide-area probe data. Transp Res Rec 2643(1):93–104

Barai SK (2003) Data mining applications in transportation engineering. Transport 18(5):216–223

Biuk-Aghai RP, Kou WT, Fong S (2016) Big data analytics for transportation: problems and prospects for its application in china. In: IEEE Region 10 symposium (TENSYMP). IEEE, pp 173–178

Borning A, Ševcíková H, Waddell P (2008a) A domain-specific language for urban simulation variables. In: Proceedings of the 2008 international conference on digital government research. Digital Government Society of North America, pp 207–215

Borning A, Waddell P, Förster R (2008b) Urbansim: using simulation to inform public deliberation and decision-making. Digital government, pp 439–464

Chakraborty P, Hess JR, Sharma A, Knickerbocker S (2017) Outlier mining based traffic incident detection using big data analytics. Technical Report

Chen CP, Zhang CY (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347

Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Du B, Huang R, Chen X, Xie Z, Liang Y, Lv W, Ma J (2016) Active CTDaaS: a data service framework based on transparent IoD in city traffic. IEEE Trans Comput 65(12):3524–3536

Dyer R, Nguyen HA, Rajan H, Nguyen TN (2015) Boa: ultra-large-scale software repository and source-code mining. ACM Trans Softw Eng Methodol 25(1):7:1–7:34

El Faouzi NE, Leung H, Kurian A (2011) Data fusion in intelligent transportation systems: progress and challenges—a survey. Inf Fusion 12(1):4–10

Fan J, Han F, Liu H (2014) Challenges of big data analysis. Natl Sci Rev 1(2):293–314

Huang T, Wang S, Sharma A (2016) Leveraging high-resolution traffic data to understand the impacts of congestion on safety. In: 17th International conference road safety on five continents (RS5C 2016), Rio de Janeiro, 17–19 May 2016. Statens väg-och transportforskningsinstitut

Jagadish H, Gehrke J, Labrinidis A, Papakonstantinou Y, Patel JM, Ramakrishnan R, Shahabi C (2014) Big data and its technical challenges. Commun ACM 57(7):86–94

Kambatla K, Kollias G, Kumar V, Grama A (2014) Trends in big data analytics. J Parallel Distrib Comput 74(7):2561–2573

Kitchin R (2014) The real-time city? Big data and smart urbanism. GeoJournal 79(1):1–14

Laney D (2001) 3d data management: controlling data volume, velocity and variety. META Group Res Note 6:70

Liu C, Huang B, Zhao M, Sarkar S, Vaidya U, Sharma A (2016) Data driven exploration of traffic network system dynamics using high resolution probe data. In: IEEE 55th conference on decision and control (CDC). IEEE, pp 7629–7634

Lv Y, Duan Y, Kang W, Li Z, Wang FY (2015) Traffic flow prediction with big data: a deep learning approach. IEEE Trans Intell Transp Syst 16(2):865–873

Pike R, Dorward S, Griesemer R, Quinlan S (2005) Interpreting the data: parallel analysis with Sawzall. Sci Program 13(4):277–298

Seedah DP, Sankaran B, O'Brien WJ (2015) Approach to classifying freight data elements across multiple data sources. Transp Res Rec 2529:56–65

Simmhan Y, Aman S, Kumbhare A, Liu R, Stevens S, Zhou Q, Prasanna V (2013) Cloud-based software platform for big data analytics in smart grids. Comput Sci Eng 15(4):38–47

Urso A (2012) Sizzle: a compiler and runtime for Sawzall, optimized for Hadoop

US Department of Transportation: Data Inventory (2017). https://www.transportation.gov/data

Waddell P, Borning A, Noth M, Freier N, Becke M, Ulfarsson G (2003) Microsimulation of urban development and location choices: design and implementation of urbansim. Netw Spat Econ 3(1):43–67

Wang X, Li Z (2016) Traffic and transportation smart with cloud computing on big data. IJCSA 13(1):1–16

Wang S, Knickerbocker S, Sharma A (2017) Big-data-driven traffic surveillance system for work zone monitoring and decision supporting. Technical Report

Yang J, Ma J (2015) A big-data processing framework for uncertainties in transportation data. In: IEEE international conference on fuzzy systems (FUZZ-IEEE). IEEE, pp 1–6

Zhang J, Wang FY, Wang K, Lin WH, Xu X, Chen C (2011) Data-driven intelligent transportation systems: a survey. IEEE Trans Intell Transp Syst 12(4):1624–1639

Zheng Y (2015) Methodologies for cross-domain data fusion: an overview. IEEE Trans Big Data 1(1):16–34