2019

BoaT: A domain specific language and shared data science infrastructure for large scale transportation data analysis

Md Johirul Islam
Iowa State University

BoaT: A domain specific language and shared data science infrastructure for large scale transportation data analysis

by

Md Johirul Islam

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Hridesh Rajan, Major Professor
Gurpur Prabhu
Anuj Sharma

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

To my teachers, family and friends, who made me realize the real purpose of education.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. I would like to thank Dr. Hridesh Rajan for his guidance, patience and support throughout this research and the writing of this thesis. Thanks are due to the US National Science Foundation for financially supporting this project under grants CCF-15-18897 and CNS-15-13263. I would like to thank my committee members Dr. Gurpur M. Prabhu and Dr. Anuj Sharma for their efforts and contributions to this work. I would like to specially thank Dr. Anuj Sharma and the Reactor lab for providing me access to the dataset and guidelines to use the dataset for conducting this research. I would like to extend my thanks to all the members of Laboratory of Software Design for offering constructive criticism and timely suggestions during research. The majority of this draft is adopted from the Journal of Big Data Analytics in Transportation 2019 paper [1], which is written in collaboration with Dr. Anuj Sharma and Dr. Hridesh Rajan.

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 thesis proposes Boa$_T$, 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 questions are easier to realize as Boa$_T$ computer programs, an order of magnitude faster when these programs are run, and exhibit 12-14x decrease in storage requirements.

CHAPTER 1. INTRODUCTION

The potential and challenges of leveraging big data in transportation has long been recognized [2–15]. For example, researchers have shown that big data-driven transportation engineering can help reduce congestions, fatalities and make building transportation applications easier [5, 6, 14]. The availability of open transportation data that is accessible, for example, on the web under a permissive license has the potential to further accelerate the impact of big data-driven transportation engineering. The utility of shared data science infrastructure is also being recognized in other data intensive domains like bioinformatics [16].

Despite this incredible potential, harnessing big data in transportation for research remains difficult. In order 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 [2]. First three steps are further complicated by the heterogeneity of data from multiple sources [4], e.g. speed sensors, weather station, 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, 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 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 thesis proposes a transportation-specific big data programming language and its infrastructure aims at solving these problems. We call this language Boa$_T$ (Boa [17] for Transportation). The Boa$_T$ infrastructure provides built-in transportation data schemas and converters from existing data sources. A notable advantage of Boa$_T$'s data schema is a significant reduction in storage requirements. A transportation researcher or engineer can express their queries as simple sequential looking Boa$_T$ programs. The Boa$_T$ infrastructure automatically converts a Boa$_T$ 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 Boa$_T$ infrastructure provides built-in transportation data schemas and converters from existing data sources. The four notable advantages of Boa$_T$ are: a.) significant reduction in storage requirement by using specially designed data schema, b.) a transportation researcher or engineer can express their queries as simple sequential looking Boa$_T$ programs, c.) auto conversion of sequential programs to parallelly executable programs without sacrificing correctness in the conversion process, 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 Boa$_T$ programs that lower the aforementioned barrier to entry.

The remainder of this thesis describes the Boa$_T$ approach and explores its advantages. First, in the next chapter, 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 example use cases that we have realized, and highlight the benefits of our storage strategy. Finally, we conclude.

CHAPTER 2. MOTIVATION

Transportation agencies collect a lot of data to make critical data-driven decision for Intelligent Transportation System (ITS). There have been a number of initiatives to make data available for researchers to spur innovation [18]. 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 [19]. 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. [20]. 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 is already provided by county; but in case we have data for every 5 minute 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 Figure 2.1, 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 is 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.

Java

```
... // imports

public class CountiesMaxAvgTmpc {
public static void main(String[] args){
... // File operation
PriorityQueue<CountyTemp> maxheap , minheap;
... // data processing

private String buildHeaps(/*input heaps*/) {
... // Populate the heaps with Objects
}

... // Iterating over the Map to find average temperature of each county
for (Map.Entry<String, List<Double>> entry : map.entrySet()) {
String county = entry.getKey();
List<Double> countyTemps = entry.getValue();
for(double temp : countyTemps){
...// Iterating over the county temp for avg
}



// Getting County name from another data file given the grid id
public static String getCounty(int gridid) {
...    // Code to find county from grid
}
// Internal class to hold county data
private static class CountyTemp {
... // County variables
public CountyTemp(String countyName, double temperature) {
... // Constructor
}
... // codes to manage county data
}
public static String getCounty(int gridid)  {
... // Getting County from a  grid id from API
String county = getCountNameFromAPI(gridid);
```

Boa$_T$

```
p: County = input;
max: output maximum(1) of string weight float;
min: output minimum(1) of string weight float;
count := 0;
sum := 0.0;
visit(p, visitor{
before n: Grid -> {
weatherRoot := getweather(n,"5-11-2017");
foreach(s : int; def(weatherRoot.weather[s])) {
sum = sum + weatherRoot.weather[s].tmpc;
count++;
}
}
});
max << p.countyName weight sum/count;
min <<  p.countyName weight sum/count;
```

Performance

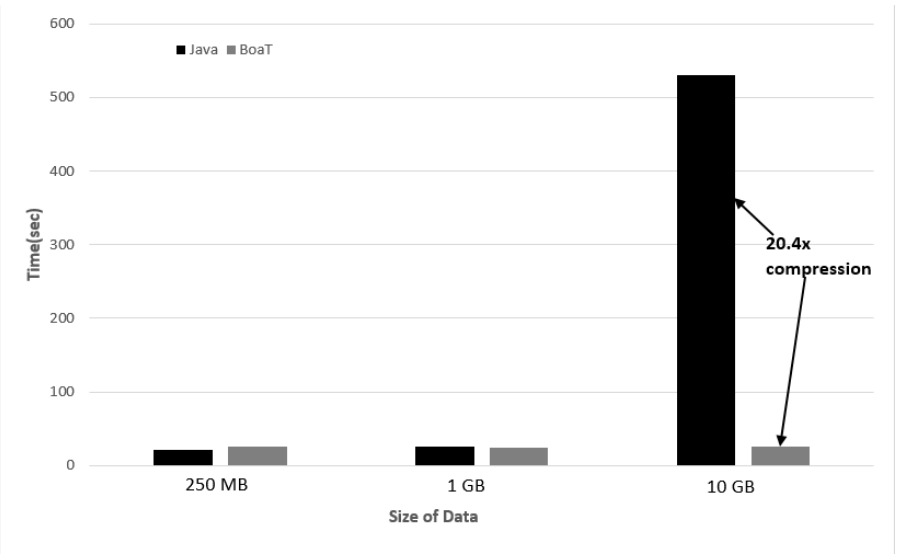


Figure 2.1 Programs for answering "Which counties have highest and lowest average temperature in a day?" and the performance with the size of data

Prior Work on Boa Infrastructure The Boa infrastructure enabling the analysis of ultra-large-scale software repositories was proposed first in 2013 [21]. Since then it has been well adopted in different applications of software analysis at large scale. The utility of Boa in the study of large scale software repositories was shown by [22], where the authors showed the utility of Boa in MSR (Mining Software Repositories) research and described the language and the infrastructure. Boa can be also used for task fusion to combine multiple tasks achieving 14-90% enhancement in the running time [23]. In the paper [24], the authors used Boa to show that mining tasks in MSR can be performed using 2-4X less code and less time. The authors also conducted a controlled study to show that, Boa programs are very easy to write compared to the complicated mining codes in prior works. The usage of Boa in mining millions of AST nodes to study the features of Java was presented in [25]. The authors have shown that Boa can mine the large scale AST nodes enabling large scale analysis of the source codes. The authors also demonstrated the distributed AST visitors in Boa. A large-scale study of substitutability of source codes has been conducted using Boa [26], where the authors studied 20 million Java classes to understand the effects of methods in subclasses and superclasses in practice. Boa has been also found effective in the detection of API misuse at scale by [27], where the authors have used Boa to mine Java code patterns and used those code patterns to detect the API misuses in Stack Overflow. Boa has been utilized by [28–30] towards accelerating mining and analysis of source code at scale. In enhancing source code traversal, a hybrid traversal strategy using Boa was proposed by [31]. Boa has been also utilized in the visualization of large scale mining results. [32–34]. The authors created a framework called Candoa. In the Candoia ecosystem, Boa has been used as the core component for generating large scale analysis results at large scale. Very recently, the utility of Boa has been demonstrated in collective program analysis [35]. The authors proposed a collective program analysis (CPA) technique for large scale source code that leverages analysis specific similarity. Analysis specific similarity tells whether two programs are similar in terms of analysis. They cluster programs based on analysis specific similarity. To determine the similarity and cluster the programs based on analysis specific similarity, they used a sparse representation and canonical labeling scheme. The authors were able to reduce the program analysis by 69% on average compared to baseline and 36% compared to prior work.

In this thesis, we propose a domain-specific programming language called Boa$_T$ built on top of Boa to solve the problems of large scale transportation data analysis. 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 Boa$_T$ data analysis are inspired from [17, 36–38]. 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.

CHAPTER 3. Boa$_T$: 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 Figure 3.1.

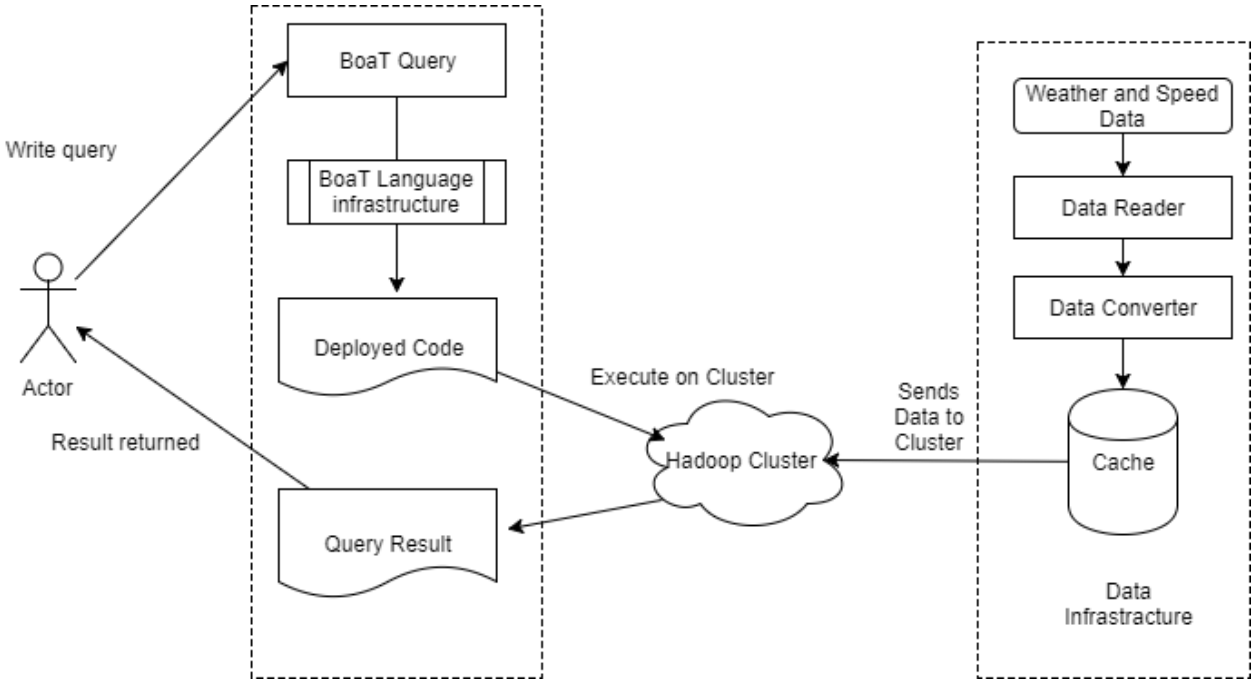


Figure 3.1 An Overview of Boa$_T$: shows workflow of a Boa$_T$ user and Boa$_T$ infrastructure

The user writes the Boa$_T$ program and submits it to the Boa$_T$ infrastructure, The Boa$_T$ 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, consider the question in Chapter 2 “Which counties have the highest and the lowest average temperatures in a day?”. A Boa$_T$ program to answer this question is shown in Figure 2.1, 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 Boa$_T$ infrastructure (once for each county). Line 2 and 3 of this program declares output variables. These *write only* output variables are shared between all parallel tasks created by the Boa$_T$ 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 Boa$_T$ and several other kinds of aggregation algorithms are supported as shown in Figure 3.3. 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 important also, astute readers would have surely observed that writing this program needed no knowledge of how the data is accessed, what is the schema of the data, how to parallelize the program. No parallelization and synchronization code is needed. The Boa$_T$ 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 Figure 2.1 on the lower right corner. The Boa$_T$ program provides output almost 20.4 times faster only on one-day weather data of Iowa (10GB). 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.

3.1 Language Design

The language Boa$_T$ is the extended version of the work done by [17]. 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 [17] data representation format that provides faster memory efficient computation in Boa$_T$.

3.2 Domain Types

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

Figure 3.2 Domain types for transportation data in Boa$_T$

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

Figure 3.3 Aggregators in Boa$_T$ to reduce manual coding requirements for parallel computations

The transportation-specific types in Boa$_T$ are shown in Figure 3.2. As we and others use this infrastructure these types will surely evolve, and the Boa$_T$ 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 213840 Grids by Iowa DOT. So, we used `Grid` as the domain type. The `Grid` has attributes `ID`, `Location` (spatial location of the `Grid`), reference to the `WeatherRoot` which refers to the weather records in that `Grid`, reference to `SpeedRoot` which refers to the speed records in that `Grid`. `WeatherRoot` contains `weatherRecords` (a list of `WeatherRecord`). `SpeedRoot` contains `speedRecords` (a list of `SpeedRecord`) . 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 `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.

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, 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, 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 hierarchy. 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 is 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 Boa$_T$, 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 don't 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 Figure 4.1 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.

CHAPTER 4. EVALUATION AND RESULTS

This chapter evaluates *applicability*, *scalability*, and *storage efficiency* of Boa$_T$ and its infrastructure. By applicability we mean whether a variety of transportation analytics use cases can be programmed using Boa$_T$. By scalability we mean whether the resulting Boa$_T$ 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 Boa$_T$ requires less storage, and if so how much.

4.1 Applicability

To support our claim of applicability we use Boa$_T$ to answer queries on weather and speed data to provide answers to multiple queries from different categories and classes. A small Boa$_T$ 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 table shown in Figure 4.1.

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 Figure 4.2. 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 Boa$_T$ infrastructure, we currently use county as the top level entry point. In Line 2 and Line 3 we are declaring 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.

Task	Classification	LOC			RTime (sec)		
		Java	Boa$_T$	Diff	Java	Boa$_T$	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

Figure 4.1 Examples of Boa$_T$ programs to compute different tasks on transportation data

```
p: County = input;
max: output maximum(1)[string] of string weight float;
min: output minimum(1) of string weight float;
foreach(i : int; def(p.grid[i])){
weatherRoot := getweather(p.grid[i],"5-11-2017");
foreach(j : int; def(weatherRoot.weather[j])){
max << p.countyName weight weatherRoot.weather[j].tmpc;
min << p.countyName weight weatherRoot.weather[j].tmpc;
}
}

```

Figure 4.2 Task A.4: Find the highest and lowest temperature in different counties

The output of the program is shown in Figure 4.3. It also contains average temperature which is computed from the task A.1.

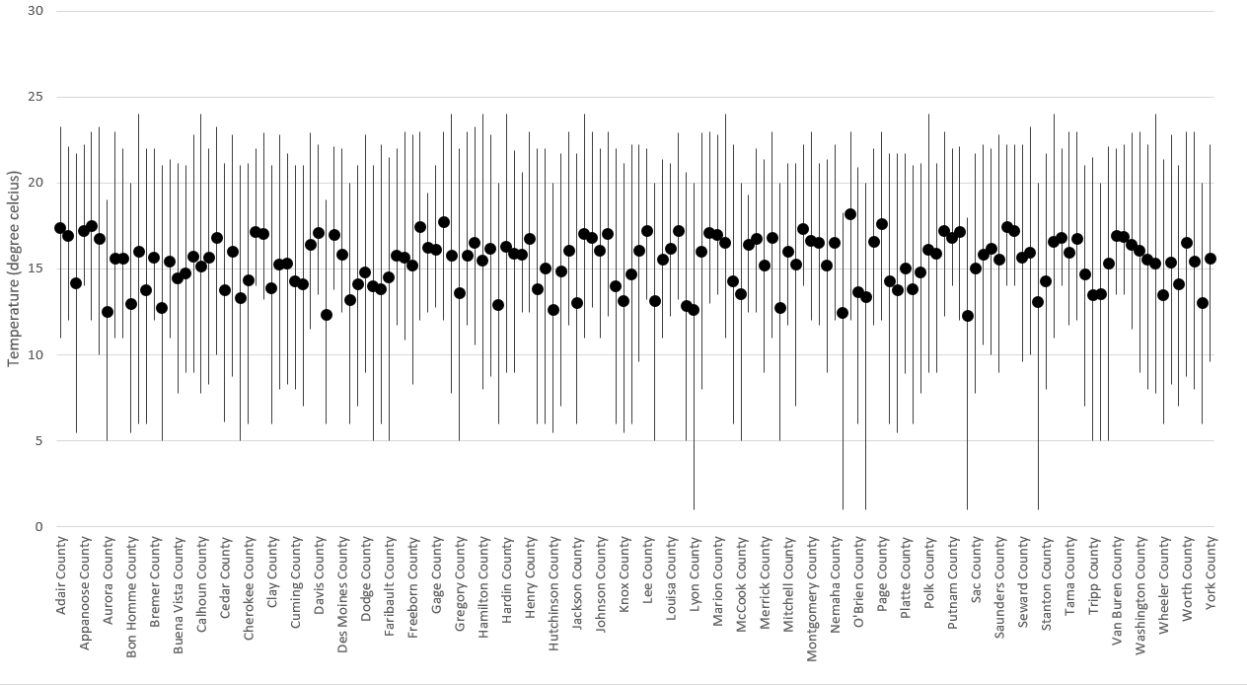


Figure 4.3 Error Bar graph of temperature showing minimum, maximum and average temperature of different counties in a day. The result is produced from the code in Figure 4.2 and average is found from task A.1

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 Figure 4.4.

```
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;
}
}
});

```

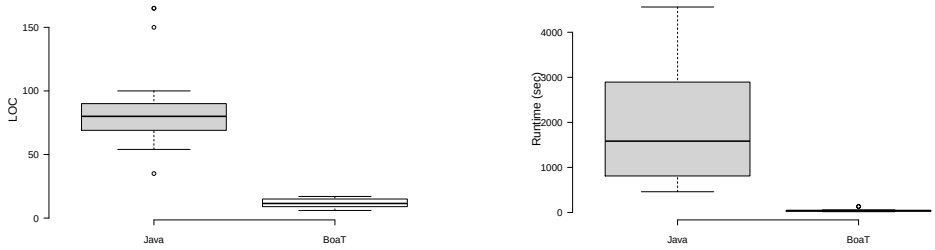
Figure 4.4 Task D.1: Compute the mean and standard deviation of speed at different locations

The program shown in Figure 4.2 first declares the output types. The output variable for mean uses the `MeanAggregator` in Boa$_T$ 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 by 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 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 Boa$_T$'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 Figure 4.1. The Java column shows the metric for Java program and the Boa$_T$ columns shows the values of the metrics for equivalent Boa$_T$ programs. The diff column shows how many times the Boa$_T$ 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.



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

Figure 4.5 Lines of Code and Run time comparison between Java and Boa$_T$ Codes

Boa$_T$ 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 Boa$_T$ dataset. In the overview shown in Figure 3.1, the **Boa$_T$ Language infrastructure** and **Data Reader** components need to be updated to support new dataset. Then the user can write Boa$_T$ query on the new dataset and use the domain specific types automatically created by the compiler.

4.2 Scalability

Now we evaluate the scalability of Boa$_T$ programs. The compiled Boa$_T$ program runs in a Hadoop cluster. Boa$_T$ provides all the advantages of parallel and distributed computation to the users that a Hadoop user would get.

To evaluate scalability we set up a Hadoop cluster with 23 nodes and with a capability of running 220 map tasks. We select one Boa$_T$ program from each category in Figure 4.1. Then we run the programs gradually increasing number of map tasks. The result of running the programs is shown in Figure 4.6. The vertical axis represents the time in seconds. We see as the number of maps increases the run time of the program decreases.

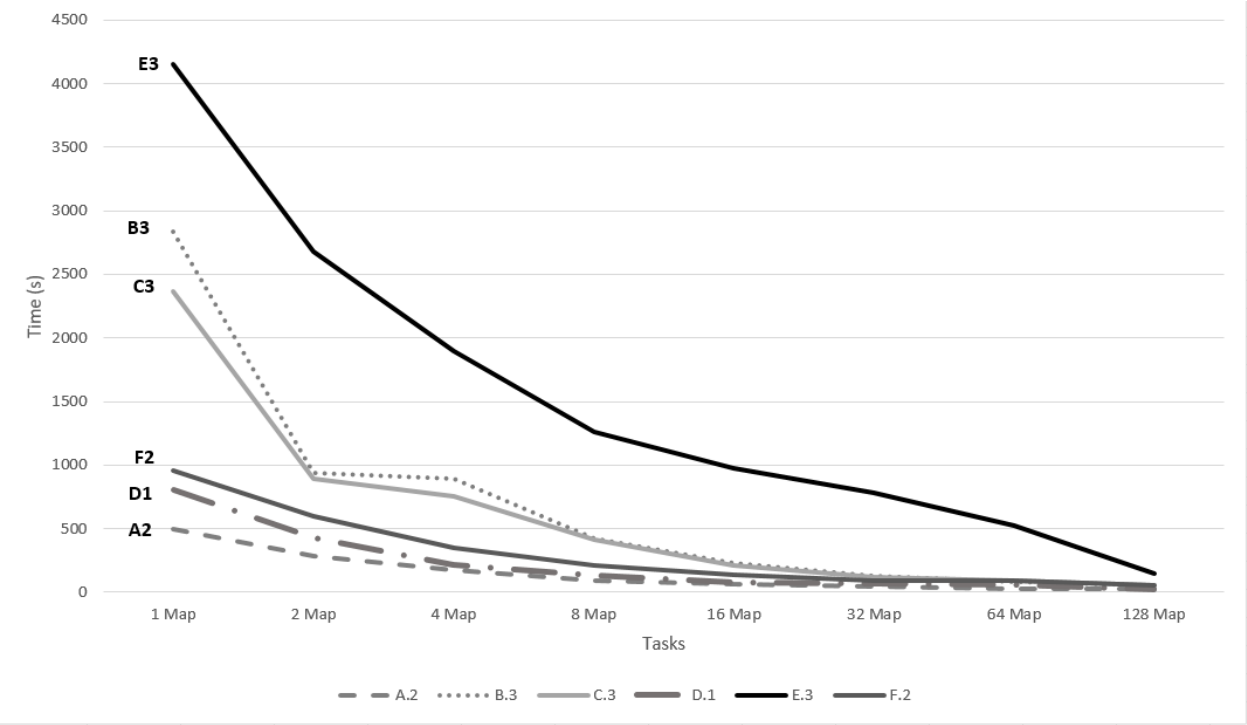


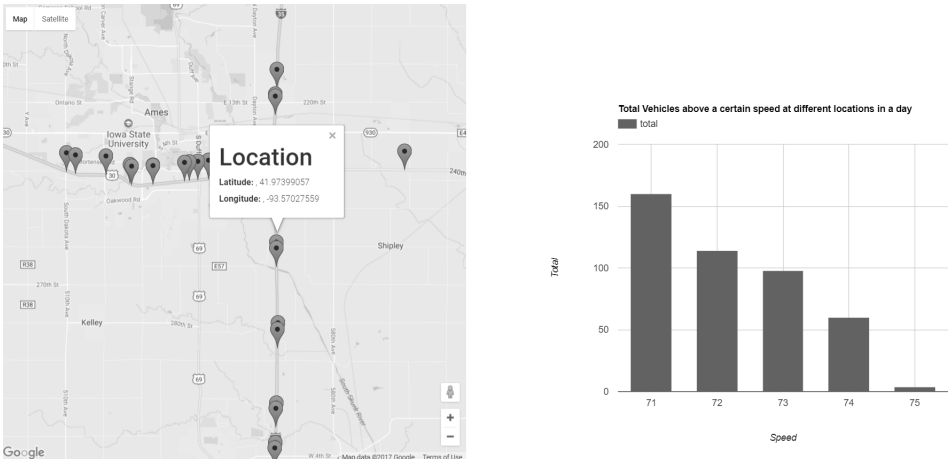
Figure 4.6 Scalability of Boa$_T$ programs. The trends show that Boa$_T$ program are able effectively leverage the underlying infrastructure.

4.3 Example Dashboard Visualization

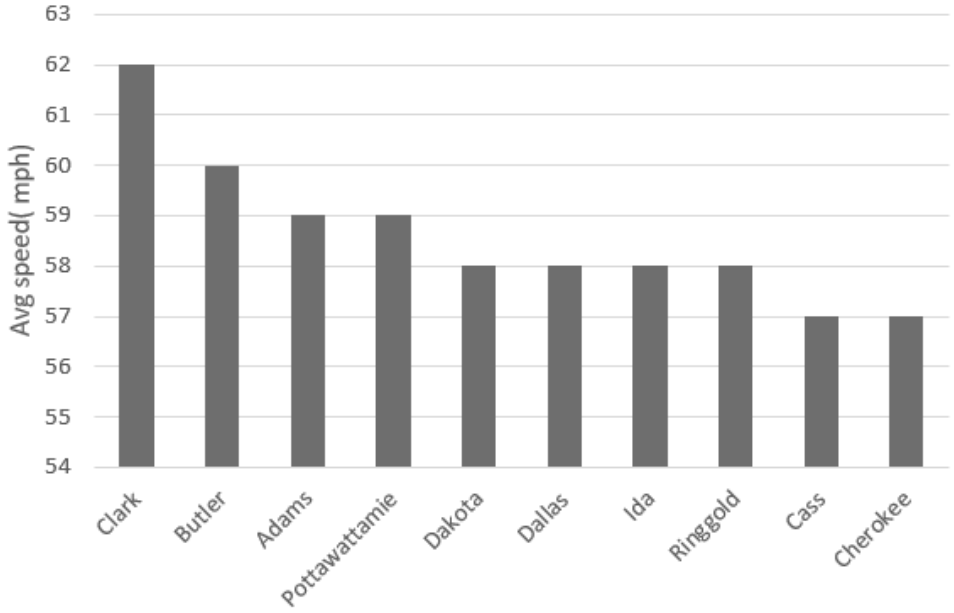
Boa$_T$ query results can be used to create interactive visualizations and dashboards. To support this claim we present a few examples of simple visualizations.

We present the query result from Task F.1 in a simple dashboard created using JavaScript and Google Map in Figure 4.7(a). 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 Figure 4.7(b). In this task, we find out the top ten counties where the average speed was higher than other counties on that day. Boa$_T$ output can be easily imported into Tableau or other visualization software. To show an example of this we visualize the result of



(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

Figure 4.7 Visualization of tasks F.1 and D.2

task A.5 in tableau in Figure 4.8. DOTs and researchers who use visualization tools like tableau can directly benefit from the Boa$_T$ results.

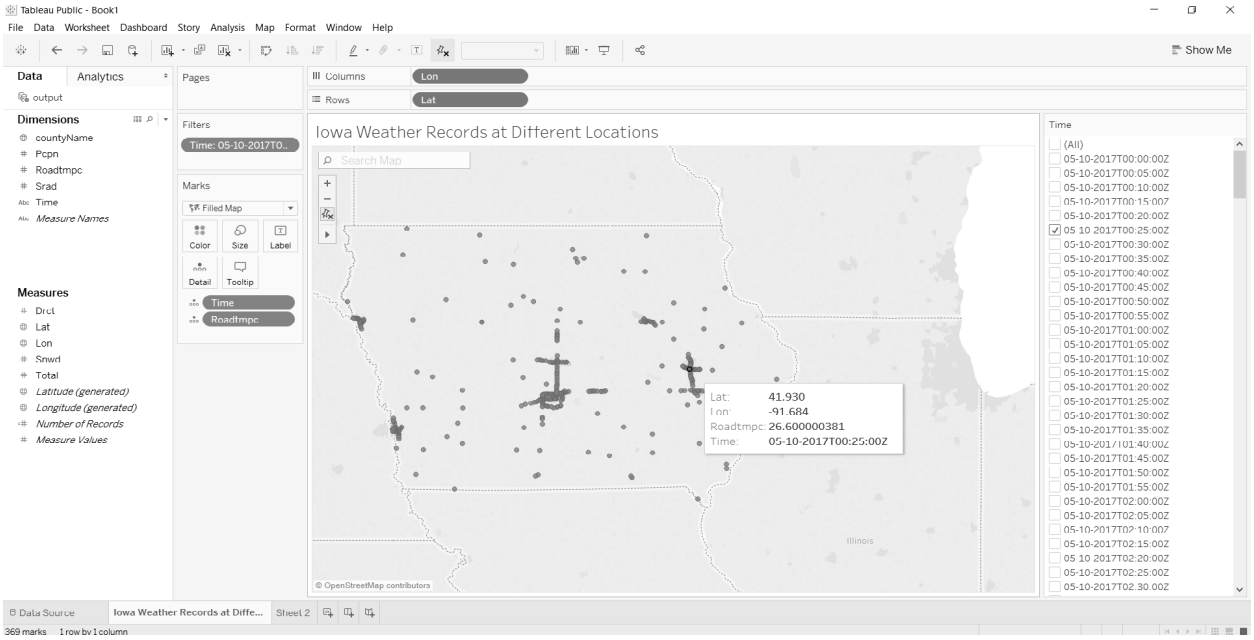


Figure 4.8 This Tableau dashboard shows the road temperatures in degree Celsius at different times of the day at different locations. We can select the time from the time selector panel on the right. And once hovering the marker we'll be able to see the road temperature at that location at that time.

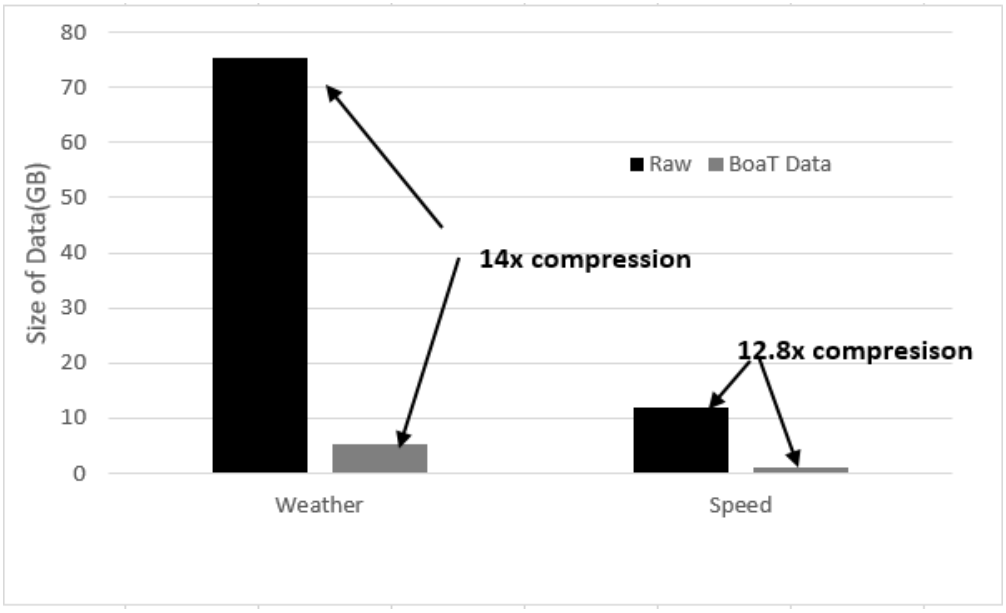


Figure 4.9 Reduction in data storage size in Boa$_T$ data infrastructure compared to the raw data

4.4 Storage Efficiency

For evaluating the benefit we compare raw data along with the data storage in Boa$_T$. 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 Boa$_T$, 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 one week for the state of Iowa. The weather data contains different weather information related grids at different locations at five minutes interval. The speed data contains the readings from Inrix sensors at 20 second intervals. The pre-processed raw weather data size 75.5 GB and the pre-processed raw speed data size is 12.07 GB. We took these datasets to generate an example Boa$_T$ 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, 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 Boa$_T$ dataset size is much smaller than the original raw data. The original 75.5 GB speed dataset is reduced to 5.38 GB in Boa$_T$ and the original 12.07 Gb speed dataset is reduced to 942 MB in Boa$_T$ as shown in Figure 4.9

CHAPTER 5. RELATED WORK

Due to the rapid growth of data-driven Intelligent transportation system (ITS) [39, 40] applications and smart cities the necessity of harnessing the power of ultra-large-scale data is becoming more important 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 dpecific programming languages. For example, [17] used the early version of Boa to analyze ultra-large-scale software repository data (data from repositories like GitHub). However, [17]'s work is limited to software repositories whereas Boa$_T$ 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 has been some efforts to support domain types and computation in transportation in an integrated modeling tool called UrbanSim [41–43]. 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 [43]. While UrbanSim focuses on simulation, Boa$_T$ is for analyzing gathered data. Furthermore, supporting analysis of large-scale data has not been the focus of UrbanSim, whereas Boa$_T$ focuses on providing scalable support for data analysis. [44] provides a cloud-based software platform for data analytics in Smart Grids, whereas Boa$_T$ is focussed on transportation data. [45] proposed City Traffic Data-as-a-Service (CTDaaS). They have used service-oriented architecture to provide access to data, but does 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 [46] and [47] highlight the challenges of doing big data-driven transportation engineering today. For example, [46] use HDFS, MLlib, 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.

CHAPTER 6. 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 thesis, we have proposed Boa$_T$, a transportation-specific big data programming language that is designed from the ground up to simplify expressing data analysis tasks 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 Boa$_T$'s facilities. In our current work, we only considered two datasets; however, in the future, we will show the utility of Boa$_T$ in adapting multiple datasets of the varied schema. The fusion of transportation data can be a potential future work of Boa$_T$. Currently, Boa$_T$ uses only the datasets of Iowa and the top member in the Schema is County. In the future, we would like to extend Boa$_T$ to support the federal level transportation data covering all the states. Data visualization is a crucial requirement of exploratory data analyses. So, we want to add data visualization support in Boa$_T$ to make Boa$_T$ a full-stack exploratory data analysis framework for Transportation engineers and analysts.

BIBLIOGRAPHY

[1] Md Johirul Islam, Anuj Sharma, and Hridesh Rajan. A cyberinfrastructure for big data transportation engineering. *Journal of Big Data Analytics in Transportation*, 1(1):83–94, 2019.

[2] HV Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[3] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2015.

[4] Dan PK Seedah, Bharathwaj Sankaran, and William J O'Brien. Approach to classifying freight data elements across multiple data sources. *Transportation Research Record: Journal of the Transportation Research Board*, (2529):56–65, 2015.

[5] Junping Zhang, Fei-Yue Wang, Kunfeng Wang, Wei-Hua Lin, Xin Xu, and Cheng Chen. Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1624–1639, 2011.

[6] Sudhir Kumar Barai. Data mining applications in transportation engineering. *Transport*, 18(5):216–223, 2003.

[7] Rob Kitchin. The real-time city? big data and smart urbanism. *GeoJournal*, 79(1):1–14, 2014.

[8] Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National science review*, 1(2):293–314, 2014.

[9] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

[10] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[11] Shuo Wang, Skylar Knickerbocker, and Anuj Sharma. Big-data-driven traffic surveillance system for work zone monitoring and decision supporting. Technical report, 2017.

[12] Pranamesh Chakraborty, Jacob Robert Hess, Anuj Sharma, and Skylar Knickerbocker. Outlier mining based traffic incident detection using big data analytics. Technical report, 2017.

[13] Chao Liu, Bowen Huang, Mo Zhao, Soumik Sarkar, Umesh Vaidya, and Anuj Sharma. Data driven exploration of traffic network system dynamics using high resolution probe data. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 7629–7634. IEEE, 2016.

[14] Tingting Huang, Shuo Wang, and Anuj Sharma. 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, Brazil, 17-19 May 2016*. Statens väg-och transportforskningsinstitut, 2016.

[15] Yaw Okyere Adu-Gyamfi, Anuj Sharma, Skylar Knickerbocker, Neal R Hawkins, and Michael Jackson. A framework for evaluating the reliability of wide area probe data. Technical report, 2017.

[16] Hamid Bagheri, Usha Muppirala, Rick E Masonbrink, Andrew J Severin, and Hridesh Rajan. Shared data science infrastructure for genomics data. *BMC bioinformatics*, 20(1):436, 2019.

[17] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: Ultra-large-scale software repository and source-code mining. *ACM Trans. Softw. Eng. Methodol.*, 25(1):7:1–7:34, December 2015.

[18] US Department of Transportation. Data Inventory, 2017. https://www.transportation.gov/data.

[19] Robert P Biuk-Aghai, Weng Tat Kou, and Simon Fong. Big data analytics for transportation: Problems and prospects for its application in china. In *Region 10 Symposium (TENSYMP), 2016 IEEE*, pages 173–178. IEEE, 2016.

[20] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561–2573, 2014.

[21] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering*, ICSE'13, pages 422–431, May 2013.

[22] Robert Dyer, Hoan Nguyen, Hridesh Rajan, and Tien Nguyen. Boa: An enabling language and infrastructure for ultra-large-scale msr studies. In *The Art and Science of Analyzing Software Data*, pages 593–621. Elsevier, 2015.

[23] Robert Dyer. Task fusion: Improving utilization of multi-user clusters. In *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pages 117–118. Citeseer, 2013.

[24] Robert Dyer, Hridesh Rajan, and Tien N Nguyen. Declarative visitors to ease fine-grained source code mining with full history on billions of ast nodes. *ACM SIGPLAN Notices*, 49(3):23–32, 2014.

[25] Robert Dyer, Hridesh Rajan, Hoan Anh Nguyen, and Tien N Nguyen. Mining billions of AST nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering*, pages 779–790. ACM, 2014.

[26] Jackson Maddox, Yuheng Long, and Hridesh Rajan. Large-scale study of substitutability in the presence of effects. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 528–538. ACM, 2018.

[27] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896. IEEE, 2018.

[28] Ganesha Upadhyaya and Hridesh Rajan. On accelerating source code analysis at massive scale. *IEEE Transactions on Software Engineering*, 44(7):669–688, 2018.

[29] Ganesha Upadhyaya and Hridesh Rajan. On accelerating ultra-large-scale mining. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, pages 39–42. IEEE Press, 2017.

[30] Sumon Biswas, Md Johirul Islam, Yijia Huang, and Hridesh Rajan. Boa meets python: a boa dataset of data science software in python language. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 577–581. IEEE Press, 2019.

[31] Ramanathan Ramu, Ganesha Upadhyaya, Hoan A Nguyen, and Hridesh Rajan. Hybrid traversal: efficient source code analysis at scale. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 412–413. ACM, 2018.

[32] Nitin M Tiwari, Dalton D Mills, Ganesha Upadhyaya, Eric Lin, and Hridesh Rajan. Candoia: A platform and an ecosystem for building and deploying versatile mining software repositories tools. 2015.

[33] Nitin M Tiwari, Ganesha Upadhyaya, and Hridesh Rajan. Candoia: A platform and ecosystem for mining software repositories tools. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 759–764. ACM, 2016.

[34] Nitin M Tiwari, Ganesha Upadhyaya, Hoan Anh Nguyen, and Hridesh Rajan. Candoia: A platform for building and sharing mining software repositories tools as apps. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 53–63. IEEE, 2017.

[35] Ganesha Upadhyaya and Hridesh Rajan. Collective program analysis. In *Proceedings of the 40th International Conference on Software Engineering*, pages 620–631. ACM, 2018.

[36] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[37] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[38] Anthony Urso. Sizzle: A compiler and runtime for Sawzall, optimized for hadoop, 2012.

[39] Nour-Eddin El Faouzi, Henry Leung, and Ajeesh Kurian. Data fusion in intelligent transportation systems: Progress and challenges–a survey. *Information Fusion*, 12(1):4–10, 2011.

[40] Yu Zheng. Methodologies for cross-domain data fusion: An overview. *IEEE transactions on big data*, 1(1):16–34, 2015.

[41] Alan Borning, Hana Ševčíková, and Paul Waddell. A domain-specific language for urban simulation variables. In *Proceedings of the 2008 international conference on Digital government research*, pages 207–215. Digital Government Society of North America, 2008.

[42] Alan Borning, Paul Waddell, and Ruth Förster. Urbansim: Using simulation to inform public deliberation and decision-making. *Digital government*, pages 439–464, 2008.

[43] Paul Waddell, Alan Borning, Michael Noth, Nathan Freier, Michael Becke, and Gudmundur Ulfarsson. Microsimulation of urban development and location choices: Design and implementation of UrbanSim. *Networks and spatial economics*, 3(1):43–67, 2003.

[44] Yogesh Simmhan, Saima Aman, Alok Kumbhare, Rongyang Liu, Sam Stevens, Qunzhi Zhou, and Viktor Prasanna. Cloud-based software platform for big data analytics in smart grids. *Computing in Science & Engineering*, 15(4):38–47, 2013.

[45] Bowen Du, Runhe Huang, Xi Chen, Zhipu Xie, Ye Liang, Weifeng Lv, and Jianhua Ma. Active ctdaas: A data service framework based on transparent iod in city traffic. *IEEE Transactions on Computers*, 65(12):3524–3536, 2016.

[46] Jie Yang and Jun Ma. A big-data processing framework for uncertainties in transportation data. In *Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015.

[47] Xiaoxia Wang and Zhanqiang Li. Traffic and transportation smart with cloud computing on big data. *IJCSA*, 13(1):1–16, 2016.